ФИЗИЧЕСКИЙ
ФАКУЛЬТЕТ
МГУ ИМЕНИ
М.В. ЛОМОНОСОВА

teach-in
ЛЕКЦИИ УЧЕНЫХ МГУ

ЧИСЛЕННЫЕ МЕТОДЫ

ЛУКЬЯНЕНКО ДМИТРИЙ ВИТАЛЬЕВИЧ

ФИЗФАК МГУ

Содержание

Лекция 1

Тема этой лекции – нелинейные уравнения. Мы будем рассматривать вопрос о том, как решить нелинейное уравнение вида

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

Уравнение достаточно простое, но на его примере можно продемонстрировать многие проблемы, с которыми можно столкнуться при использовании численных методов.

Мы будем исходить из того, что аналитическое решение нам не известно (потому что если это не так, то численные методы использовать просто бессмысленно). Более того, будем предполагать, что график функции мы нарисовать также не можем. График может быть построен в том случае, если мы знаем значения функции во многих точках, а значит, можем приблизительно найти решение. На практике каждая операция вычисления значения функции $f(x)$ в конкретной точке x может быть очень сложной и долгой, поэтому будем стараться строить алгоритмы с минимальным количеством таких операций.

Метод дихотомии (метод деления отрезка пополам)

Алгоритм

1. Выберем такой отрезок $[x_0, x_1]$, на границах которого функция $f(x)$ принимает значения разного знака (т.е. $f(x_0) \cdot f(x_1) < 0$).
2. Разобьём отрезок пополам и выберем тот отрезок, на которой функция вновь принимает значения разного знака.
3. Повторим шаг 2, пока не будет достигнута заданная точность.

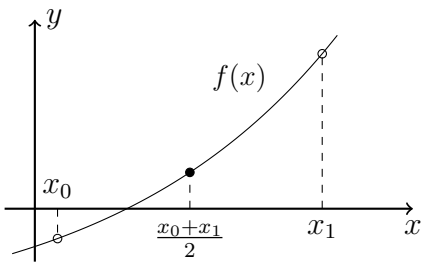


Рис. 1.1: Метод дихотомии.

Получаем, что на каждой итерации необходимо вычислить значение функции в одной точке – $\frac{x_0+x_1}{2}$.

Из построения видно, что процесс будет сходиться к точному решению. Для этого функция $f(x)$ должна быть непрерывной (Теорема из курса мат. анализа: если $f(x)$ – непрерывна, и $f(x_0) \cdot f(x_1) < 0$, то $f(x)$ пересекает ось OX на отрезке $[x_0, x_1]$). На каждой итерации мы выбираем отрезок, на котором лежит $\overline{x}$ – корень уравнения, а длина отрезка уменьшается в два раза.

При бесконечном повторении шага 2 алгоритм сойдётся к точному решению, но при решении реальных задач нам нужно найти решение за ограниченное время, и не обязательно с абсолютной точностью. Пусть известна допустимая точность ε – максимальное отклонение найденного решения от точного.

Критерий остановки

$$|x_0 - x_1| \leq \varepsilon. \tag{1.2}$$

После этого мы можем брать последнюю найденную точку как решение уравнения, найденного с точностью *не хуже*, чем ε (точность будет равна ε, если точное решение лежит строго на одной из границ отрезка, но реально-то оно лежит внутри отрезка).

Метод дихотомии – один из самых простых в реализации, пример программы доступен в материалах к лекциям под номером 1-1 (все листинги программ можно найти по ссылке `https://teach-in.ru/course/numerical-methods-lukyanenko/material`, первое число соответсвует номеру лекции, а второе – номеру примера).

Рассмотрим работу программы на примере функции $f(x) = (x-1)(x-2)^2(x-3)^3$.

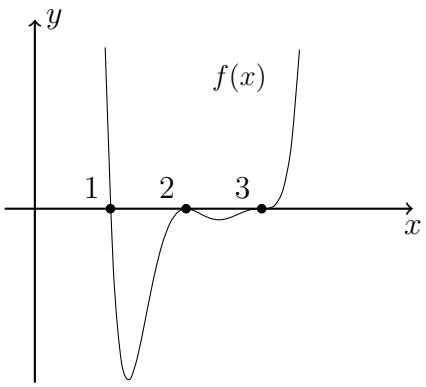


Рис. 1.2: График функции $f(x) = (x-1)(x-2)^2(x-3)^3$.

