ЧИСЛЕННЫЕ МЕТОДЫ ОБРАБОТКИ СИГНАЛОВ В УСЛОВИЯХ ПОМЕХ

учебное пособие для студентов вузов

И. В. Сысоев

Саратов, 2019

УДК 621.396.1, 519.6
ББК 32.811, 22.19
С95

Сысоев И.В.
С95 Численные методы обработки сигналов в условиях помех. - Учебное пособие для студентов вузов. – ООО Издательство «КУБиК», 2019. – 58 с.

ISBN 978-5-91818-638-1

УДК 621.396.1, 519.6
ББК 32.811, 22.19

Учебное издание

Сысоев И.В.

Численные методы обработки сигналов в условиях помех

Подписано в печать 22.11.19 г. Формат 60x84 1/16. Бумага офсетная.
Шрифт Times New Roman. Усл. печ. л. 3,37.
Тираж 100 экз. Заказ № 2322.
ООО Издательство «КУБиК»
410056, г. Саратов, ул. Чернышевского, 94 а
тел.: (8452) 60-33-20

ISBN 978-5-91818-638-1

Оглавление

Введение

В книге рассматриваются известные численные методы, актуальные для задач обработки зашумлённых сигналов. Пособие строится по принципу от простого к сложному. В первой, вводной главе рассматривается задача решения нелинейных алгебраических уравнений, фигурирующая как вспомогательная при решении дифференциальных уравнений, аппроксимации зависимостей, преобразовании сигналов. Во второй и третьей главах рассматриваются различные подходы к анализу линейных систем: поиск определителя матрицы, определение её ранга, QR-разложение, решение систем линейных уравнений и задачи на наименьшие квадраты. Эти методы напрямую используются в нелинейной динамике и статистической радиофизике при анализе динамических и стохастических моделей в виде дифференциальных уравнений, в том числе для определения устойчивости режимов, аппроксимации экспериментальных зависимостей, реконструкции моделей по временным рядам. Значительное внимание уделяется борьбе с шумами измерений.

Четвёртая и пятая главы посвящены непосредственно задачам аппроксимации временных рядов с использованием полиномов и кубических сплайнов. Наиболее полезны эти подходы для интерполяции при редкой выборке, а также для работы с неэквидистантными сигналами. Шестая глава посвящена различным подходам к численному дифференцированию. Численное дифференцирование — актуальный подход для косвенного измерения; оно также часто используется при реконструкции пространства состояний и аттрактора динамических систем. Большое внимание уделяется сопоставлению изложенных подходов к численному дифференцированию с точки зрения борьбы с шумами наблюдения.

В пособии подробно рассматривается специфика программной реализации представленных методов с учётом особенностей различных языков программирования. Пособие также содержит задания и рекомендации для каждого изложенного алгоритма.

Глава 1

Численное решение нелинейных уравнений

Уравнение с одним переменным в общем виде можно записать следующим образом:

$$f(x) = 0. \tag{1.1}$$

В школе учат решать линейные ($f(x) = kx + b$), квадратные ($f(x) = ax^2 + bx + c$), биквадратные ($f(x) = ax^4 + bx^2 + c$) уравнения, а также некоторые тригонометрические, где $f(x) = sin(ax+\phi_0)+b$ и др. Хорошо известны формулы для решения кубических уравнений (формула Кардано) с функцией вида $f(x) = x^3+px+q$ и уравнений четвёртой степени. Для некоторых более сложных уравнений также найдены решения, в том числе записанные через различные специальные функции: функции Бесселя, эллиптические функции, интеграл ошибок, гамма, дигамма и бета функции и др. Однако для большинства встречающихся в инженерной и научной практике уравнений найти решение даже с использованием специальных функций либо не удаётся, либо невозможно в принципе; это строго доказано, например, для функций f, представляющих собою полином общего вида степени 5 или выше, хотя отдельные классы уравнений с полиномивальными функциями высших порядков могут быть решаемы. В таком случае прибегают к численному решению.

Численное решение в отличие от аналитического не претендует на абсолютную точность, а только на близость найденного корня к истинному значению в некотором смысле [3]. Численное решение даёт только один корень, не позволяя узнать, сколько всего корней имеет уравнение. Численное решение проводят на некотором отрезке или возле некото-

рой начальной (стартовой) догадки, т. е. обладая априорными знаниями о положении, существовании и единственности корня.

1.1 Метод половинного деления

Пусть на отрезке $[a; b]$ имеется ровно 1 корень уравнения (1.1). Тогда можно построить следующий алгоритм поиска корня.

1. Находится середина отрезка, $c = (a + b)/2$.

2. Вычисляются значения функции f в начале отрезка и в середине: $f_1 = f(a)$ и $f_2 = f(c)$ (можно построить алгоритм и иначе, используя конец и середину).

3. Если оказывается, что $f_1 \cdot f_2 > 0$ (то есть они одного знака), значит корень находится на второй половине отрезка, тогда начало отрезка перемещается в середину: $a := c$. Иначе — корень на второй половине или в середине отрезка, следовательно, конец отреза перемещается в середину: $b := c$. В любом случае длина отрезка сокращается вдвое.

4. Если длина отрезка всё ещё больше наперёд заданного значения ε, пункты с 1 по 3 повторяются, то есть их необходимо поместить в цикл с условием.

Таким образом, основным элементом алгоритма является цикл с условием, в котором и производятся все необходимые вычисления. В качестве решения годится любая точка итогового отрезка, как правило, используют его середину $c = (a + b)/2$. Вместо условия $(b - a) < \varepsilon$ возможна проверка альтернативного: $\left|f(\frac{a+b}{2})\right| < \delta$. В качестве ε или δ задают малое значение, согласованное с точностью вычислений (например, точность расчётов для 64-битного действительного аргумента составляет порядка 16 десятичных цифр).

Метод деления отрезка пополам имеет то важное на практике преимущество, что алгоритм гарантированно сходится, даже если на отрезке $[a; b]$ не один, а любое нечётное число корней. Однако какой по порядку корень удаётся в таком случае получить — узнать нельзя. Скорость

сходимости алгоритма (число шагов n, за которое удаётся получить решение с требуемою точность) пропорциональна вдоичному (поскольку деление происходит пополам) логарифму точности: $n \sim log_2(1/\varepsilon)$.

Метод можно и следует реализовывать как функцию или процедуру без использования глобальных переменных вообще. Такая функция в программировании называется *чистой*. В простейшем случае входными её параметрами являются a, b и ε (или δ), выходными — значение корня и, при необходимости, число шагов n, за которые достигается корень с требуемою точностью. Функцию f следует реализовать как функцию с одним входным и одним выходным параметром. Если способности программиста и специфика используемого языка программирования позволяют (например, в Питоне, Окамле или Хаскеле), саму функцию также следует передавать как входной параметр, тогда параметров будет четыре, причём по традиции функцию нужно передавать первою. В Паскале несложно объявить функциональный тип и передать функцию как переменную этого типа.

1.2 Метод Ньютона (касательных)

Пусть есть стартовая догадка для корня, которую обозначим x_0. Иначе: если известно, что корень лежит на отрезке $[a; b]$, положим $x_0 = (a+b)/2$. Тогда, используя приближении функции f в виде ряда Тейлора и ограничившись только нулевым и первым слагаемым, имеем:

$$f(x) \approx f(x_0) + (x - x_0)f'(x_0), \tag{1.2}$$

где $f'(x_0)$ — производная функции f в точке x_0. Тогда можно построить рекуррентное соотношение для нахождения корня:

$$x_{i+1} = x_i - \frac{f(x_0)}{f'(x_0)}, \tag{1.3}$$

где i — номер шага алгоритма. Величина $\Delta_i = \frac{f(x_i)}{f'(x_i)}$ имеет смысл поправки к оценке клорня на i-том шаге. Вычисления по формуле (1.3) следует прервать, если наступит условие $|\Delta_n| < \varepsilon$, то есть если поправка к корню стала очень мала. Можно использовать альтернативное условие $|f(x_n)| < \delta$, в точности совпадающее с аналогичным условием, используемым в методе деления отрезка пополам.

Основным элементом алгоритма является цикл с условием. При этом хранить всю последовательность x_i не требуется, хотя и не запрещается; достаточно иметь только две переменные: для текущего i-того значения и для нового — $(i+1)$-ого. Следует реализовать f и её производную f' как функции. Для этого производную нужно найти аналитически (это всегда реализуемо). Весь метод следует реализовать в виде функции или процедуры, на вход получающей x_0 (или границы отрезка a и b), точность расчётов ε или δ в зависимости от выбранного подхода и, если возможно, две функции: f и f' (они передаются первыми). Если f и f' передаются как аргументы, при реализации на Паскале достаточно объявить один процедурный тип, общий для них обеих, поскольку тип и число аргументов и тип результата у них одинаковые. На выход следует выдать оценку корня x_n и, если требуется, число шагов n, за которые корень был рассчитан. От использования глобальных переменных следует полностью отказаться.

В отличие от метода деления отрезка пополам, сходимость метода Ньютона не гарантирована в общем случае. То есть может так получиться, что в результате итерирования оценка корня начнёт быстро удаляться от истинного значения и за небольшое число шагов убежит на бесконечность в машинном смысле (величина оценки станет не представима в рамках используемого типа данных с плавающею точкою). Однако зато для типичных гладких функций f скорость сходимости метода Ньютона существенно выше (требуемое число шагов при той же точности меньше), чем метода половинного деления.

Задания

Для тестирования методов вам необходимо придумать тестовое уравнение. Лучше всего взять функцию f в виде полинома 3-го или 4-го порядка, либо несложной тригонометрической функции. Чтобы легко было знать расположение всех корней, полиномиальную функцию можно задать в виде произведения одночленов вида: $f(x) = (x - x_{r1})(x - x_{r2})(x - x_{r3})$, где x_{r1}, x_{r2} и x_{r3} суть корни.

1. Реализуйте метод деления отрезка или метод Ньютона пополам в виде функции, получающей на вход границы отрезка и параметр,

отвечающий за точность вычислений, а на выходе выдающей оценку корня. Возможны 4 варианта задания:

   (a) метод половинного деления с контролем точности по размеру отрезка: $(b - a) < \varepsilon$,

   (b) метод половинного деления с контролем точности по значению функции: $\left|f(\frac{a+b}{2})\right| < \delta$,

   (c) метод Ньютона с контролем точности по размеру приращения: $|\Delta_n| < \varepsilon$,

   (d) метод Ньютона с контролем точности по значению функции: $|f(x_n)| < \delta$.

2. Реализуйте метод половинного деления или метод Ньютона, дополнительно рассчитав число шагов, за которое получен результат. Чтобы выдать 2 числа из функции, используйте запись или кортеж. Варианты задания аналогичны представленным в п. 1.

3. Реализуйте метод деления отрезка пополам или метод Ньютона, передав функции f и f' (только для метода Ньютона) как входные параметры.

4. Реализуйте оба метода с контролем точности по значению функции и постройте зависимость числа итераций n, необходимых для получения корня, от точности вычислений. Точность задавайте как $\delta = \left(\frac{1}{2}\right)^k$. Удобнее может быть строить не зависимость $n(\delta)$, а зависимость $n(k)$. Убедитесь, что: во-первых, скорость сходимости метода половинного деления в логарифмическом масштабе линейно зависит от точности; во-вторых, скорость сходимости метода Ньютона существенно выше, чем метода половинного деления.

Глава 2

Линейные системы

2.1 Расчёт определителя матрицы

Расчёт определителя часто необходим для многих прикладных задач, в частности, для определения ранга матрицы, доказательства существования решения задач на наименьшие квадраты при аппроксимации и интерполяции сигналов, для определения мультипликаторов при исследовании решений дифференциальных уравнений на устойчивость [10].

Задача формулируется следующим образом (подробнее см., например, [2, 6]). Пусть дана матрица $\hat{A}$ размера $L \times L$, будем для определённости считать, что $L \geqslant 2$. Нужно найти её определитель D. Для $L = 2$ эта задача решается элементарно:

$$D(A) = a_{0,0}a_{1,1} - a_{0,1}a_{1,0}. \tag{2.1}$$

Для случая $L = 3$ также широко известно решение:

$$\begin{aligned} D(A) = a_{0,0}a_{1,1}a_{2,2} + a_{0,1}a_{1,2}a_{2,0} + a_{1,0}a_{2,1}a_{0,2} - \\ a_{0,2}a_{1,1}a_{2,2} - a_{0,1}a_{1,0}a_{2,2} - a_{0,0}a_{1,2}a_{2,1}. \end{aligned} \tag{2.2}$$

Как видно, при $L = 2$ мы имеем 2 слагаемых, состоящих из произведения 2 элементов матрицы каждое. Для $L = 3$ — уже 6 слагаемых, состоящих из произведений 3 элементов. Можно показать, что в произвольном случае будет $L!$ слагаемых, состоящих из L элементов каждое. Таким образом, уже для $L = 5$ получается 120 слагаемых, т. е. аналитически выписать такое количество членов очень непросто. На практике бывает необходимо рассчитывать определители матриц размером 20×20, 50×50 и даже 200×200.

2.1.1 Алгоритм расчёта через миноры

Чтобы рассчитать определитель матрицы произвольного размера, задачу сводят к вычислению определителей матриц на 1 меньшего размера по формуле (2.3)

$$D(A) = \sum_{j=0}^{L-1} \delta_{i,j} D(M_{i,j}) a_{i,j} \tag{2.3}$$

где $\delta_{i,j} = (-1)^{i+j}$, M_i — матрица, полученная вычёркиванием i-той строки и j-того столбца, где $0 \leqslant i \leqslant L-1$, то есть подойдёт любая строка. Можно аналогично произвести разложение и по столбцу, то, что всё равно по строке или по столбцу раскладывать, следует из правила, что определитель матрицы $\hat{A}$ и определитель транспонированной матрицы $\hat{A}^T$ равны. Этот процесс повторяют рекурсивно, выражая $D(M_{i,j})$ через их миноры до тех пор, пока не достигнут матрицы размером 2×2, определитель которой считают по формуле (2.1). Для простоты будем всегда разложение проводить по нулевому столбцу: $j = 0$.

Алгоритм кратко можно сформулировать следующим образом.

- Создаётся рекурсивная функция, принимающая на вход единственный параметр — матрицу (двумерный массив) $\hat{A}$. Размер матрицы L в большинстве современных языков дополнительно передавать не нужно, его можно найти при помощи встроенных средств (единственное исключение — это язык Си).

- После определения размера массива нужно сделать проверку: если $L = 2$, считаем определитель по формуле (2.1), иначе переходим к следующим пунктам. Это две ветви алгоритма: нерекурсивная и рекурсивная. Обратите внимание, в любом рекурсивном алгоритме должна быть нерекурсивная ветвь, иначе расчёты никогда не закончатся!

- Выделяем память под новую матрицу $\hat{M}$ размером $(L-1) \times (L-1)$. Полагаем $\delta = 1$, $D = 0$ как начальные значения. D будет использоваться для накопления суммы, δ будет менять знак на каждом шаге.

- Запускаем цикл по минорам (i от 0 до $L-1$). Для этого на каждой итерации этого цикла нужно заполнить элементы матрицы $\hat{M}$ по

следующему правилу: $m_{p,q} = a_{p,q+1}$, если $p < i$ и $m_{p,q} = a_{p+1,q+1}$ если $p \geqslant i$. Если разложение происходит не по нулевой и не по последней строке (столбцу), придётся писать четыре условия и собирать матрицу $\hat{M}$ из четырёх частей. После заполнения матрицы $\hat{M}$, всё ещё внутри цикла нужно вызвать нашу рекурсивную функцию саму из себя и, получив значение определителя $\hat{M}$, добавить $\delta D(M)a_{i,0}$ к сумме D. Наконец, всё ещё в цикле, нужно поменять знак δ.

У этой процедуры возможны варианты. В частности, можно вычислять δ на каждом шаге как $\delta = (-1)^j$. При формировании матрицы $\hat{M}$ можно копировать все элементы в двух вложенных циклах, а можно использовать вместо внутреннего цикла встроенные в язык операции для копирования массивов, например, `copy` в Паскале, срезы в D и Go. Можно даже вовсе обойтись без циклов, если язык поддерживает двумерные срезы, как, например, Фортран, Питон с использованием `numpy` и D с использованием `mir.ndslice`. Вообще, физическое создание всех строк матрицы $\hat{M}$ не является необходимым, поскольку в алгоритме они не меняются. Если можно обеспечить срезы–ссылки, этим следует воспользоваться.

Основную трудность представляет осознание рекурсивности программируемой функции. Студенты часто не могут представить, как вызвать функцию саму из себя. Вместо этого они начинают писать отдельные функции для подсчёта определителя 2×2, для подсчёта определителя 3×3 и так далее. Этого категорически нельзя делать: функция должна быть одна. При этом среда исполнения сама создаст несколько экземпляров нашей функции, которые будут как бы выполняться одновременно (в действительности все экземпляры, кроме текущего, будут стоять на месте и ждать его результата).

Вторая часто распространённая ошибка — использование глобальных переменных. Если в обычных процедурах их использование далеко не всегда влечёт проблемы и ошибки, то в рекурсивной — гарантированно приведёт к трагическим последствиям, поскольку несколько одновременно на разных этапах исполняемых экземпляров функции будут менять одну и ту же переменную. В результате вероятно зависание аналогичное зацикливанию или досрочный выход из тела функции с неверным результатом.

Наконец, типична ошибка при формировании матрицы $\hat{M}$. Нужно понимать, что все L миноров одинаковы по размеру и используются последовательно, поэтому не нужно создавать L матриц, можно повторно использовать одну и ту же. Но при этом они отличаются по содержанию, поскольку получены вычёркиванием различных строк. Поэтому заполнять их нужно для каждого минора заново.

Задания на расчёт определителя через миноры

Реализуйте расчёт определителя через миноры в виде функции в соответствии с предложенным алгоритмом. Сделайте проверочные матрицы размером 2×2, 3×3 и 4×4. Рассчитайте для них определитель самостоятельно: для первых двух по формулам (2.1) и (2.2), матрицу 4×4 для этого разложите на миноры. Для уменьшения количества аналитических вычислений можно в матрице 4×4 задать один из элементов верхней строки равным нулю, тогда определитель одного из миноров считать не придётся. Напишите программу, использующую вашу функцию и проверьте правильность расчётов. Матрицы читайте из файла.

Дополнительные задания:

1. Проведите тестирование производительности метода. Для этого перебирайте L в цикле в широких пределах, например, от 2 до 20. При каждом L создавайте большое число случайных матриц, например 100 или 500 штук. Для каждой матрицы производите расчёт определителя и засекайте встроенными средствами время, требуемое для расчёта определителя всех 500 матриц (время, необходимое для генерации случайных матриц, не следует включать в него). Постройте график зависимости среднего времени расчётов одного определителя от размера матрицы L.

2. Заметим, что i-ый минор отличается от $(i-1)$-ого только одною строкою с номером $(i-1)$. Например, для матрицы 4×4 список строк, попавших в каждый минор, будет выглядеть следующим образом: В таком случае переписывать всю матрицу на каждом шаге не обязательно, достаточно только поменять $i-1$-ую строчку. Модифицируйте алгоритм на основе такого подхода.

3. Объедините результаты, полученные в двух предыдущих заданиях. Постройте кривые для обеих реализаций метода на одном графике.

0 из 4: 1, 2, 3
1 из 4: 0, 2, 3
2 из 4: 0, 1, 3
3 из 4: 0, 1, 2

Альтернативное основное задание: реализуйте алгоритм для разложения не по нулевому, а по произвольному j-тому столбцу. Номер столбца задаётся как входной параметр. Учтите, что функция рекурсивная с уменьшающимся размером матрицы и поэтому на определённом этапе ваш номер будет слишком велик (такого столбца у очередной уменьшенной матрицы нет), в этом случае используйте 0-ой столбец. Покажите, что результат не зависит от выбора j с точностью вычислений.

Дополнительное задание: осуществляйте автоматический выбор номера столбца, по которому проводить разложение. Всегда используйте столбец, имеющий самую большую норму $S_j^2 = \sum_{i=0}^{L-1} a_{i,j}^2$. В этом случае передавать номер столбца в функцию в качестве параметра не нужно, он будет определяться внутри. Такой подход обычно позволяет уменьшить погрешность вычислений.

2.2 Расчёт определителя путём эквивалентных преобразований

К эквивалентным преобразованиям матрицы $\hat{A}$ относят такие её преобразования, как: перестановка строк и вычитание из одной строки другой, умноженной на ненулевое число. В результате этих преобразований можно получить другую матрицу, эквивалентную исходной в том смысле, что её определитель будет отличаться только знаком (если было выполнено нечётное число перестановок строк) или совсем не отличаться.

Для некоторых видов матриц рассчитать определитель гораздо проще, чем в общем случае. Определитель верхней треугольной матрицы (2.4) — в формуле нулевые элементы обозначены как 0, а ненулевые как x — равен просто произведению диагональных элементов главной диа-

гонали.

$$\hat{A} = \begin{bmatrix} \mathrm{x} & \mathrm{x} & \mathrm{x} & \mathrm{x} & \mathrm{x} \\ 0 & \mathrm{x} & \mathrm{x} & \mathrm{x} & \mathrm{x} \\ 0 & 0 & \mathrm{x} & \mathrm{x} & \mathrm{x} \\ 0 & 0 & 0 & \mathrm{x} & \mathrm{x} \\ 0 & 0 & 0 & 0 & \mathrm{x} \end{bmatrix} \tag{2.4}$$

Таким образом, идея подхода состоит в том, чтобы эквивалентными преобразованиями свести матрицу к верхней треугольной форме (2.4).

2.2.1 Простейший алгоритм расчёта определителя

Для этого нужно последовательно вычитать верхние строки из нижних, обнуляя один столбец за другим. Простейший вариант алгоритма можно представить следующим образом:

- Во внешнем цикле $k = 0, \ldots, L-2$,
- для каждой i-той строки, $i > k$, находит находим отношение $g = a_{i,k}/a_{k,k}$,
- домножаем каждый элемент k-той строки на g и вычитаем из элемента i-той строки с тем же номером: $a_{i,j} := a_{i,j} - g a_{k,j}$, при этом достаточно преобразовывать только элементы с $j \geqslant k$, поскольку элементы с меньшими индексами во всех строках ниже k-той уже равны нулю.
- По окончании преобразования матрицы перемножаем диагональные элементы.

Такой простой алгоритм будет работать, если только на диагонали не появится ноль или величина, численно к нему очень близкая, в результате чего вычисление g окажется невозможно или произойдёт с ошибкою. Чтобы решить эту проблему, необходимо переставлять строки в случае, если на диагонали оказался нулевой элемент. Менять можно только строку с лежащею ниже: в выше лежащих строках слева ненулевые элементы. При этом нужно считать число перестановок, поскольку каждая перестановка меняет знак определителя. Таким образом, нужно дополнить алгоритм.

2.2.2 Алгоритм с перестановкою строк

- Пусть $D = 1$ — первоначальный знак определителя.
- Во внешнем цикле $k = 0, \ldots, L - 2$.
- Проверяем, что $|a_{k,k}| \leqslant \varepsilon$. Если да:
  - $n := k + 1$;
  - запускаем цикл с условием, пока $|a_{n,k}| \leqslant \varepsilon$, увеличиваем счётчик n, таким образом будет найден номер первой строки, k-тый элемент которой годится для замены $a_{k,k}$;
  - переставляем местами n-ную и k-тую строки,
  - умножаем D на -1.
- Для каждой i-той строки, $i > k$, находит находим отношение $g = a_{i,k}/a_{k,k}$.
- Домножаем каждый элемент k-той строки на g и вычитаем из элемента i-той строки с тем же номером: $a_{i,j} := a_{i,j} - g a_{k,j}$, при этом достаточно преобразовывать только элементы с $j \geqslant k$, поскольку элементы с меньшими индексами во всех строках ниже k-той уже равны нулю.
- По окончании преобразования матрицы перемножаем диагональные элементы и умножаем на D.

Важно отметить две часто встречающиеся ошибки. Во-первых, умножать на g и вычитать k-тую строку из всех ниже лежащих нужно вне зависимости от того, переставлялись строки или нет. Во-вторых, переставлять строки нужно только после того, как найден такой номер n, что стоящий в k-том столбце в n-ной строке элемент отличен от нуля с заданною точностью. Не нужно переставлять строки на каждом шаге цикла при поиске n. Т. е. для каждого k сама перестановка может быть только одна, причём она случается тогда и только тогда, когда элемент $a_{k,k}$ на k-том шаге был близок к нулю.

При перестановке строк предпочтительнее переименовать соответствующие массивы–указатели. Если это невозможно, например, потому,

что матрица задана в виде статического массива, и перестановка осуществляется копированием значений элементов, достаточно менять номера с k-того, поскольку элементы с меньшими номерами являются нулевыми.

Усложнённый алгоритм, тем не менее, не предусматривает одну существенную возможность: что делать, если найти строку для перестановки не удалось? Т. е. все строки ниже k-той также содержат в k-том столбце нули. Такое обязательно происходит, если определитель в действительности равен нулю. Если забыть об этой возможности, при вычислении n произойдёт ошибка выхода за границы диапазона. Поэтому обязательно нужно контролировать, не превысило ли n значение $L-1$ и если да, дальше можно не считать: это значит, что определитель равен 0.

Задания на расчёт определителя эквивалентными преобразованиями матрицы

- Реализуйте простейший алгоритм расчёта определителя через преобразование матрицы.
- Реализуйте алгоритм расчёта определителя с перестановкою строк.

Дополнительное задание. Проведите тестирование метода на производительность с матрицами размера от 3 до 20. С матрицею каждого размера проводите тестирование 100 раз, засекайте суммарное время. Постройте график зависимости времени от размера матрицы.

2.3 Метод Гаусса решения систем линейных уравнений

Метод Гаусса позволяет решить систему линейных уравнений, представленную в виде:

$$\hat{A}\mathbf{c} = \mathbf{b}, \tag{2.5}$$

где $\hat{A}$ — квадратная матрица размером $L \times L$ с ненулевым определителем, $\mathbf{b} = (b_0, b_1, \ldots, b_{L-1})$ — «вектор свободных членов» длиною L и $\mathbf{c} = (c_0, c_1, \ldots, c_{L-1})$ — искомые значения корней, тоже L штук [2, 6].