Естественно, из вида функции сразу понятно, какие у неё корни, и в реальности никому бы и в голову не пришло искать корни методом дихотомии. Но работу численных методов лучше проверять на задачах, решения которых заранее известны.

Запустим программу с $\varepsilon = 0.1$ на отрезке $[2.1, 3.5]$. Решение $x = 3.06$ найдётся за 4 итерации. А вот если начальные приближения выбрать так, чтобы на отрезке оказалось несколько корней, то заранее неизвестно, к какому конкретно сойдётся метод. Есть возможность исключать корни, но об этом мы поговорим позже.

Ещё одну особенность можно увидеть на примере корня $x = 2$. Так как это корень чётной кратности, в его окрестности функция $f(x)$ одного знака как справа, так и слева, а значит, методом дихотомии этот корень найти невозможно.

Исходный алгоритм не учитывает случай, когда один из концов отрезка точно попадает на корень уравнения $f(x) = 0$. Можно проводить эту проверку на каждом шаге, так как значение функции все равно уже посчитано.

Также необходимо помнить об ошибках округления. В приведённом примере не имеет смысла брать больше 50-ти итераций.

Метод простой итерации

Нам необходимо решить уравнение вида $f(x) = 0$. Сопоставим ему эквивалентное уравнение вида

$$x = \varphi(x), \tag{1.3}$$

то есть новое уравнение будет иметь такие же корни, что и исходное. Вариантов

переписать исходное уравнение в таком виде – бесконечно много (например, можно взять $\varphi(x) = f(x) + x$), но не всякий выбор $\varphi(x)$ будет удачен.

Алгоритм

1. Выберем x_0 – любое.
2. Будем искать последующее приближение как $x_{n+1} = \varphi(x_n)$, при $n = 0, 1, 2, ...$

Если последовательность $\{x_n\}$ сходится, то $\overline{x} = \lim\limits_{n\to\infty} x_n$ – корень уравнения $f(\overline{x}) = 0$.

Действительно, если φ – непрерывна, то при $n \to \infty$ получаем

$$\lim_{n\to\infty} x_{n+1} = \lim_{n\to\infty} \varphi(x_n) \quad \Rightarrow \quad \overline{x} = \varphi(\overline{x}).$$

Теперь рассмотрим, при каких функциях $\varphi(x)$ итерационный процесс сходится. Потребуем, чтобы $\varphi(x)$ была непрерывно-дифференцируема.

$$\begin{cases} \overline{x} = \varphi(\overline{x}) \\ x_{n+1} = \varphi(x_n) \end{cases}$$

Вычтем из верхнего уравнения нижнее и возьмём модуль от обеих частей:

$$|\overline{x} - x_{n+1}| = |\varphi(\overline{x}) - \varphi(x_n)| = \text{ // по формуле конечных приращений Лагранжа // } =$$

$$= |\varphi'(x^*)| \cdot |\overline{x} - x_n|, \text{ где } x^* \in [x_n, \overline{x}].$$

Метод будет сходиться, если решение на $n + 1$ итерации будет ближе к точному решению, чем решение на n итерации, то есть $|\overline{x} - x_{n+1}| < |\overline{x} - x_n|$. Тогда

$$|\varphi'(x^*)| < 1 \tag{1.4}$$

(неравенство строгое, так как в противном случае $|\overline{x} - x_n|$ может все время оставаться постоянным). Более того, если $\overline{x}$ – корень, то $\varphi(\overline{x}) = 0$.

В качестве примера рассмотрим классическую задачу о вычислении корня числа

$$x = \sqrt{a}.$$

Можно было бы предложить переписать это уравнение в виде $f(x) \equiv x - \sqrt{a} = 0$, но мы, наоборот, хотим избежать операции извлечения корня, поэтому запишем уравнение в виде

$$f(x) \equiv x^2 - a = 0.$$

Теперь будем искать удачный вид функции $\varphi(x)$. Например, возьмём $\varphi(x) \equiv a/x$. Тогда при начальном приближении x_0

$$x_1 = \frac{a}{x_0}; \quad x_2 = \frac{a}{x_1} = \frac{a}{\frac{a}{x_0}} = x_0,$$

процесс зацикливается. Это связано с тем, что $\varphi'(\sqrt{a}) = -a/x^2|_{x=\sqrt{a}} = -1 \neq 0$.