Идея метода состоит в том, чтобы последовательно выражать неизвестные: из одного уравнения выражаем c_0 через другие, затем из одного из оставшихся — c_1 и так далее, пока не получится последнее уравнение, содержащее только c_{L-1}, откуда его можно вычислить.

Численно метод сводится к преобразованию матрицы $\hat{A}$ к треугольному виду (2.4) с одновременным преобразованием вектора $\mathbf{b}$. Преобразование производится точно так же, как при вычислении определителя при добавлении вдух пунктов:

1. При вычитании строк $a_{i,j} := a_{i,j} - g a_{k,j}$ нужно также преобразовывать $b_i := b_i - g b_k$, при этом нужно помнить, что одно значение b_i соответствует всей строке $\mathbf{a}_i$, то есть вычисления следует проводить вне цикла по j.

2. Следует запоминать порядок перестановки строк, иначе в случае перестановок порядок корней будет отличаться от исходного. Для этого нужно сделать массив длины L из целых чисел, заполнить его значениями от 0 до $L-1$ и в случае перестановки k-той и n-ной строк менять местами k-тый и n-ный элементы.

После окончания преобразования матрицы можно вычислять корни $\mathbf{c}$. Из последнего уравнения можно сразу получить последний корень. Далее нужно подставить его в предпоследнее и получить предпоследний и т. д. Что проделать это, нужно организовать цикл с уменьшением индекса i от $L-1$ до 0, при этом внутри нужно сделать ещё один цикл по номерам от $i+1$ до $L-1$, в котором будет накапливаться сумма $\sum_{j=i+1}^{L-1} a_{i,j} c_j$. Поскольку в большинстве языков программирования цикл не будет выполняться, если нижний индекс больше верхнего, отдельно считать самый нижний корень нет необходимости.

Если имела место перестановка строк, корни будут вычислены в неверном порядке. В таком случае после их расчёта нужно переупорядочить корни, для чего проще всего создать новый массив и скопировать туда значения из массива $\mathbf{c}$ в соответствии с массивом номеров, созданным ранее.

Задания на метод Гаусса

- Реализуйте метод Гаусса без перестановки строк в виде отдельной процедуры или функции. Ввод матрицы и вектора свободных

членов осуществляйте в главной программе из файла или файлов. Число строк определяйте автоматически (без необходимости задавать отдельно).

- Реализуйте метод Гаусса с возможностью перестановки строк в виде отдельной процедуры или функции. Ввод матрицы и вектора свободных членов осуществляйте в главной программе из файла или файлов. Число строк определяйте автоматически (без необходимости задавать отдельно).

2.4 Сведение задачи на наименьшие квадраты к методу Гаусса

Пусть имеются две выборки $\{x_n\}_{n=0}^{N-1}$ и $\{y_n\}_{n=0}^{N-1}$ такие, что каждому x_n соответствует y_n. Нужно произвести аппроксимацию (приближение) значений выборки $\{y_n\}_{n=0}^{N-1}$ с помощью некоторой функции от $\{x_n\}_{n=0}^{N-1}$. Пусть такая функция обозначается f и представляет собою разложение на базисные, заранее известные функции с неизвестными коэффициентами:

$$f(x) = \sum_{i=0}^{P-1} c_i \phi_i(x). \tag{2.6}$$

Задача наименьших квадратов заключается в минимизации функции (2.7) [4], представляющей собою сумму квадратов разностей между истинным значением наблюдаемой y_n и её же аппроксимированным (приближенным) значением, представляющим собою сумму базисных функций ϕ_i, взятых от x_i с неизвестными коэффициентами c_i.

$$S(\mathbf{c}) = \sum_{n=0}^{N-1} \left(\sum_{i=0}^{P-1} c_i \phi_i(x_n) - y_n \right)^2 \to \min \tag{2.7}$$

Целью подхода, фактически, является именно расчёт вектора коэффициентов $\mathbf{c}$. Если продифференцировать $S(\mathbf{c})$ по всем c_i, получится система из P линейных уравнений с P неизвестными вида (2.5), где элементы

матрицы $\hat{A}$ и вектора свободных членов $\mathbf{b}$ задаются соотношением (2.8).

$$\begin{aligned} a_{i,j} &= \sum_{n=0}^{N-1} \phi_i(x_n)\phi_j(x_n), \\ b_i &= \sum_{n=0}^{N-1} \phi_i(x_n)y_n. \end{aligned} \tag{2.8}$$

Вычислив значения $a_{i,j}$ и b_i, можно передавать матрицу $\hat{A}$ и вектор $\mathbf{b}$ в процедуру метода Гаусса, на выходе которой будут значения искомых коэффициентов $\mathbf{c}$.

Кроме самих коэффициентов часто необходимо вычислить среднеквадратичную ошибку аппроксимации $\varepsilon^2 = \frac{1}{N-P}S(\mathbf{c})$.

Задания на метод наименьших квадратов

Реализуйте расчёт коэффициентов $\mathbf{c}$ методом наименьших квадратов путём сведения задачи к методу Гаусса для различных видов базисных функций:

- степенных полиномов: $\phi_i(x) = x^i$, $i = 0, \ldots, P-1$;
- для тригонометрических полиномов: $\phi_0 = \frac{1}{2}$, $\phi_{2i-1}(x) = \sin(i\omega x)$, $\phi_{2i}(x) = \cos(i\omega x)$, $i = 1, \ldots, (P-1)/2$, P нечётно, $\omega = \frac{2\pi}{T}$, где T — период сигнала [9].

Для тестирования метода сгенерируйте проверочную выборку по соответствующему закону. Например, для степенного полинома в виде кубической параболы на отрезке $[-3; 3]$ с шагом 0.01, для тригонометрического полинома в виде суммы синусоид и косинусоид на разных частотах, кратных основной частоте (можно выбрать $\omega = 2\pi$, отрезок $[-3, 3]$, шаг 0.01). Сравните исходные коэффициенты с восстановленными, они должны совпасть с точностью вычислений, например, при использовании двойной точности результаты будут отличаться на $\sim 10^{-16}$ истинного значения.

Дополнительные задания.

1. Реализуйте расчёт ошибки аппроксимации ε^2.

2. Проведите тестирование точности расчёта коэффициентов в зависимости от погрешностей измерения и длины выборки. Для этого добавьте к тестовой выборке $\{y_n\}_{n=0}^{N-1}$ нормально распределённые случайные числа с нулевым средним и небольшим среднеквадратичным отклонением (например, 0.05). Полученные по такой зашумлённой выборке значения коэффициентов будут неточными. Увеличьте размер выборки N вчетверо (можно и вдесятеро), при этом отрезок, на котором она генерируется, сохраняйте, но уменьшайте шаг. Проделайте это несколько раз. Проследите, как при увеличении N возрастает точность определения коэффициентов.

Глава 3

QR-разложение

Любую прямоугольную матрицу $\hat{A}$ размером $N \times L$ (то есть N строк и L столбцов, $N \geqslant L$) можно единственным образом представить в виде произведения двух других матриц, обозначаемых $\hat{Q}$ и $\hat{R}$, имеющих специальные свойства (на самом деле это не совсем так: столбцы получившейся матрицы $\hat{Q}$ могут иметь различный знак в зависимости от техники разложения).

$$\hat{A} = \hat{Q}\hat{R} \tag{3.1}$$

Матрица $\hat{Q}$ размером $N \times L$ (размер совпадает с размером матрицы $\hat{A}$) является ортонормированною, то есть если рассмотреть любую пару её столбцов как два N-мерных вектора, их скалярное произведение будет равно 0, если только эти столбцы разные. При этом норма (длина в евклидовом смысле — корень и суммы квадратов элементов) каждого столбца равна 1:

$$\hat{Q}_i^T \cdot \hat{Q}_j^T = 0 \qquad \forall i \neq j \tag{3.2}$$

$$\|\hat{Q}_i^T\| = \hat{Q}_i^T \cdot \hat{Q}_i^T = 1 \tag{3.3}$$

Здесь для i-того столбца матрицы $\hat{Q}$ используется обозначение $\hat{Q}_i^T$, поскольку мы условились, что i-тый элемент матрицы есть её i-тая строка. В таком случае i-тый столбец матрицы $\hat{Q}$ есть i-тая строка $\hat{Q}^T$ — транспонированной матрицы. Из условий (3.2, 3.3) следует важное соотношение (3.4):

$$\hat{Q}^T\hat{Q} = \hat{E}, \tag{3.4}$$

где $\hat{E}$ — единичная матрица.

Матрица $\hat{R}$ — верхняя треугольная имеет размер $L \times L$ (L строк и L столбцов).

Следует отметить, что все указанные соотношения работают, если ранг матрицы $\hat{A}$ равен L, то есть матрица $\hat{A}$ не содержит линейно зависимых столбцов (ни один столбец нельзя получить как линейную комбинацию двух или нескольких других, отличных отнего).

3.1 Вычисление матриц Q и R

Существует несколько методов вычисления матрицы $\hat{Q}$: процесс Грама–Шмидта, вращение Гивенса, отображение Хаусхолдера. Далее разберём процесс Грама–Шмидта как самый простой алгоритм.

Для успешной реализации алгоритма нужно иметь две вспомогательные функции:

1. функция расчёта скалярного произведения двух векторов (массивов),

2. функция нормировки вектора (массива), вычисляющая его норму (можно с использованием предыдущей функции — как скалярное произведение вектора самого на себя) и делящая все элементы на корень из нормы, так что норма вектора оказывается равна 1.

Вторая функция должна изменять исходный вектор, а не создавать новый.

Основная часть алгоритма состоит в следующем:

1. Транспонируем матрицу $\hat{A}$, создав новую матрицу. В новой матрице $\hat{A}^T$ строки соответствуют столбцам исходной матрицы $\hat{A}$ [1].

2. Копируем матрицу $\hat{A}^T$ в матрицу $\hat{Q}^T$. Обязательно произведите полное копирование элементов, а не объявлеине новых массивов/указателей, указывающих на старые данные. Обе матрицы понадобятся в дальнейшем.

3. Нормируем нулевую строку $\hat{Q}^T$.

4. Для каждой последующей строки $\hat{Q}^T$ с номером k делаются следующие преобразования:

    - Для каждой строки $\hat{Q}_j^T$ с номерами $j < k$ находим скалярное произведение $\nu_{k,j} = \hat{Q}_j^T \hat{A}_k^T$ и вычитаем из вектора $\hat{Q}_k^T$

вектор $\nu_{k,j}\hat{Q}_j^T$, то есть из k-той строки j-тую строку, умноженную на $\nu_{k,j}$. Этим достигается ортогональность строки $\hat{Q}_k^T$ всем предыдущим стокам $\hat{Q}_j^T$.

- Полученный на предыдущем этапе вектор $\hat{Q}_k^T$ нормируется.

Теперь матрица $\hat{Q}^T$ готова.

Элементы матрицы $\hat{R}$, которые обозначим $r_{i,j}$, можно вычислить при готовой $\hat{Q}^T$ по следующей формуле:

$$r_{i,j} = \begin{cases} \hat{A}_i^T \hat{Q}_j^T, & j \geqslant i, \\ 0 & j < i. \end{cases} \tag{3.5}$$

3.2 Решение систем линейных уравнений и задачи на наименьшие квадраты с помощью QR-разложения

QR-разложение часто используется для решения задачи на наименьшие квадраты (2.7). Это связано с более высокою численною точностью такого подхода по сравнению с реализацией сведением к методу Гаусса [2, 6].

Чтобы использовать QR-разложение, необходимо построить так называемую «матрицу значений базисных функций» (3.6), которая содержит значения всех базисных функций ϕ_i для всех элементов ряда $\{x_n\}_{n=0}^{N-1}$.

$$\hat{\Phi} = \begin{bmatrix} \phi_0(x_0) & \phi_1(x_0) & \cdots & \phi_{P-1}(x_0) \\ \phi_0(x_1) & \phi_1(x_1) & \cdots & \phi_{P-1}(x_1) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_0(x_{N-1}) & \phi_1(x_{N-1}) & \cdots & \phi_{P-1}(x_{N-1}) \end{bmatrix} \tag{3.6}$$

Матрица $\hat{\Phi}$ имеет размер $N \times P$, где $P \leqslant N$, и является в данной задаче аналогом матрица $\hat{A}$ из формулы (3.1). Можно отметить, что целевая функция может быть следующим образом выражена через матрицу $\hat{\Phi}$: $S(\mathbf{c}) = \|\hat{\Phi}\mathbf{c} - \mathbf{y}\|$, тогда фомулу (2.7) можно переписать через матрицу $\hat{\Phi}$ следующим образом:

$$\|\hat{\Phi}\mathbf{c} - \mathbf{y}\| \to \min \tag{3.7}$$

Далее полезно рассмотреть вектор невязок (ошибок аппроксимации) $\{\xi\}_{n=0}^{N-1}$, длина которого, делённая на его размерность и есть ошибка аппроксимации ε^2. Вектор невязок выражается следующим образом через матрицу значений базисных функций:

$$\boldsymbol{\xi} = \hat{\Phi}\mathbf{c} - \mathbf{y} \tag{3.8}$$

Если подгонка коэффициентов прошла успешно, то вектор невязок $\boldsymbol{\xi}$ будет перпендикулярен всем столбцам матрицы $\hat{\Phi}_i^T$ (их скалярное произведение будет равно нулю). Это условие можно записать как (3.9):

$$\hat{\Phi}^T\boldsymbol{\xi} = \mathbf{0}, \tag{3.9}$$

где справа стоит нулевой вектор (массив нулей) длины P. Если далее применить к матрице $\hat{\Phi}$ QR-разложение, записав $\hat{\Phi} = \hat{Q}\hat{R}$, и воспользоваться свойствами преобразования траспонирования, в соответствии с которыми $\hat{\Phi}^T = \hat{R}^T\hat{Q}^T$, а вектор невязок $\boldsymbol{\xi}$ расписать по формуле (3.8), получим следующее соотношение:

$$\hat{R}^T\hat{Q}^T\hat{Q}\hat{R}\mathbf{c} = \hat{R}^T\hat{Q}^T\mathbf{y} \tag{3.10}$$

Используя соотношение $\hat{Q}^T\hat{Q} = \hat{E}$ и свойства единичной матрицы, а также поделив уравнение на $\hat{R}^T$ слева, получим окончательное соотношение:

$$\hat{R}\mathbf{c} = \mathbf{b}, \tag{3.11}$$

$$\mathbf{b} = \hat{Q}^T\mathbf{y}. \tag{3.12}$$

Соотношение (3.11) представляет собою систему линейных уравнений, аналогичную (2.5), но только матрица $\hat{R}$ в отличие от $\hat{A}$ уже верхняя треугольная, поэтому можно сразу приступать к выражению значений коэффициентов, начиная с последнего из них — c_{P-1}. Соотношение (3.12) используется для расчёта вектора правых частей в уравнении (3.11), для чего окажется полезна ранее написанная при реализации QR-разложения функция скалярного произведения, поскольку $b_i = \hat{Q}_i^T\mathbf{y}$, $i = 0 \ldots P-1$. Заметим также, что для метода наименьших квадратов нетранспонированная матрица $\hat{Q}$ оказывается ненужна, поэтому то, что при реализации QR-разложения мы всё время работали только с транспонированною матрицею даже удобно.

Решение системы линейных через QR-разложение является частным случаем задачи на наименьшие квадраты, когда $N = P$, то есть матрицы $\hat{\Phi}$ и $\hat{Q}$ квадратные [2].

Задания на QR-разложение

Расчёт матриц $\hat{Q}$ и $\hat{R}$

Напишите подпрограмму (процедуру или функцию), реализующую вычисление матриц $\hat{Q}$ и $\hat{R}$ для произвольной прямоугольной матрицы $\hat{A}$. Подпрограмма не должна использовать глобальные переменные. При реализации напишите функции (если они отсутствуют в стандартной библиотеке используемого языка программирования) для:

- вычисления скалярного произведения двух векторов (на входе — два массива, на выходе — число),
- нормировки вектора (единственный параметр — входной массив, который изменяется внутри процедуры).

Протестируйте подпрограмму на матрицах различного размера, прямоугольных и квадратных.

Дополнительное задание. Модифицируйте вашу подпрограмму так, чтобы можно было выбирать, подаётся ли на вход матрица $\hat{A}$ или уже транспонированная матрица $\hat{A}^T$, а также чтобы можно было вывести либо матрицу $\hat{Q}$, либо транспонированную матрицу $\hat{Q}^T$. Модификаторы реализуйте в виде логических параметров вашей подпрограммы; если в языке программирования предусмотрена такая возможность, задайте им значения по умолчанию.

Реализация метода наименьших квадратов и решение системы линейных уравнений через QR-разложение

Напишите подпрограмму (процедуру или функцию), которая реализует метод наименьших квадратов и решение системы линейных уравнений через QR-разложение, используя подпрограмму QR-разложения, написанную ранее. Подпрограмма не должна использовать глобальные переменные, на её вход подаётся матрица и вектор свободных членов, на выходе — массив коэффициентов (решение системы линейных уравнений). Проверьте задачу с матрицами различных размеров. Для проверки правильности решения можете воспользоваться методом Гаусса и реализацией МНК через него.

Дополнительное задание. Сгенерируйте последовательность по формуле, известной как логистическое отображение (3.13)

$$x_{n+1} = 1 - \lambda x_n^2 \tag{3.13}$$

Возьмите значение $x_0 = 0.1$ (можно взять и другие значения) и $\lambda = 1.56$ (можно взять другие значения, отличающиеся на несколько сотых). Сформируйте ряд $\{y_n\}_{n=0}^{N-1}$ путём сдвига ряда $\{x_n\}_{n=0}^{N-1}$ на единицу, т. е. по формуле $y_n = x_{n+1}$. С помощью метода наименьших квадратов восстановите значения коэффициентов $\mathbf{c}$ по рядам $\{x_n\}_{n=0}^{N-1}$ и $\{y_n\}_{n=0}^{N-1}$, учтите при этом, что ряды должны быть одинаковой длины, для чего от полученного по формуле (3.13) ряда нужно будет отбросить последний элемент. Базисные функции ϕ_i следует выбрать в виде степенных одночленов: $\phi_0(x) = 1$, $\phi_1(x) = x$, $\phi_2(x) = x^2$ и т. д. Длину ряда N можно использовать различную, например, 100 или 500. Если вы всё сделали правильно, вы получите $c_0 = 1$, $c_1 = 0$, $c_2 = -\lambda$, все последующие будут равны нулю.

Глава 4

Аппроксимация полиномами

Аппроксимация — замена одного объекта другим, в некотором смысле близким к исходному. В каком смысле близким, решает тот, кто проводит аппроксимацию. Например, для перевозки системных блоков компьютеров их можно аппроксимировать прямоугольными параллелепипедами (или в просторечии — кирпичами), а вот для изучения принципов их функционирования лучше подойдёт абстрактная модель машины фон Немана. Как правило, при аппроксимации часть свойств объекта теряются, но ингода возникают и новые. Есть два частных случая аппроксимации, распространённых при обработке временных рядов, дискретных последовательностей значений некоторой величины, упорядоченных во времени: интерполяция и экстраполяция.

Интерполяция — доопределение функции между моментами измерения. Экстраполяция — доопределение функции за пределами области измерения. Задача интерполяции чаще всего возникает в ситуации, когда значения функции определены только для некоторых значений аргумента, например, в результате измерения, но желательно знать её значения также в промежуточных точках, где измерение не проводилось. Пары значений аргумента и функции (x_n, y_n) называют узлами. Задача экстраполяции может возникнуть в аналогичной ситуации, например, если функция определена на некотором отрезке (скажем, в результате измерения) и нужно узнать её значения за его пределами.

Пусть есть две выборки $\{x_n\}_{n=0}^{N-1}$ и $\{y_n\}_{n=0}^{N-1}$. Необходимо построить степенной полином (4.1) такой, чтобы он проходил через все измененные

точки (x_n, y_n).

$$L(x) = \sum_{i=0}^{P-1} c_i x^i \tag{4.1}$$

Решение этой задачи возможно в общем случае, только если $N \leqslant P$. Вариант $N < P$ нас не устраивает, поскольку в таком случае существует бесконечно много интерполирующих полиномов и какой выбрать, не ясно. При $N = P$ полученное решение единственно.

4.1 Вычисление коэффициентов полиномов Лагранжа

Существуют различные подходы решению задачи интерполяции полиномами. Один из наиболее популярных — использование полиномов в форме (4.2, 4.3), предложенной Лагранжем.

$$L(x) = \sum_{i=0}^{N-1} y_i l_i(x) \tag{4.2}$$

$$l_i(x) = \prod_{j=0, j\neq i}^{N-1} \frac{x - x_j}{x_i - x_j} \tag{4.3}$$

Полиномы $l_i(x)$, суммою которых является интерполяционный полином Лагранжа вида (4.2), обладают по построению некоторыми важными свойствами:

1. являются полиномами степени $N - 1$, содержащими N коэффициентов,
2. $l_i(x_i) = 1$,
3. $l_i(x_j) = 1$ при $j \neq i$.

Использование формы записи (4.2, 4.3) для реальных расчётов неудобно, поскольку такой подход требует для каждого x вычислений порядка $O(N^2)$, в то время как форма (4.1) — только порядка $O(N)$. Поэтому если интерполирующий полином будет использоваться для вычисления во многих новых точках x, следует перейти от формы (4.2, 4.3)

к форме (4.1). Далее рассмотрим задачу о вычислении коэффициентов c_i.

Преобразуем формулу (4.3) к виду (4.4), введя обозначения для числителя и знаменателя.

$$l_i(x) = \frac{p_i(x)}{q_i} = \frac{\prod\limits_{j=0, j\neq i}^{N-1} (x - x_j)}{\prod\limits_{j=0, j\neq i}^{N-1} (x_i - x_j)} \tag{4.4}$$

В формуле (4.4) p_i — полином порядка $N-1$ с коэффициентом 1 при старшей степени, а q_i — просто число. Расчёт q_i не составляет проблем, поэтому далее будем рассматривать только расчёт коэффициентов полинома $p_i(x)$, который представим в виде (4.5).

$$p_i(x) = \sum_{j=0}^{N-1} \alpha_{i,j} x^j \tag{4.5}$$

Рассчитать коэффициенты $\alpha_{i,j}$ можно с помощью итеративной процедуры. Обозначим массив (вектор) коэффициентов полинома p_i как $\boldsymbol{\alpha}_i$. Рассмотрим сначала самый первый множитель в произведении в числителе формулы (4.4). Это двучлен первого порядка, который будем использовать в качестве заготовки, последовательно преобразуя его. Его коэффициенты можно представить как массив из N значений, в котором нулевой элемент равен $-x_0$ (или $-x_1$, если $i = 0$), первый — единице, а остальные — нулю.

Далее будем последовательно домножать этот полином на следующие линейные двучлены, соответствующим образом изменяя коэффициенты. Каждое домножение можно представить в виде трёх операций над массивом:

1. Домножение на x приводит к сдвигу всех коэффициентов вправо — значение, лежавшее в $\alpha_{i,0}$, перемещается в $\alpha_{i,1}$, значение, лежавшее в $\alpha_{i,1}$ — в $\alpha_{i,2}$ и так далее; последнее значение теряется, а $\alpha_{i,0}$ становится равным 0. В результате получается новый массив, который обозначим $\boldsymbol{\alpha}_{\rightarrow}$.

2. Домножение на $-x_j$ приводит к просто умножению всех элементов массива. Полученный массив обозначим $\boldsymbol{\alpha}_{*}$.

3. Сложение полученных на двух предыдущих этапах массивов поэлементно: $\boldsymbol{\alpha}_i = \boldsymbol{\alpha}_{\rightarrow} + \boldsymbol{\alpha}_{*}$.

Действия 1–3 повторяются для всех множителей из-под знака произведения в числителе формулы (4.4).

Этот алгоритм следует повторить для всех $p_i(x)$ при $i = 0, \ldots, N-1$, в результате чего получится матрица $\hat{A}$ коэффициентов $\alpha_{i,j}$. Используя её можно рассчитать коэффициенты c_i в формуле (4.1) следующим образом:

$$c_i = \sum_{i=0}^{N-1} \frac{\alpha_{i,j} y_i}{q_i}. \tag{4.6}$$

4.2 Интерполяция полиномами

После того, как коэффициенты полинома в форме (4.1) получены, можно приступать собственно к интерполяции, то есть расчёту значений полинома $L(x)$ при различных новых значениях x, которые обозначим как ξ_k, $k = 0, \ldots, M-1$.

Для расчёта значения полинома при произвольном x лучше всего написать функцию, которая будет принимать на вход массив коэффициентов $\mathbf{c}$ и значение x (передавать число элементов массива $\mathbf{c}$ как отдельный параметр нет смысла, так как его легко получить встроенными средствами почти в любом языке программирования за исключением C и C++ при неиспользовании стандартных контейнеров). Основным элементом функции должен быть цикл с обходом по всем элементам $\mathbf{c}$, подсчитывающий на очередной своей итерации $c_i x^i$.

Далее вызывайте вашу функцию для всех $\{\xi_k\}_{k=0}^{M-1}$ в цикле.

Задания на аппроксимацию полиномами

Задание на расчёт коэффициентов аппроксимирующего полинома

Основное задание. Реализуйте расчёт коэффициентов аппроксимирующего полинома описанными выше методами в виде функции или процедуры. На вход должны подаваться два массива: $\{x_n\}_{n=0}^{N-1}$ и $\{y_n\}_{n=0}^{N-1}$, на

выходе — один массив коэффициентов $\{c_i\}_{i=0}^{N-1}$. Испытайте вашу функцию расчёта коэффициентов на тестовых данных, сгенерированных вами самостоятельно описанным далее способом.

1. Составьте полином 4-го или 5-го порядка с ненулевым коэффициентом при старшей степени, значения всех коэффициентов придумайте самостоятельно.

2. Выберите 5 или 6 (в зависимости от степени полинома) значений аргумента, лежащие в пределах от -2 до 2 и рассчитайте вручную значения вашего полинома от этих аргументов.

3. Далее придуманные аргументы используйте как ряд $\{x_n\}_{n=0}^{N-1}$, а рассчитанные значения — как ряд $\{y_n\}_{n=0}^{N-1}$.

4. Убедитесь, что коэффициенты аппроксимирующего полинома, рассчитанные по сформированным таким образом данным, совпадают с придуманными вами изначально коэффициентами.