Рассмотрим теперь другой вид $\varphi(x)$. Этот вариант был реализован в первых калькуляторах.

$$x = \underbrace{\frac{1}{2}\left(x + \frac{a}{x}\right)}_{\varphi(x)}.$$

Например, если взять $a = 4$, $x_0 = 10$, то уже на 5 итерации $|2 - x_5| < 10^{-5}$. Это связано с тем, что

$$\varphi'(\sqrt{a}) = \frac{1}{2}\left(1 - \frac{a}{x^2}\right)\Big|_{x=\sqrt{a}} = 0.$$

В критерии прекращения итерационного процесса, по сравнению с методом дихотомии, будут существенные отличия. В методе дихотомии корень *всегда* лежит внутри известного отрезка, поэтому оценка погрешности гарантирована. В методе простой итерации корень не обязан лежать между x_n и x_{n+1}, поэтому критерий $|x_n - x_{n+1}| < \varepsilon$ **неверен**.

Запишем

$$\begin{cases} x_{n+1} = \varphi(x_n) \\ x_n = \varphi(x_{n-1}) \end{cases}$$

и получим

$$|x_{n+1} - x_n| = \underbrace{|\varphi'(x^*)|}_{q} \cdot |x_n - x_{n-1}|.$$

Предположим, что процесс сходящийся ($q < 1$) и $q = const$ (очевидно, что это не так). Тогда можно рассматривать $|x_{n+1} - x_n|$ как члены бесконечно-убывающей геометрической прогрессии.

Оценим

$$|\overline{x} - x_n| \leq \sum_{m=n}^{\infty} |x_{m+1} - x_m| = |x_{n+1} - x_n| \sum_{k=0}^{\infty} q^k = \frac{|x_{n+1} - x_n|}{1 - q} = \left| \frac{x_{n+1} - x_n}{1 - \frac{x_{n+1} - x_n}{x_n - x_{n-1}}} \right|.$$

Критерий остановки

$$\left| \frac{x_{n+1} - x_n}{1 - \frac{x_{n+1} - x_n}{x_n - x_{n-1}}} \right| \leq \varepsilon. \tag{1.5}$$

Метод Ньютона

Разложим функцию $f(x)$ в ряд Тейлора в окрестности некоторой точки x_n:

$$f(x) \approx f(x_n) + f'(x_n)(x - x_n).$$

Потребуем $f(x_{n+1}) = 0$. Тогда

$$0 = f(x_{n+1}) \approx f(x_n) + f'(x_n)(x_{n+1} - x_n) \quad \Rightarrow \quad x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}. \tag{1.6}$$

Алгоритм

1. Выбираем x_0 – любое.
2. Применяем формулу (1.6) для $n = 0, 1, 2, ...$

У этой формулы есть простая геометрическая интерпретация: в каждой точке мы ищем касательную прямую, и следующее приближение ищем как пересечение касательной и оси OX.

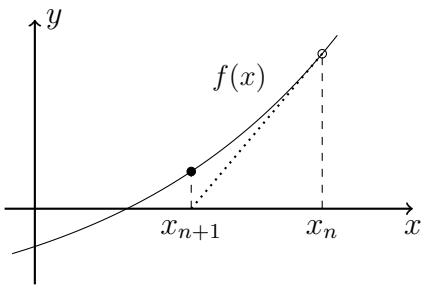


Рис. 1.3: Метод Ньютона.

Так как в формулах встречается производная $f'(x)$, логично также требовать её непрерывности.

Метод Ньютона является частным случаем метода простой итерации, так как в данном случае

$$\varphi(x) = x - \frac{f(x)}{f'(x)}.$$

Сделаем проверку:

$$\varphi'(x) = 1 - \frac{f'(x)f'(x) - f(x)f''(x)}{(f'(x))^2} = \frac{f(x)f''(x)}{(f'(x))^2},$$
$$\varphi'(\overline{x}) = \frac{f(\overline{x})f''(\overline{x})}{(f'(\overline{x}))^2} = 0, \text{ т.к. } f(\overline{x}) = 0.$$

Так как метод Ньютона – частный случай метода простой итерации, критерий прекращения итерационного процесса будет тем же. Ещё раз напоминаем, что критерий $|x_n - x_{n+1}| < \varepsilon$ в данном случае **неверен**.

Критерий остановки

$$\left| \frac{x_{n+1} - x_n}{1 - \frac{x_{n+1} - x_n}{x_n - x_{n-1}}} \right| \leq \varepsilon. \tag{1.7}$$

Для линейной функции $f(x)$ метод Ньютона находит точное решение с любым начальным приближением за одну итерацию.

Если на n-ой итерации мы попадаем в точку локального экстремума, то $f'(x_n) = 0$ и метод не работает (ноль в знаменателе).

Метод дихотомии называют *методом нулевого порядка*, а метод Ньютона – *методом первого порядка*. Эта терминология связана с максимальным порядком производной, которая используется.

Практически к любому численному методу можно придумать контрпример, на котором метод не будет работать. Пример такой функции можно увидеть на рис. 1.4.

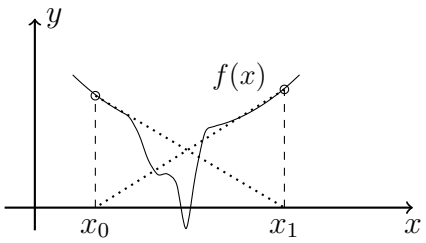


Рис. 1.4: Контрпример к методу Ньютона.

При неудачном выборе начального приближения x_0 последующие итерации x_n будут равняться либо x_0, либо x_1.

Исключение найденных корней

Пусть найден корень уравнения $\overline{x}$, то есть $f(\overline{x}) = 0$. Чтобы исключить найденный корень, можно разложить исходную функцию на множители:

$$f(x) = (x - \overline{x})\,\tilde{f}(x).$$

Тогда можно продолжать искать корни, заменив исходное уравнение новым:

$$\tilde{f}(x) \equiv \frac{f(x)}{(x - \overline{x})} = 0. \tag{1.8}$$

Далее, после нахождения следующего корня, можно повторить процесс.

Если запустить алгоритм и проверить его работоспособность на примере функции $f(x) = (x-1)(x-2)^2(x-3)^3$, то будут найдены следующие корни (при $\varepsilon = 0.1$, начальное приближение $= 4$): $x_0 = 3.06,\ x_1 = 3.26,\ x_2 = 2.03,\ x_3 = 3.32,\ x_4 = 3.09,\ x_5 = 1.00$. С учётом погрешности видно, что некоторые корни (например, 3.06 и 3.26) нельзя считать равными.

Можно модифицировать алгоритм, чтобы учитывать кратность корня.

Диагностика кратности корня в методе Ньютона

Пусть найден корень $\overline{x}$ кратности p, при этом p – неизвестно. Тогда в окрестности точки $\overline{x}$

$$f(x) \approx c(x-\overline{x})^p.$$

Тогда

$$f'(x) \approx cp(x-\overline{x})^{p-1},$$
$$f''(x) \approx cp(p-1)(x-\overline{x})^{p-2}.$$

Для метода Ньютона получаем

$$q = |\varphi'(x)| = \left|\frac{f(x)f''(x)}{(f'(x))^2}\right| = \left|\frac{c(x-\overline{x})^p cp(p-1)(x-\overline{x})^{p-2}}{(cp(x-\overline{x})^{p-1})^2}\right| = \left|\frac{c^2p(p-1)(x-\overline{x})^{2p-2}}{c^2p^2(x-\overline{x})^{2p-2}}\right| = \left|\frac{p-1}{p}\right|,$$

$$p = \frac{1}{1-q} = \frac{1}{1-\frac{x_{n+1}-x_n}{x_n-x_{n-1}}}. \tag{1.9}$$

То есть после нахождения корня мы по последним трём итерациям можем оценить его кратность.

Пример программной реализации можно найти в материалах к лекциям под номером 1-2.

Запустим программу для функции $f(x) = (x-1)(x-2)^2(x-3)^3$ при $\varepsilon = 0.1$, нулевое приближение $x_0 = 4$. Получим следующий результат:

Найденный корень	Кратность корня
3.055	3.448
$2.027 + 1.6i \times 10^{-18}$	2.180
$1080.7 - 3.9i \times 10^{-14}$	0.372

Видно, что последний корень восстанавливается не очень адекватно. Дело в том, что кратность каждого корня получилась немного больше, чем заданная, поэтому на последней итерации поиска корня получаем слишком пологую функцию

$$\tilde{f}(x) = \frac{(x-1)(x-2)^2(x-3)^3}{(x-3.055)^{3.448}(x-2.027)^{2.180}},$$

степень полинома можно оценить как $1 + 2 + 3 - 3.448 - 2.180 = 0.372$. Первая же итерация метода Ньютона начинает идти не в ту сторону из-за неудачного начального приближения (см. рис. 1.5).