Дополнительное задание. Оптимизируйте решение с точки зрения вычислительной сложности. В частности, вынесите все общие блоки во внешние циклы, а расчёт величин типа x_i^j реализуйте в виде накопления произведения с использованием полученного на предыдущем этапе x_i^{j-1}.

Задание на интерполяцию

Задание выполняется после того, как выполнено задание на расчёт коэффициентов.

1. Получите или придумайте неравномерную выборку исходных значений $\{x_n\}_{n=0}^{N-1}$ — такую, для которой расстояние между соседними значениями не одинаково. x_n можно получить как случайные числа, либо придумать, поскольку большое N не требуется (лучше ограничиться значениями $N = 5$ или $N = 6$).

2. Далее используйте те же коэффициенты, что и в предыдущем задании, чтобы сгенерировать значения полинома $\{y_n\}_{n=0}^{N-1}$.

3. Рассчитайте коэффициенты, как в предыдущем задании.

4. Сгенерируйте новую выборку $\{\xi_n\}_{n=0}^{M-1}$, где $M = 100$, причём значения ξ_n представляют собою последовательно с одинаковым шагом возрастающие на отрезке $[x_0; x_{N-1}]$ числа. Запишите их в файл.

5. По новой выборке рассчитайте значения аппроксимирующего полинома $\{\eta_n\}_{n=0}^{M-1}$. Реализуйте этот расчёт в виде процедуры или функции, принимающей на вход два массива: массив коэффициентов $\mathbf{c}$ и массив значений $\boldsymbol{\xi}$, в которых нужно рассчитать полином. Результат — массив значений полинома $\boldsymbol{\eta}$.

6. Постройте график, на котором отложите исходные пары значений $\{(x_n, y_n)\}_{n=0}^{N-1}$ крупными точками, звёздочками или треугольниками, а значения $\{(\xi_n, \eta_n)\}_{n=0}^{M-1}$ — маленькими точками другого цвета.

Дополнительное задание. Возьмите те же значения $\{x_n\}_{n=0}^{N-1}$, что и в основном задании, но $\{y_n\}_{n=0}^{N-1}$ рассчитайте иначе, как $y_n = \sin(x_n)$. Далее используйте вашу функцию для расчёта коэффициентов, возьмите те же $\{\xi_n\}_{n=0}^{M-1}$ и рассчитайте $\{\eta_n\}_{n=0}^{M-1}$. Дополнительно рассчитайте $\zeta_n = \sin(\xi_n)$. Отложите на одном графике $y_n(x_n)$, $\eta_n(\xi_n)$ как в прошлом задании, а также $\zeta_n(\xi_n)$.

Сравните значения η_n, полученные в результате интерполяции, со значениями ζ_n на графике. Также рассчитайте среднеквадратичную ошибку аппроксимации $\varepsilon^2 = \sum_{n=0}^{M-1} (\eta_n - \zeta_n)^2$.

Задание на экстраполяцию

Аппроксимирующий полином можно использовать для экстраполяции за пределы области измерения. Для этого выполните задания предыдущего раздела, используя $\xi_n \notin [x_0; x_{N-1}]$. Можно предложить 3 варианта:

1. ξ_n равномерно распределены в диапазоне слева от x_0, например $\xi_n \in [2x_0 - x_{N-1}; x_0]$;

2. ξ_n равномерно распределены в диапазоне справа от x_{N-1}, например $\xi_n \in [x_{N-1}; 2x_{N-1} - x_0]$;

3. половина всех значений лежит слева, а вторая — справа.

Пронаблюдайте на графике, как для дополнительного задания, где $y_n = \sin(x_n)$, ошибка (расстояние между η_n и ζ_n) увеличивается с удалением от исходного интервала, на котором был построен аппроксимирующий полином.

4.3 Аппроксимация тригонометрическими полиномами

Для периодических функций использование степенных полиномов часто неудобно, поскольку их экстраполяционные свойства весьма слабы, так как сами они непериодичны. Поэтому прибегают к аппроксимации тригонометрическими полиномами вида (4.7).

$$W(x) = \frac{c_0}{2} + \sum_{i=1}^{P} \left(c_{2i-1} \sin(i\omega x) + c_{2i} \cos(i\omega x) \right), \tag{4.7}$$

где $\omega = \frac{2\pi}{T}$ — круговая частота, T — период, $P = \frac{N-1}{2}$, если N нечётно. Если N чётно, то следует либо отбросить одну точку, либо дополнить до нечётного, либо дополнительно взять ещё один член с синусом вида $c_{N-1} \sin(\frac{N}{2}\omega x)$.

Расчёт коэффициентов $\mathbf{c}$ можно произвести, сведя эту задачу к решению системы линейных уравнений любым известным способом. Матрицу $\hat{A}$ можно рассчитать, представив $a_{i,j}$ по формуле (4.8).

$$\begin{cases} a_{n,0} &= \frac{1}{2} \\ a_{n,2i-1} &= \sin(i\omega x_n) \\ a_{n,2i} &= \cos(i\omega x_n) \\ i &= 1, \ldots, P. \end{cases} \tag{4.8}$$

Вектор свободных членов совпадает с массивом $\{y_n\}_{n=0}^{N-1}$.

При аппроксимации тригонометрическим полиномом очень важно верно определить период T. Существуют несколько подходов к решению этой проблемы, хотя при работе с экспериментальными данными ни один из них не может гарантировать 100%-ный успех.

- Период T или частота ω известны из некоторых априорных соображений о физической природе наблюдаемого сигнала.

- В ряде отыскиваются все локальные минимумы или максимумы, пусть их K штук и им соответствуют $\{x_{n_k}\}_{k=0}^{K-1}$, где n_k — номер точки в исходном ряде $\{x_n\}_{n=0}^{N-1}$. Тогда период можно рассчитать как расстояние между крайними максимумами (или минимумами), делённое на их число минус один: $T = \frac{x_{n_{K-1}} - x_{n_0}}{K-1}$.

- Вся длина ряда принимается за период: $T = x_{N-1} - x_0$. Такой подход хорош для аппроксимации непериодических колебательных рядов. Иногда делают поправку: если все значения x_n эквидистантны, то есть $x_{n+1} - x_n = \Delta x \ \forall n$, то за период считают не расстояние между начальным и конечным x_n, равное $(N-1)\Delta x$, а расстояние, равное произведению Δx на число точек, т. е. $T = N\Delta x$.

От того, насколько точно выбрано значение T, будет зависеть качество как интерполяции, так и экстраполяции.

Глава 5

Аппроксимация кубическими сплайнами

5.1 Метод прогонки

5.1.1 Постановка задачи о решении системы уравнений методом прогонки

В ряде случаев решить систему из L линейных уравнений можно существенно проще, чем используя метод Гаусса или QR-разложение. Это относится в частности к системам с трёхдиагональными матрицами типа (5.1) (пример приведён для $L = 5$), где знаком x обозначены ненулевые элементы [2, 5].

$$\hat{A} = \begin{bmatrix} \mathrm{x} & \mathrm{x} & 0 & 0 & 0 \\ \mathrm{x} & \mathrm{x} & \mathrm{x} & 0 & 0 \\ 0 & \mathrm{x} & \mathrm{x} & \mathrm{x} & 0 \\ 0 & 0 & \mathrm{x} & \mathrm{x} & \mathrm{x} \\ 0 & 0 & 0 & \mathrm{x} & \mathrm{x} \end{bmatrix} \tag{5.1}$$

Условие трёхдиагональности можно сформулировать следующим образом:

$$a_{i,j} = 0 \quad \forall\ |i - j| > 1, \tag{5.2}$$

т. е., проще говоря, если номера строки i и столбца j отличаются более чем на 1, то соответствующий элемент $a_{i,j}$ матрицы $\hat{A}$ равен нулю.

При работе с трёхдиагональными матрицами, как правило, переходят от представления в общем виде (5.1) к представлению через диагонали $\boldsymbol{\gamma}_0$ (нижняя), $\boldsymbol{\gamma}_1$ (центральная) и $\boldsymbol{\gamma}_2$ (верхняя), коэффициенты

которых выражаются по формуле (5.3).

$$\begin{aligned} \boldsymbol{\gamma}_{0,i} &= a_{i+1,i}, \quad i = 0, \ldots, L-2, \\ \boldsymbol{\gamma}_{1,i} &= a_{i,i}, \qquad i = 0, \ldots, L-1, \\ \boldsymbol{\gamma}_{2,i} &= a_{i,i+1}, \quad i = 0, \ldots, L-2, \end{aligned} \tag{5.3}$$

Верхняя и нижняя диагонали короче центральной на 1 и имеют длину $(L-1)$. Представление через диагонали имеет существенное преимущество, если работают с матрицами большого размера, например, $L = 10000$ и более, поскольку оно требует хранить только $\sim 3L$ значений, в то время как при использовании матрицы общего вида приходится хранить $\sim L^2$ значений. Поиск нужного значения в такой огромной матрице требует больших вычислительных ресурсов, а при дальнейшем её росте она вскоре перестаёт умещаться в оперативную память даже современных компьютеров. Необходимость работы с матрицами такого размера возникает, в частности, в задачах интерполяции сплайнами, где метод прогонки используется для поиска коэффициентов сплайна.

5.1.2 Алгоритм метода прогонки

Пусть имеется задача решить систему вида (2.5), где матрица $\hat{A}$ имеет вид (5.1), а вектор свободных членов обозначим $\boldsymbol{\beta}$. Весь алгоритм разделяется на 3 этапа.

1. Прямая прогонка. Домножим нулевую строку на $a_{1,0}$ и поделим на $a_{0,0}$ (по аналогии с методом Гаусса обозначим отношение $g = a_{1,0}/a_{0,0}$), а теперь вычтем из первой. Элементы первой строки с номерами $i > 1$ никак не изменятся, поскольку в нулевой строке соответствующие элементы равны 0. Элемент $a_{1,0}$ станет равен нулю, преобразуются только $a_{1,1} = a_{1,1} - g a_{0,1}$ и $\beta_1 = \beta_1 - g\beta_0$. После этой операции матрица $\hat{a}$ преобразуется к виду (5.4):

$$\hat{A} = \begin{bmatrix} \mathrm{x} & \mathrm{x} & 0 & 0 & 0 \\ 0 & \mathrm{x} & \mathrm{x} & 0 & 0 \\ 0 & \mathrm{x} & \mathrm{x} & \mathrm{x} & 0 \\ 0 & 0 & \mathrm{x} & \mathrm{x} & \mathrm{x} \\ 0 & 0 & 0 & \mathrm{x} & \mathrm{x} \end{bmatrix} \tag{5.4}$$

   Далее можно аналогично вычесть первую строку из второй,пользуясь тем, что теперь $a_{1,0} = 0$. И так далее. В результате

можно записать общую формулу для прямой прогонки:

$$\begin{aligned} g &= a_{i+1,i}/a_{i,i}, \\ a_{i+1,i+1} &= a_{i+1,i+1} - g a_{i,i+1}, \\ \beta_{i+1} &= \beta_{i+1} - g\beta_i, \\ i &= 0 \ldots L-2. \end{aligned} \tag{5.5}$$

Эту же формулу можно переписать через диагонали $\boldsymbol{\gamma_0}$, $\boldsymbol{\gamma_1}$ и $\boldsymbol{\gamma_2}$ следующим образом:

$$\begin{aligned} g &= \gamma_{0,i}/\gamma_{1,i}, \\ \gamma_{1,i+1} &= \gamma_{1,i+1} - g\gamma_{2,i}, \\ \beta_{i+1} &= \beta_{i+1} - g\beta_i, \\ i &= 0 \ldots L-2. \end{aligned} \tag{5.6}$$

по окончании прямой прогонки матрица $\hat{A}$ примет вид:

$$\hat{A} = \begin{bmatrix} \mathrm{x} & \mathrm{x} & 0 & 0 & 0 \\ 0 & \mathrm{x} & \mathrm{x} & 0 & 0 \\ 0 & 0 & \mathrm{x} & \mathrm{x} & 0 \\ 0 & 0 & 0 & \mathrm{x} & \mathrm{x} \\ 0 & 0 & 0 & 0 & \mathrm{x} \end{bmatrix} \tag{5.7}$$

2. Обратная прогонка. Целью обратной прогонки является преобразование матрицы $\hat{A}$ к такому виду, где отсутствовала бы (все элементы были бы равны нулю) верхняя диагональ $\boldsymbol{\gamma}_2$. Для этого вычитают сначала последнюю строку из предпоследней, далее предпоследнюю из предпредпоследней и так далее. Поскольку все элементы вычитаемой i-той строки при этом равны нулю, кроме элемента с номером i, $(i-1)$-ая строка пир этом не преобразуется кроме её i-того элемента, который становится равен 0. Поэтому всё преобразование по сути сводится к изменению вектора $\boldsymbol{\beta}$ по следующему правилу:

$$\begin{aligned} \beta_{i-1} &= \beta_{i-1} - \beta_i \frac{\gamma_{2,i-1}}{\gamma_{1,i}}, \\ i &= L-1, \ldots, 1 \end{aligned} \tag{5.8}$$

В результате обратной прогонки матрица $\hat{A}$ сведётся к однодиагональному виду.