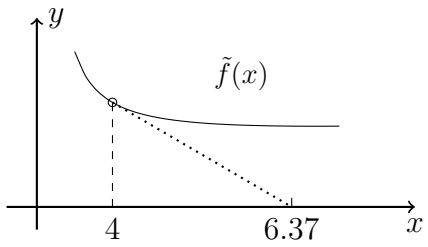


Рис. 1.5: Поиск последнего корня.

Если увеличить точность и взять $\varepsilon = 0.01$, то все корни восстановятся довольно точно. Нужно помнить, однако, что эта функция – модельная, а значит, при решении реальной задачи мы бы не знали, как сильно нужно увеличивать точность, чтобы получить правильный результат.

Нужно уточнить, что кратность корня нельзя округлять до целых чисел. Например, кратность корня может равняться $2/3$ для корня $x = 1$ уравнения $f(x) = (x - 1)^{2/3} = 0$.

Обобщённый метод Ньютона

Рассмотрим теперь работу метода Ньютона на примере функции $f(x) = e^x - 2$, выберем начальное приближение $x_0 = -2$. На первой же итерации происходит достаточно большой скачок за корень (см. рис. 1.6).

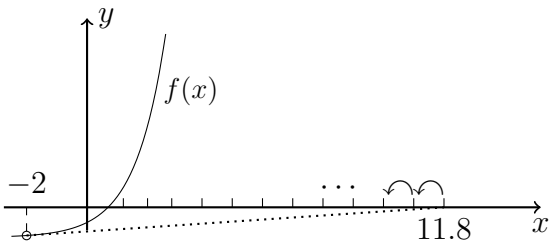


Рис. 1.6: Метод Ньютона для функции $f(x) = e^x - 2$.

На таких больших значениях экспонента очень быстро растёт, поэтому касательные

будут практически перпендикулярны оси OX. Возвращение к корню будет происходить крайне медленно.

Говорят, что метод Ньютона *неустойчив по отношению к выбору начального приближения*, то есть количество итераций, которое требуется, чтобы метод Ньютона сошёлся к правильному ответу с заданной точностью, может значительно зависеть от выбора начального приближения.

Изменим формулу метода Ньютона (1.6), добавив дополнительный коэффициент τ_n:

$$x_{n+1} = x_n + \tau_n \underbrace{\left(-\frac{f(x_n)}{f'(x_n)}\right)}_{\xi_n} = x_n + \tau_n \xi_n, \text{ при } n = 0, 1, 2, ...$$

ξ_n – шаг на каждой итерации. Будем вычислять τ_n так, чтобы вблизи корня $\tau_n \to 1$, а если шаг ξ_n будет достаточно велик, чтобы τ_n его нивелировало.

Исходная задача поиска корня уравнения $f(x) = 0$ эквивалентна поиску корня уравнения

$$f^2(x) = 0.$$

Введём функцию

$$\psi(\tau) = f^2(x_n + \tau\xi_n). \tag{1.10}$$

Тогда

$$\psi'(0) = 2f(x_n + \tau\xi_n)f'(x_n + \tau\xi_n)\xi_n|_{\tau=0} = 2f(x_n)f'(x_n)\underbrace{\left(-\frac{f(x_n)}{f'(x_n)}\right)}_{\xi_n} = -2f^2(x_n) = -2\psi(0).$$

$$\psi(\tau) = \psi(0) + \psi'(0)\tau + \mathcal{O}(\tau^2) = \psi(0) - 2\psi(0)\tau + \mathcal{O}(\tau^2) \approx \psi(0) - 2\psi(0)\tau + a\tau^2.$$

По сути мы приблизили исходную функцию параболой, совпадающей с исходной функцией в точке $\tau = 0$. Так как нас интересует диапазон шагов $\tau \in [0, 1]$, потребуем также совпадения в точке $\tau = 1$ за счёт произвольного a:

$$\psi(1) = \psi(0) - 2\psi(0)\tau + a\tau^2\big|_{\tau=1} = \psi(0) - 2\psi(0) + a \;\; \Rightarrow \;\; a = \psi(0) + \psi(1).$$

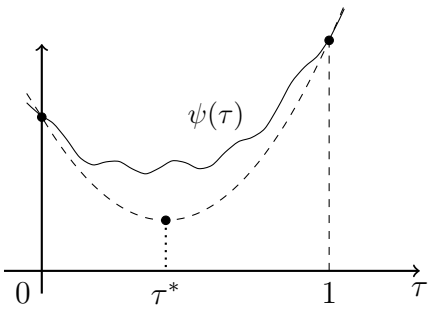


Рис. 1.7: Приближение функции $\psi(\tau)$ для поиска минимума.

Можно легко найти

$$\tau^* = \underset{\tau\in[0,1]}{\operatorname{argmin}} \left(\psi(0) - 2\psi(0)\tau + a\tau^2\right) = \frac{\psi(0)}{\psi(0)+\psi(1)}. \tag{1.11}$$

Алгоритм

1. $n = 0$: выберем x_0 – любое.
2. Вычислим $\tilde{x}_{n+1} = x_n + \xi_n$, где $\xi_n = -\dfrac{f(x_n)}{f'(x_n)}$,
3. Найдём $\tau_n = \dfrac{\psi(0)}{\psi(0)+\psi(1)} = \dfrac{f^2(x_n)}{f^2(x_n)+f^2(\tilde{x}_{n+1})}$,
4. Пересчитаем $x_{n+1} = x_n + \tau_n\xi_n$,
5. Вернёмся к шагу 2 с заменой n на $n+1$.

Проверим такой вид формулы для поиска шага на правильность. Пусть мы приближаемся к корню, тогда $|f(\tilde{x}_{n+1})| < |f(x_n)|$, в предельном случае $f^2(\tilde{x}_{n+1}) \ll f^2(x_n)$, а значит,

$$\tau_n = \frac{f^2(x_n)}{f^2(x_n)+f^2(\tilde{x}_{n+1})} \approx \frac{f^2(x_n)}{f^2(x_n)} = 1 \quad \Rightarrow \quad \text{обычный метод Ньютона.}$$

Пусть на каком-то шаге нас, наоборот, далеко отбросило от искомого корня. Тогда $|f(\tilde{x}_{n+1})| \gg |f(x_n)|$, получаем

$$\tau_n = \frac{f^2(x_n)}{f^2(x_n)+f^2(\tilde{x}_{n+1})} \approx 0,$$

чего мы и добивались.

Однако, на том же самом примере с экспонентой можно показать, что $f(\tilde{x}_{n+1})$ будет настолько велико, что $\tau_n \xi_n \approx 0$. Да, корень мы перескакивать не будем, но сходиться будет все равно очень медленно.

Добавим "кухонную" поправку в числитель:

$$\tau = \frac{(0) + \theta\psi(1)}{(0) + \psi(1)}. \tag{1.12}$$

Можно взять, например, $\theta = 0.05$. Тогда при $\psi(1) \gg \psi(0)$ шаг будет фиксированным $\tau = \theta$, а не бесконечно малым. Если выбрать $\theta = 1$, то метод вырождается в обычный метод Ньютона.

Пример программы можно найти в материалах к лекциям под номером 1-3.

Существует множество других методов для решения нелинейных уравнений. Например: *метод секущих*, на каждой итерации которого проводится не касательная прямая, а секущая (метод – двухшаговый, так как на каждой итерации нужно две точки, чтобы провести секущую). Также существует *метод параболы* – трёхшаговый метод, который может сойтись к комплекснозначному корню при действительном начальном приближении.

Системы нелинейных уравнений решаются только методом Ньютона, а решению линейных систем посвящена следующая лекция.

Лекция 2

Лекция посвящена решению систем линейных алгебраических уравнений

$$Ax = b. \tag{2.1}$$

Основные вопросы, которые мы будем рассматривать: 1) решение системы линейных алгебраических уравнений, 2) как найти определитель матрицы данной системы, 3) как найти обратную матрицу и 4) как найти спектр матрицы (набор её собственных значений) и её собственные векторы. Вопрос №4 в данной лекции не рассматривается, он будет рассмотрен позже.