3. Вычисление неизвестных. По преобразованной матрице легко вычислить искомый вектор $\mathbf{c}$:

$$\begin{aligned} c_i &= \beta_i / \gamma_{1,i}, \\ i &= 0, \ldots, L-1. \end{aligned} \tag{5.9}$$

Метод прогонки имеет важное преимущество перед методом Гаусса и решением через QR-разложение с точки зрения производительности, особенно при большом L. Алгоритмическая сложность метода прогонки пропорциональна L, в то время как сложность метода Гаусса и QR-разложения пропорциональна L^3.

Задания на метод прогонки

Реализуйте метод прогонки в виде отдельной подпрограммы (процедуры или функции), получающей на вход четыре массива: три диагонали $\boldsymbol{\gamma}_0$, $\boldsymbol{\gamma}_1$ и $\boldsymbol{\gamma}_2$ и массив свободных членов $\boldsymbol{\beta}$, а на выход выдающей массив $\mathbf{c}$. Для тестирования правильности написанного алгоритма можно использовать ранее освоенные методы решения систем линейных уравнений.

Дополнительные задания:

1. При решении методом прогонки на диагонали может образоваться нулевой элемент. В таком случае решение нужно прекратить, поскольку пир делении на 0 возникнет ошибка (в таком случае решение проводят методом Гаусса или через QR-разложение). Реализуйте такую проверку, в случае обнаружения нуля на диагонали выдавайте предупреждение.

2. В методе прогонки вектор свободных членов $\boldsymbol{\beta}$ и векторы-диагонали $\boldsymbol{\gamma}_0$, $\boldsymbol{\gamma}_1$ и $\boldsymbol{\gamma}_2$ изменяются. Реализуйте метод таким образом, чтобы с помощью дополнительного формального параметра можно было контролировать, изменяются ли исходные массивы, либо для вычислений используются их копии.

5.2 Расчёт коэффициентов сплайна

5.2.1 Свойства кубических сплайнов

Пусть имеется набор измерений $\{x_n\}_{n=0}^{N-1}$ физической величины X, измеренных в моменты времени $\{t_n\}_{n=0}^{N-1}$, причём эти моменты времени отсортированы по возрастанию, т. е. выполняется условие $\cdots < t_{n-1} < t_n < t_{n+1} < \ldots$. Тогда кубическим сплайном, построенным в узлах $\{t_n, x_n\}_{n=0}^{N-1}$ называется функция $s(t)$, удовлетворяющая следующим условиям.

1. $s(t)$ проходит через все экспериментальные точки, то есть

$$s(t_n) = x_n \tag{5.10}$$

   Здесь $n = 0, \ldots, N-2$, поскольку промежутков на 1 меньше, чем точек (узлов).

2. На каждом промежутке между узлами с номером n и $(n+1)$ $s(t)$ представляет собою кубический полином $s_n(t)$ вида

$$s_n(t) = a_n + b_n(t - t_n) + \frac{c_n}{2}(t - t_n)^2 + \frac{d_n}{6}(t - t_n)^3, \tag{5.11}$$

   Аналогично предыдущему пункту здесь $n = 0, \ldots, N-2$.

3. $s(t)$ непрерывна на всём отрезке определения от t_0 до t_{N-1}, что приводит к соотношению:

$$s_n(t_{n+1}) = s_{n+1}(t_{n+1}) \tag{5.12}$$

   Непрерывности можно требовать только в средних узлах, для которых сплайн определён как слева, так и справа, поэтому $n = 1, \ldots, N-2$.

4. Первая производная функции $s(t)$ существует на всей области определения. Поскольку кубический полином по определению есть функция всюду непрерывно дифференцируемая, достаточно потребовать существования производной в узлах сплайна. Известно, что производная существует тогда и только тогда, когда она существует слева, справа и они равны:

$$\left.\frac{ds_n}{dt}\right|_{t_{n+1}} = \left.\frac{ds_{n+1}}{dt}\right|_{t_{n+1}} \tag{5.13}$$

Равенства производных можно требовать только в средних узлах, для которых сплайн определён как слева, так и справа, поэтому $n = 1, \ldots, N-2$.

5. Вторая производная функции $s(t)$ существует на всей области определения. Аналогично, поскольку кубический полином по определению есть функция всюду непрерывно дифференцируемая, достаточно потребовать существования производной в узлах сплайна. Известно, что производная существует тогда и только тогда, когда она существует слева, справа и они равны:

$$\left.\frac{d^2 s_n}{dt^2}\right|_{t_{n+1}} = \left.\frac{d^2 s_{n+1}}{dt^2}\right|_{t_{n+1}} \tag{5.14}$$

Аналогично первой производной $n = 1, \ldots, N-2$.

Для простоты интерпретации соотношений (5.13) и (5.14) выпишем формулы для первой и второй производных в явном виде:

$$\frac{ds_n}{dt} = b_n + c_n(t - t_n) + \frac{d_n}{2}(t - t_n)^2, \tag{5.15}$$

$$\frac{d^2 s_n}{dt^2} = c_n + d_n(t - t_n). \tag{5.16}$$

5.2.2 Вывод уравнений для расчёта коэффициентов

На основании свойств (5.10, 5.11, 5.12, 5.13, 5.14) можно построить уравнения для расчёта коэффициентов сплайна. Для этого нужно подставить определение кубического полинома (5.11) в соотношения (5.10, 5.12, 5.13, 5.14), используя формулы для производных (5.15) и (5.16). Чтобы итоговая запись была короче, введём дополнительное обозначение длины интервала по времени между узлами:

$$h_n = t_{n+1} - t_n, \tag{5.17}$$

где $n = 0, \ldots, N-2$, поскольку интервалов на 1 меньше, чем узлов.

В результате подстановки и замены получится система уравнений

(5.18).

$$\begin{aligned} a_n &= x_n, \\ a_n + b_n h_n + \frac{c_n}{2} h_n^2 + \frac{d_n}{6} h_n^3 &= a_{n+1}, \\ b_n + c_n h_n + \frac{d_n}{2} h_n^2 &= b_{n+1}, \\ c_n + d_n h_n &= c_{n+1}. \end{aligned} \tag{5.18}$$

Первое уравнение системы (5.18) имеет совершенно очевидное решение и даёт нам значения для коэффициентов a_n. Остальные уравнения, однако, являются связанными и просто не решаются. Чтобы решить их численно, выразим b_n и d_n через c_n. Для этого сначала из последнего уравнения системы 5.18 получим:

$$d_n = \frac{c_{n+1} - c_n}{h_n} \tag{5.19}$$

А затем, подставив (5.19) во второе уравнение системы (5.18) и используя знание о том, что $a_n = x_n$, получим:

$$b_n h_n + \frac{c_n}{2} h_n^2 + \frac{c_{n+1} - c_n}{6} h_n^2 = x_{n+1} - x_n,$$

откуда, приведя подобные, получим:

$$b_n = \frac{x_{n+1} - x_n}{h_n} - \frac{c_n}{3} h_n - \frac{c_{n+1}}{6} h_n. \tag{5.20}$$

Теперь, наконец, можно подставить соотношения (5.20) и (5.19) в третье уравнение системы (5.18) и получить систему линейных уравнений для коэффициентов $\mathbf{c}$:

$$\begin{aligned} &\frac{x_{n+1} - x_n}{h_n} - \frac{c_n}{3} h_n - \frac{c_{n+1}}{6} h_n + c_n h_n + \frac{c_{n+1} - c_n}{2} h_n = \\ &\frac{x_{n+2} - x_{n+1}}{h_{n+1}} - \frac{c_{n+1}}{3} h_{n+1} - \frac{c_{n+2}}{6} h_{n+1} \end{aligned}$$

которая сводится к виду (5.21) после приведения подобных.

$$c_n h_n + 2c_{n+1}(h_n + h_{n+1}) + c_{n+2} h_{n+1} = 6 \left(\frac{x_{n+2} - x_{n+1}}{h_{n+1}} - \frac{x_{n+1} - x_n}{h_n} \right) \tag{5.21}$$

Можно заметить, что система (5.21) имеет трёхдиагональную матрицу, поэтому её можно решать методом прогонки. Для этого нужно

доопределить некоторые коэффициенты. Метод прогонки будет работать, если в первой и последней строках матрицы вида (5.1) будет по 2 ненулевых коэффициента, но в системе (5.21) во всех уравнениях по 3 неизвестных: c_n, c_{n+1} и c_{n+2}. Отметим, что число условий на коэффициенты в системе (5.18) на самом деле не достаточно, поскольку уравнения для непрерывности первой и второй производных: (5.13) и (5.14) не могут быть только для $(N-2)$ узлов (только для тех, для которых есть отрезок и справа, и слева), так что всего уравнений — $(4N-6)$, в то время как коэффициентов по числу отрезков — $(4N-4)$. Чтобы задача имела решение, можно произвольно определить любую пару коэффициентов. Удобнее всего положить $c_0 = 0$ и наложить условие на $d_{N-2} = -c_{N-2}/h_{N-2}$, то есть положить несуществующий в действительности коэффициент $c_{N-1} = 0$. После этого можно выразить значения диагоналей $\boldsymbol{\gamma}_0$, $\boldsymbol{\gamma}_1$ и $\boldsymbol{\gamma}_2$ и вектора свободных членов $\boldsymbol{\beta}$ следующим образом:

$$\begin{aligned} \gamma_{0,n} = \gamma_{2,n} &= h_{n+1} \qquad n = 0, \ldots, N-3 \\ \gamma_{1,n} &= 2\left(h_n + h_{n+1}\right) \qquad n = 0, \ldots, N-2 \\ \beta_n &= 6\left(\frac{x_{n+2} - x_{n+1}}{h_{n+1}} - \frac{x_{n+1} - x_n}{h_n}\right) \qquad n = 0, \ldots, N-2 \end{aligned} \tag{5.22}$$

Из системы (5.21, 5.22) можно получить $(N-2)$ значения коэффициентов c_n: с номерами от 1 до $(N-1)$, в то время как $c_0 = 0$ по построению. Решать систему (5.21, 5.22) предпочтительно методом прогонки. Получив из неё значения коэффициентов c_n можно выразить значения d_n по формуле (5.19) и b_n по формуле (5.20), не забывая, что для вычисления b_{N-2} и d_{N-2} следует принять $c_{N-1} = 0$.

5.3 Расчёт значений сплайна по новой сетке

Конечная цель интерполяции сплайнами состоит, как правило, не в определении коэффициентов, а в расчёте значений x при других значениях t, отличных от значений в узлах $\{t_n\}_{n=0}^{N-1}$. Эти новые значения t обозначим $\{\tau_k\}_{k=0}^{K-1}$, а значения x, вычисленные в этих точках, — $\{\xi_k\}_{k=0}^{K-1}$. Важно, что все τ_k принадлежат исходному отрезку $[t_0;\ t_{N-1}]$

Чтобы вычислить $\{\xi_k\}_{k=0}^{K-1}$, необходимо для каждого τ_k выполнить два действия:

1. определить номер n отрезка $[t_n;\ t_{n+1}]$ в сетке исходных узлов, к которому принадлежит τ_k,

2. рассчитать значение ξ_k по формуле (5.11).

Чтобы определить n, можно организовать цикл с условием с перебором n, начиная с n_0 пока $\tau_k > t_n$. Из получившегося значения n нужно вычесть 1, поскольку мы перескочили на 1 отрезок вперёд. Для расчёта функции s_n лучше всего написать отдельную функцию, принимающую скалярное значение t и либо 4 параметра: a_n, b_n, c_n, d_n либо массив или список из 4-х значений этих параметров.

Глава 6

Численное дифференцирование

6.1 Постановка задачи

При разного рода исследованиях и измерениях далеко не всегда возможно померить ту самую величину, которую необходимо. Например, вам необходимо знать скорость спортсмена на каждой контрольной отметке (предположим, что отметки стоят достаточно часто), но измерить вы можете только время и расстояние между отметками. Другой пример: учёные пытаются вычислить ускорение некоторой звезды или галактики, но метод, основанный на допплеровском смещении, может дать нам информацию только об относительной скорости. В таких случаях прибегают к такой процедуре, как численное дифференцирование. Сразу оговоримся, что численное дифференцирование, как и любая другая численная процедура, не позволяет определить точные значения искомой величины, а только рассчитать некоторые приближения для них. Точность этих приближений может зависеть как от точности измерения самих данных, от интервала между измеренными значениями, от используемого метода.

Итак, пусть нам даны два ряда: набор измеренных значений $\{x_n\}_{n=0}^{N-1}$ и набор моментов времени, в которые они измерены, — $\{t_n\}_{n=0}^{N-1}$. Требуется для всех значений t_n оценить значения производной $z_n = dx/dt|_{t=t_n}$.

6.2 Дифференцирование без сглаживания

Простейший способ оценить производную — вспомнить её определение (6.1). Заменив бесконечно малые разности на конечные, можно получить некоторое приближение к искомой величине.

$$\frac{dx}{dt} = \lim_{\Delta t \to 0} \frac{\Delta x}{\Delta t} \tag{6.1}$$

В нашем случае логично использовать в качестве интервала Δt величину $t_n - t_{n-1}$, поскольку это минимальный шаг по времени, который доступен из наблюдаемых данных. Тогда формула (6.1) преобразуется к виду:

$$z_n \approx \frac{x_n - x_{n-1}}{t_n - t_{n-1}} \tag{6.2}$$

Заметим, что полученная оценка не даёт ответа на вопрос, каково значение производной в точке t_0, поскольку t_{-1} отсутствует среди измеренных значений, т. е. временной ряд производной будет на 1 значение короче, чем два исходных.

Оценка (6.2) имеет множество недостатков. Первый из них состоит в том, что оценка несимметрична: в самом деле, для оценки z_n используются x_n и x_{n-1}, но не x_{n+1}. То есть при оценке производных последующие значения имеют меньший вес, чем предыдущие, что вводит систематическое смещение по времени примерно на половину характерного шага по времени $\Delta t = \langle t_n - t_{n-1} \rangle$. Эту неточность можно попробовать устранить следующим образом:

$$z_n \approx \frac{x_{n+1} - x_{n-1}}{t_{n+1} - t_{n-1}} \tag{6.3}$$

Такой переход часто называют переходом от разницы с лева к симметричной, при этом из-за отсутствия необходимого количества данных производные z_0 и z_{N-1} в начальной и конечной точках оказываются не определены. Однако подобная правка имеет свои недостатки, в частности полностью не учитываются значения x_n и t_n, т. е. оказывается, что значение производной не зависит от значения координаты в тот же момент времени. Кроме того, в такой постановке формула (6.3) не учитывает возможность того, что интервалы $(t_{n+1} - t_n)$ и $(t_n - t_{n-1})$ не равны между собою, поскольку реально оценивается значение производной в середине промежутка $(t_{n+1} - t_{n-1})$, а середина промежутка при неравномерном шаге по времени вовсе не обязана совпасть с положением точки t_n.

Помимо перечисленных оба подхода: и оценка слева (6.2), и симметричная оценка (6.3) имеют один общий базовый недостаток: если исходные данные зашумлены, в оценку производных будет внесена огромная погрешность [7, 8]. В самом деле, если характерная величина шумовой добавки σ_{rum} окажется сравнима или больше среднего изменения сигнала $\{x_n\}_{n=0}^{N-1}$ на промежутке Δt, значения производной будут более зависеть от конкретных значений измерительного шума, чем от реального сигнала. Причина такой неэффективности в том, что при построении производной в каждый момент времени учитываются только 2 значения, а не сигнал в целом. Чтобы ослабить влияние шума на результат численного дифференцирования, вводят специализированную процедуру усреднения.

6.3 Дифференцирование со сглаживанием по формуле

Идея дифференцирования со сглаживанием состоит в следующем: для каждого момента времени t_n, в который необходимо найти производную, строится аппроксимация исходных данных с помощью некоторой функции $\varphi(t)$, после чего оценка производной рассчитывается аналитически как $z_n = d\varphi/dt|_{t=t_n}$. При этом функция $\varphi(t)$ подгоняется по m точкам, включая саму точку с номером n.

6.3.1 Сглаживание прямою

Простейший пример функции — линейная зависимость:

$$\varphi_n(t) = k_n t + b_n \tag{6.4}$$

Для неё значение производной в точке t_n вычисляется как $d\varphi_n/dt|_{t=t_n} = k_n$. Заметим, что для каждой точки n значения коэффициентов k_n и b_n свои, поскольку набор значений t_i и x_i, по которым подгонялась функция, каждый раз отличается.

Для удобства далее будем считать, что m всегда нечётно, при этом и слева, и справа от n-ой учитывается одинаковое число точек — $m_2 = (m-1)/2$. Следует отметить, что при таком выборе найти производную

в первых m_2 и последних m_2 точках не получится, поскольку не хватит данных для сглаживания слева или справа соответственно. Функцию (6.4) будем подгонять методом наименьших квадратов, выражение для минимизируемого отклонения выглядит следующим образом:

$$S_n(k_n, b_n) = \sum_{j=n-m_2}^{n+m_2} (x_j - (k_n t_j + b_n))^2 = \min \tag{6.5}$$

Следует рассматривать S_n именно как функцию двух переменных: k_n и b_n. Чтобы найти k_n и b_n, решим задачу оптимизации (нахождения минимума), для чего выпишем производные S_n по k_n и b_n и приравняем их нулю:

$$\begin{aligned} \frac{\partial S_n}{\partial k_n} &= \sum_{j=n-m_2}^{n+m_2} \big(x_j - (k_n t_j + b_n)\big) t_j = 0, \\ \frac{\partial S_n}{\partial b_n} &= \sum_{j=n-m_2}^{n+m_2} \big(x_j - (k_n t_j + b_n)\big) = 0. \end{aligned} \tag{6.6}$$

Раскроем суммы в системе (6.6) и используем то, что ни k_n, ни b_n не зависят от j, тогда (6.6) перепишется к виду:

$$\begin{aligned} \sum_{j=n-m_2}^{n+m_2} x_j t_j &= k_n \sum_{j=n-m_2}^{n+m_2} t_j^2 + b_n \sum_{j=n-m_2}^{n+m_2} t_j, \\ \sum_{j=n-m_2}^{n+m_2} x_j &= k_n \sum_{j=n-m_2}^{n+m_2} t_j + b_n m. \end{aligned} \tag{6.7}$$

Последнее слагаемое во втором уравнении системы (6.7) появилось вследствие суммирования m единиц. Из второго уравнения выражаем b_n:

$$b_n = \frac{1}{m} \left(\sum_{j=n-m_2}^{n+m_2} x_j - k_n \sum_{j=n-m_2}^{n+m_2} t_j \right) \tag{6.8}$$

и подставляем в первое, разделяем слагаемые с k_n и без него, в результате получаем:

$$\sum_{j=n-m_2}^{n+m_2} x_j t_j - \frac{1}{m} \sum_{j=n-m_2}^{n+m_2} x_j \sum_{j=n-m_2}^{n+m_2} t_j = k_n \cdot \left(\sum_{j=n-m_2}^{n+m_2} t_j^2 - \frac{1}{m} \left(\sum_{j=n-m_2}^{n+m_2} t_j \right)^2 \right) \tag{6.9}$$

Отсюда получаем формулу для нахождения k_n (а, значит, и оценки производной в точке n) в общем случае:

$$k_n = \frac{\sum\limits_{j=n-m_2}^{n+m_2} x_j t_j - \frac{1}{m} \sum\limits_{j=n-m_2}^{n+m_2} x_j \sum\limits_{j=n-m_2}^{n+m_2} t_j}{\sum\limits_{j=n-m_2}^{n+m_2} t_j^2 - \frac{1}{m} \left(\sum\limits_{j=n-m_2}^{n+m_2} t_j \right)^2} \tag{6.10}$$

Формула (6.10) универсальна, но имеет 2 ключевых недостатка: во-первых, при больших значениях времени величины типа t_j^2 приведут к существенному снижению точности, во-вторых, очень велики вычислительные затраты. В самом деле, для каждой из N точек нужно посчитать 4 суммы по m элементов, две из которых имеют ещё m операций умножения.

Обе эти проблемы можно решить хотя бы частично. Чтобы минимизировать потери точности от возведения в квадрат больших значений t_j, вспомним, что для нас важен только наклон полученной прямой — коэффициент k_n, коэффициент b_n нас не волнует, наклон же не зависит от выбора начала отсчёта времени t, поэтому мы будем всегда помещать начало отсчёта в точку n, т. е. считать $t_n = 0$. Тогда формула (6.10) преобразуется к виду:

$$k_n = \frac{\sum\limits_{j=n-m_2}^{n+m_2} x_j (t_j - t_n) - \frac{1}{m} \sum\limits_{j=n-m_2}^{n+m_2} x_j \sum\limits_{j=n-m_2}^{n+m_2} (t_j - t_n)}{\sum\limits_{j=n-m_2}^{n+m_2} (t_j - t_n)^2 - \frac{1}{m} \left(\sum\limits_{j=n-m_2}^{n+m_2} (t_j - t_n) \right)^2} \tag{6.11}$$

При этом, правда, вычислительная сложность ещё возросла.

Проблема вычислительной сложности решается для достаточно типичной ситуации, когда все интервалы выборки $\Delta t_n = t_{n+1} - t_n$ одинаковы и равны Δt. Временной ряд, для которого $\Delta t_n = \Delta t \ \forall n$ называется *эквидистантным*. Для такого ряда суммы в формуле (6.11) существенно упростятся. Рассмотрим суммы по времени:

$$\sum_{j=n-m_2}^{n+m_2} (t_j - t_n) = \sum_{j=n-m_2}^{n+m_2} (j\Delta t - n\Delta t) = \Delta t \sum_{j=n-m_2}^{n+m_2} (j - n) \tag{6.12}$$

Заменим в последней формуле индекс суммирования с j на $i = j - n$, получим:

$$\Delta t \sum_{j=n-m_2}^{n+m_2} (j-n) = \sum_{i=-m_2}^{m_2} i = 0 \tag{6.13}$$

поскольку очевидно, что сумма последовательных целых чисел от $-m_2$ до m_2 включительно равна нулю $\forall m_2$, так как для любого положительного числа в такой последовательности всегда присутствует равное ему по модулю отрицательное число. Таким образом, в формуле (6.11) оба вторых слагаемых (и в числителе, и в знаменателе) обращаются в 0, а оставшиеся члены с учётом замены $i = j - n$ приобретают вид:

$$k_n = \frac{\sum_{i=-m_2}^{m_2} x_{i+n} i \Delta t}{\sum_{i=-m_2}^{m_2} i^2 (\Delta t)^2} \tag{6.14}$$

В формуле (6.14) можно сделать ещё несколько упрощений. Очевидно, что следует вынести из под знака сумм Δt и сократить на него 1 раз. Кроме того, можно заметить, что сумма в знаменателе может быть ещё упрощена:

$$\sum_{i=-m_2}^{m_2} i^2 = \sum_{i=1}^{m_2} i^2 + 0 + \sum_{i=-m_2}^{-1} i^2 = 2 \sum_{i=1}^{m_2} i^2 \tag{6.15}$$

Более того, полученная сумма общая для всех n, поэтому её можно посчитать единожды. С учётом этого получаем окончательную формулу для эквидистантного ряда:

$$k_n = \frac{1}{z} \sum_{i=-m_2}^{m_2} x_{i+n} \cdot i, \quad z = 2\Delta t \cdot \sum_{i=1}^{m_2} i^2 \tag{6.16}$$

6.3.2 Сглаживание параболою

Во многих случаях использование прямой может быть недостаточно. Основная проблема заключается в том, что при достаточно большом m или Δt зависимость $x(t)$ в окрестности t_n будет явно не похожа на прямую. Как правило, это приводит к занижению значений производной. Чтобы улучшить оценку, следует использовать для сглаживания кривую

более высокого порядка, чем линейная функция, например, квадратичную параболу (6.17).

$$\varphi_n(t) = a_n t^2 + b_n t + c_n \tag{6.17}$$

Подгонку коэффициентов параболы будем осуществлять, как и раньше, методом наименьших квадратов. Формула для минимума целевой функции выглядит следующим образом:

$$S_n = \sum_{j=n-m_2}^{n+m_2} \left(x_j - (a_n t_j^2 + b_n t_j + c_n)\right)^2 = \min \tag{6.18}$$

Для нахождения минимума следует продифференцировать (6.18) по всем трём коэффициентам: a_n, b_n и c_n, чтобы получить систему уравнений (6.19):

$$\begin{aligned}
\frac{dS_n}{da_n} &= \sum_{j=n-m_2}^{n+m_2} \left(a_n t_j^2 + b_n t_j + c_n\right) t_j^2 = 0, \\
\frac{dS_n}{db_n} &= \sum_{j=n-m_2}^{n+m_2} \left(a_n t_j^2 + b_n t_j + c_n\right) t_j = 0, \\
\frac{dS_n}{dc_n} &= \sum_{j=n-m_2}^{n+m_2} \left(a_n t_j^2 + b_n t_j + c_n\right) = 0
\end{aligned} \tag{6.19}$$

Искомая производная будет задаваться в виде:

$$\frac{d\varphi_n}{dt} = \frac{d(a_n t^2 + b_n t + c_n)}{dt} = 2a_n t + b_n \tag{6.20}$$

то есть коэффициент c_n нам не нужен; однако решить систему (6.19) без выражения c_n в общем виде (для произвольных t_j) не получится, поэтому преобразуем её к более удобному виду:

$$\begin{aligned}
a_n \sum_{j=n-m_2}^{n+m_2} t_j^4 + b_n \sum_{j=n-m_2}^{n+m_2} t_j^3 + c_n \sum_{j=n-m_2}^{n+m_2} t_j^2 &= \sum_{j=n-m_2}^{n+m_2} x_j t_j^2, \\
a_n \sum_{j=n-m_2}^{n+m_2} t_j^3 + b_n \sum_{j=n-m_2}^{n+m_2} t_j^2 + c_n \sum_{j=n-m_2}^{n+m_2} t_j &= \sum_{j=n-m_2}^{n+m_2} x_j t_j, \\
a_n \sum_{j=n-m_2}^{n+m_2} t_j^2 + b_n \sum_{j=n-m_2}^{n+m_2} t_j + c_n \sum_{j=n-m_2}^{n+m_2} 1 &= \sum_{j=n-m_2}^{n+m_2} x_j
\end{aligned} \tag{6.21}$$

В системе (6.21) множество однотипных сумм, очень громоздких и затрудняющих восприятие, для простоты введём обозначения:

$$T_i = \sum_{j=n-m_2}^{n+m_2} t_j^i, \quad X_i = \sum_{j=n-m_2}^{n+m_2} x_j t_j^i \tag{6.22}$$