Метод Гаусса

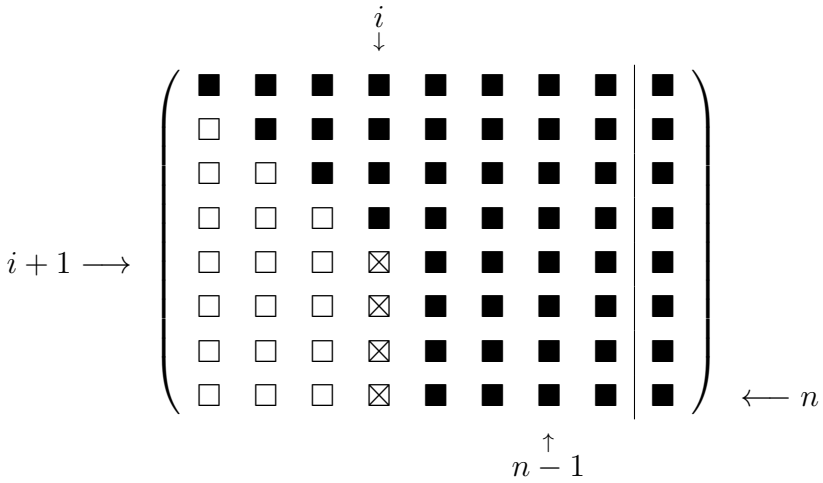
Рассмотрим вопрос о решении системы с произвольной матрицей. Для такой системы нет более универсального метода, чем метод Гаусса. Алгоритм данного метода подробно рассматривался на младших курсах. Подсчитаем количество операций, которое нужно совершить, чтобы найти решение системы с помощью метода Гаусса. Выпишем расширенную матрицу системы (2.1)

$$(A|\,b) = \left(\begin{array}{cccccccc|c}
\blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\
\square & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\
\blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\
\blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\
\blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\
\blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\
\blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\
\blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare
\end{array}\right) \tag{2.2}$$

Метод Гаусса часто разделяют на два этапа. Первый этап – это прямой ход, когда целью является приведение расширенной матрицы системы к верхнетреугольному виду. Второй

этап – это обратный ход, когда требуется исключить все элементы, лежащие выше главной диагонали, а затем найти решение системы.

Рассмотрим прямой ход метода Гаусса. В каждом столбце, начиная с первого, последовательно исключаем все элементы, лежащие ниже главной диагонали. Например, первый элемент второй строки (2.2) занулится, если из второй строки вычесть первую, умноженную на отношение первого элемента второй строки к первому элементу первой строки. Далее в первом столбце аналогичным образом зануляются оставшиеся элементы, лежащие ниже главной диагонали. Данную операцию нужно повторить с каждым столбцом. В результате все элементы ниже главной диагонали окажутся равными нулю, а оставшиеся элементы расширенной матрицы изменяться. Посчитаем количество операций, необходимое, чтобы реализовать эти действия. Нужно пробежаться по всем столбцам с 1 по $n-1$, где n - размерность матрицы системы (верхний предел $n-1$, потому что последний элемент, который занулится, расположен в $n-1$ столбце), и сложить вместе количество действий на отдельных столбцах. Допустим, что сейчас обнуляются все элементы под главной диагональю в столбце с номером i. Тогда обнулить элементы нужно в строках с $i+1$ по n.



(2.3)

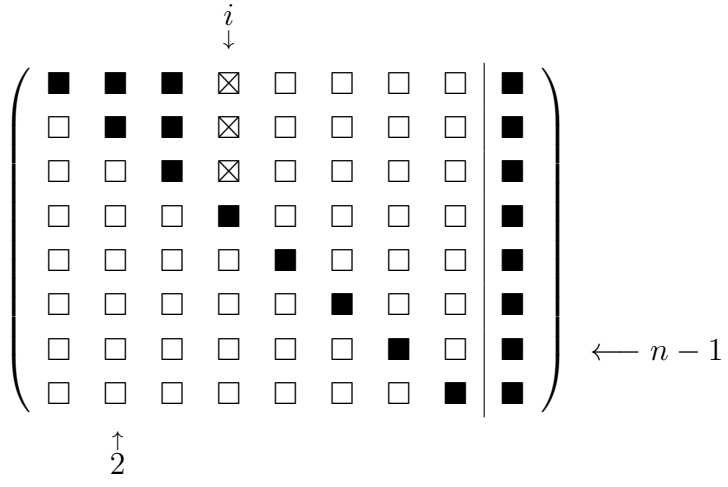
Найдём число операций в столбце с номером i для зануления элемента под диагональю. Одна операция идёт на вычисление коэффициента, равного отношению соответствующего элемента на диагональный элемент. Далее нужно умножить часть i строки матрицы системы на данный коэффициент. Это займёт $n-i$ операций, кроме того, нужно умножить элемент правой части, а значит добавится ещё одна операция. Затем часть i – той строки расширенной матрицы нужно вычесть из $i+1$ строки. Эта процедура занимает $(n-i)+1$ операций, так как вычитание в i столбце тривиально и, соответственно, не учитывается. Вернёмся