Учитывая, что $\sum_{j=n-m_2}^{n+m_2} 1 = m$, выражаем из последнего уравнения системы (6.21) параметр c_n:

$$c_n = \frac{X_0 - a_n T_2 - b_n T_1}{m} \tag{6.23}$$

Теперь из второго уравнения выражаем b_n:

$$\begin{gathered} b_n T_2 = X_1 - a_n T_3 - c_n T_1 = X_1 - a_n T_3 - \tfrac{1}{m} X_0 T_1 + a_n \cdot \tfrac{1}{m} T_2 T_1 + b_n \cdot \tfrac{1}{m} T_1^2 \\ b_n = \frac{X_1 - \frac{1}{m} X_0 T_1}{T_2 - \frac{1}{m} T_1^2} + a_n \frac{\frac{1}{m} T_2 T_1 - T_3}{T_2 - \frac{1}{m} T_1^2} \end{gathered} \tag{6.24}$$

Теперь осталось подставить результат наших вычислений — выражения (6.23, 6.24) в первое уравнение системы (6.21):

$$\begin{aligned} X_2 &= a_n T_4 + \left(T_3 - \tfrac{1}{m} T_2 T_1\right) \frac{X_1 - \frac{1}{m} X_0 T_1}{T_2 - \frac{1}{m} T_1^2} + a_n \left(T_3 - \tfrac{1}{m} T_2 T_1\right) \frac{\frac{1}{m} T_2 T_1 - T_3}{T_2 - \frac{1}{m} T_1^2} + \\ &+ \tfrac{1}{m} T_2 X_0 - a_n \cdot \tfrac{1}{m} T_2^2 \end{aligned} \tag{6.25}$$

Теперь можно получить выражение для коэффициента a_n:

$$a_n = \frac{X_2 - \frac{1}{m} T_2 X_0 - \left(T_3 - \frac{1}{m} T_2 T_1\right) \frac{X_1 - \frac{1}{m} X_0 T_1}{T_2 - \frac{1}{m} T_1^2}}{T_4 - \frac{1}{m} T_2^2 + \left(T_3 - \frac{1}{m} T_2 T_1\right) \frac{\frac{1}{m} T_2 T_1 - T_3}{T_2 - \frac{1}{m} T_1^2}} \tag{6.26}$$

Вследствие сложности формул рассчитывать производную лучше в три этапа: сначала по формуле (6.26) подсчитать a_n, затем по формуле (6.24) — b_n, наконец, по формуле (6.20) значение оценки производной. Однако если интервалы $\Delta t_i = (t_{i+1} - t_i)$ эквидистантны, вычисления значительно упрощаются, поскольку из-за симметричности все T_i с нечётными индексами обращаются в ноль, поэтому из уравнения 2 системы (6.21) сразу же можно получить выражение для b_n:

$$b_n = \frac{X_1}{T_2} \tag{6.27}$$

Из уравнений 1 и 3 получается система для a_n и b_n, поскольку b_n выпадает из-за равенства нулю коэффициентов при нём — T_3 и T_1.

$$\begin{aligned} a_n T_4 + c_n T_2 &= X_2, \\ a_n T_2 + c_n m &= X_0 \end{aligned} \tag{6.28}$$

Выражаем c_n из первого уравнения системы (6.28) и подставляем во второе:

$$\begin{aligned} c_n &= \tfrac{1}{m} X_0 - a_n \cdot \tfrac{1}{m} T_2, \\ a_n T_4 + \tfrac{1}{m} X_0 T_2 - a_n \cdot \tfrac{1}{m} T_2^2 &= X_2 \end{aligned} \tag{6.29}$$

В результате имеем:

$$a_n = \frac{X_2 - \frac{1}{m} X_0 T_2}{T_4 - \frac{1}{m} T_2^2} \tag{6.30}$$

Итак, оценка производной может быть получена по формуле:

$$\left.\frac{d\varphi}{dt}\right|_{t=t_n} = 2a_n t_n + b_n = 2\frac{X_2 - \frac{1}{m} X_0 T_2}{T_4 - \frac{1}{m} T_2^2} t_n + \frac{X_1}{T_2} \tag{6.31}$$

Если теперь подобно изложенному выше для случая аппроксимации прямою перенести момент начала отсчёта времени в n-ную точку, то формула (6.31) упрощается и мы получаем

$$\left.\frac{d\varphi}{dt}\right|_{t=t_n} = \frac{X_1}{T_2} \tag{6.32}$$

то есть то же, что и для случая сглаживания с помощью прямой.

Таким образом, использование параболы не даёт преимущества с точки зрения точности оценки производной по сравнению с использованием прямой, если ряд эквидистантный. Тем не менее, проделанная работа была не напрасна, поскольку попутно мы получили формулу для оценки второй производной. В самом деле,

$$\frac{d^2\varphi}{dt^2} = \frac{d}{dt}\left(\frac{d\varphi}{dt}\right) = 2a_n \tag{6.33}$$

Полученные в данном разделе результаты вполне достаточны для большинства прикладных задач. Например, теперь мы можем оценить скорость и ускорение, имея только временной ряд координаты. В то же время, если получение более высоких производных или использования более сложного сглаживания всё же необходимо, сложность полученных

для приведённого в данном разделе способа сглаживания формул однозначно показывает, что аналитический вывод для случая полиномов более высокого порядка крайне затруднён, хотя и принципиально возможен. В таком случае следует прибегнуть к численному решению задачи на наименьшие квадраты для подгонки функции.

6.4 Дифференцирование со сглаживанием полиномом произвольной степени

6.4.1 Случай произвольного ряда

Задача о сглаживании полиномом произвольной степени P решается в целом так же, как и задача о сглаживании прямою или квадратичною параболою. Однако её бóльшая сложность и общность заставляют сделать несколько изменений в алгоритме. Во-первых, решить задачу наименьших квадратов аналитически для полинома произвольного порядка невозможно, а для известного, но большого порядка — очень сложно, поэтому используют численное решение. Во-вторых, для удобства и сокращения вычислительных затрат перенормируют время, введя замену $\tau_i = t_{n+i} - t_n$, $i = -m_2, \ldots, m_2$. При таком подходе сглаживающий полином имеет вид:

$$S_n(\mathbf{c}_n) = \sum_{j=0}^{P} c_{n,j}\tau^j, \tag{6.34}$$

где вектор (т. е. массив) коэффициентов $\mathbf{c}_n = \left(c_{n,0}, c_{n,1}, \ldots, c_{n,P}\right)$ имеет длину $P+1$. Одночлены τ^j представляют собою базисные функции. Число базисных функций также равно $P+1$. Матрица их значений $\hat{\Phi}$, записанная в экспериментальных точках τ_i, $i = -m_2, \ldots, m_2$, имеет вид:

$$\hat{\Phi}_n = \begin{bmatrix} 1 & \tau_{n-m_2} & \tau_{n-m_2}^2 & \cdots & \tau_{n-m_2}^P \\ 1 & \tau_{n-m_2+1} & \tau_{n-m_2+1}^2 & \cdots & \tau_{n-m_2+1}^P \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \tau_{n+m_2} & \tau_{n+m_2}^2 & \cdots & \tau_{n+m_2}^P \end{bmatrix} \tag{6.35}$$

Целевой вектор $\boldsymbol{\beta}_n$ представляется как:

$$\boldsymbol{\beta}_n = [x_{n-m_2}, x_{n-m_2+1}, \ldots, x_{n+m_2}] \tag{6.36}$$

Как всегда при решении задачи на наименьшие квадраты необходимо, чтобы число базисных функций было меньше или равно числу точек, что задаётся условием $m \leqslant P+1$ (высота $\hat{\Phi}_n$ должна быть не меньше её ширины).

В результате решения задачи на наименьшие квадраты (6.37) можно получается вектор коэффициентов $\mathbf{c}_n$ полинома (6.34).

$$\hat{\Phi}_n \mathbf{c}_n - \boldsymbol{\beta}_n \to \min \tag{6.37}$$

Заметим, что k-тая производная полинома (6.34) в точке $\tau_0 = 0$, соответствующей исходному моменту времени t_n, выражается как:

$$\frac{d^k S_n}{d\tau^k} = \sum_{j=k}^{P} c_{n,j} \frac{j!}{(j-k)!} \tau^{j-k} \tag{6.38}$$

Однако нас интересует в первую очередь производная в точке $\tau_n = 0$. Очевидно, что для неё все $\tau^{j-k} = 0$ кроме случая, когда $j = k$, поскольку $\tau^0 = 1$ и подстановка $\tau = \tau_n$ не играет здесь роли. Таким образом, при подстановке $\tau = \tau_n$ из всей суммы в формуле (6.38) остаётся только одно единственное слагаемое, соответствующее $j = 0$. В результате, получаем итоговую формулу для определения любой производной:

$$z_n \approx \left. \frac{d^k S_n}{d\tau^k} \right|_{\tau=\tau_n} = c_{n,k} k! \tag{6.39}$$

Интересно, что формула (6.39) позволяет найти в том числе так называемую «нулевую» производную. Фактически, «нулевая» производная — это сглаженный временной ряд исходной переменной x_n. Его часто используют вместо оригинального ряда, поскольку такое сглаживание позволяет частично избавиться от высокочастотных шумов измерений.

6.4.2 Случай эквидистантного ряда

С вычислительной точки зрения алгоритм сглаживания полиномом произвольного вида — самый тяжёлый из перечисленных, поскольку требует численного решения задачи на наименьшие квадраты для каждой из $N - m + 1$ точек, где будут рассчитаны значения производных. Кроме того, получить только часть коэффициентов $c_{n,j}$ невозможно — только все разом. В случае эквидистантного ряда можно существенно уменьшить число расчётов, использовав одинаковую для всех

$n = m_2, m_2 + 1, \ldots, N - m_2$ матрицу базисных функций. Для этого заметим, что в таком случае $\tau_i = i\Delta t$, т. е. τ_i более не зависит от n. Тогда можно ещё раз перенормировать время, поделив τ на Δt и далее используя i вместо времени. В итоге, матрица базисных функций (6.35) преобразуется к виду (6.40).

$$\hat{\Phi}'_n = \begin{bmatrix} 1 & n - m_2 & (n - m_2)^2 & \cdots & (n - m_2)^P \\ 1 & n - m_2 + 1 & (n - m_2 + 1)^2 & \cdots & (n - m_2 + 1)^P \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & (n + m_2) & (n + m_2)^2 & \cdots & (n + m_2)^P \end{bmatrix} \tag{6.40}$$

Эту матрицу можно заполнить однажды и использовать для всех n. Целевой вектор (6.36) не претерпевает изменений, но в итоговую формулу (6.39) необходимо внести поправку, поскольку перенормировав время, мы изменили и соответствующий дифференциал. Разница с предыдущим случаем состоит в том, что замена t на $\tau = t - t_n$ не приводит к изменению дифференциала, поскольку t_n есть константа и, следовательно, $d\tau = d(t - t_n) = dt - dt_n = dt$, так как дифференциал константы равен нулю. Но замена τ_i на $i = \tau_i / \Delta t$ приводит к масштабированию дифференциала: $di = d(\tau/\Delta t) = \frac{1}{\Delta t} d\tau$. Таким образом, чтобы вернуться к исходному времени, нужно провести масштабирование полученной производной на Δt столько раз, каков её порядок:

$$z_n \approx \left. \frac{d^k S_n}{dt^k} \right|_{t=t_n} = c_{n,k} k! (\Delta t)^k \tag{6.41}$$

С вычислительной точки зрения это несложная процедура по сравнению с необходимостью заполнять матрицу $\hat{\Phi}_n$ для каждого n.

Литература

[1] Гайдышев И. Анализ и обработка данных: специальный справочник — СПб: Питер, 2001. — 752 с.: ил.

[2] Деммель Дж. Вычислительная линейная алгебра. Теория и приложения. Пер. с англ. — М.: Мир, 2001. — 430 с., ил.

[3] Бахвалов Н.С., Жидков Н.П., Кобельков Г.М. Численные методы. —6-ое изд. — М.: БИНОМ. Лаборатория знаний, 2008. — 636 с., ил.

[4] Гилл. Ф., Мюррей У. Райт М. Практическая оптимизация: Пер. с англ. — М.: Мир, 1985. — 509 с., ил.

[5] Калиткин Н.Н. Численные методы — М.: Наука, 1978. — 512 с.

[6] Белов С.А., Золотых Н.Ю. Численные методы линейное алгебры. Лабораторный практикум. — Нижний Новгород: Изд-во Нижегородского госуниверситета им. Н.И. Лобачевского, 2005. — 264 с.

[7] Павлов А.Н., Павлова О.Н. Методы анализа сложных сигналов: Учеб. пособие для студ. физ. фак. Изд. 2, дополн. — Саратов: Научная книга, 2011. — 120 с.: ил.

[8] Брюханов Ю.А., Приоров А.Л., Джиган В.И. Основы цифровой обработки сигналов: учебное пособие. — Ярославль: изд-во ЯрГУ, 2013. — 344 с.

[9] Калинин В.И., Герштейн Г.М. Введение в радиофизику. — М.: Гостехиздат, 1957. — 656 с.

[10] Рабинович М. И., Трубецков Д. И. Введение в теорию колебаний и волн. — Саратов: НИЦ «Регулярная и хаотическая динамика», 2000. — 560 с.