к вычислению сложности прямого хода. Имеем

$$
\begin{aligned}
&\sum_{i=1}^{n-1}\sum_{j=i+1}^{n}\left(1+(n-i)+1+(n-i)+1\right)= \\
=&\sum_{i=1}^{n-1}\sum_{j=i+1}^{n}\left(2n-2i+3\right)= \\
=&\sum_{i=1}^{n-1}\left(2n-2i+3\right)(n-i)= \\
=&\sum_{i=1}^{n-1}\left(2n^2-2in+3n-2in+2i^2-3i\right)= \\
=&\sum_{i=1}^{n-1}\left(2n^2+3n\right)+\sum_{i=1}^{n-1}\left(-4n-3\right)i+\sum_{i=1}^{n-1}(2)\,i^2= \\
=&\left(2n^2+3n\right)(n-1)+(-4n-3)\,\frac{1+(n-1)}{2}\,(n-1)+ \\
&+2\frac{(n-1)\,(n-1+1)\,(2\,(n-1)+1)}{6}= \\
=&\frac{2}{3}n^3+\frac{1}{2}n^2-\frac{7}{6}n=\mathcal{O}\left(n^3\right).
\end{aligned}
\tag{2.4}
$$

Мы получили очень важную формулу. Она выражает число операций, которые нужно совершить, чтобы привести расширенную матрицу системы к верхнетреугольному виду. Отметим, что сложность данной процедуры пропорциональна размерности системы, возведённой в куб.

После того как расширенная матрица приведена к верхнетреугольному виду, начинается второй этап - обратный ход. Теперь нужно обнулить все элементы матрицы системы, которые расположены над главной диагональю. Будем действовать аналогично прямому ходу и посчитаем число операций, необходимых для зануления первого элемента над диагональю, в столбце с номером i.



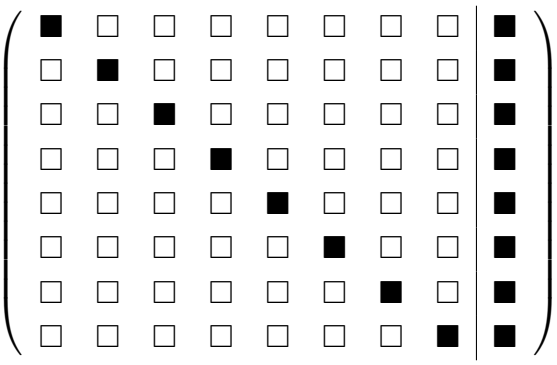
(2.5)

Для зануления данного элемента нужно будет сделать 3 операции: вычислить коэффициент, равный отношению наддиагонального элемента к диагональному, умножить соответствующий элемент из правой части на этот коэффициент и, наконец, вычесть из $i+1$ строки i строку (на самом деле вычитание осуществляется только для одного числа - элемента правой части). В i столбце эту процедуру нужно будет повторить $i-1$ раз. Таким образом, суммарная сложность для обратного хода будет равна

$$\sum_{i=n}^{2}(1+1+1)(i-1)=3\sum_{i=n}^{2}i-3(n-1)=\frac{3}{2}n^2-\frac{3}{2}n=\mathcal{O}\left(n^2\right). \tag{2.6}$$

Сделаем важное замечание: суммарная сложность обратного ходя метода Гаусса пропорциональна квадрату размерности матрицы системы.

После осуществления двух этапов метода Гаусса матрица расширенной системы примет следующий вид



(2.7)

Для получения решения x из (2.1) нужно поделить элементы из правой части на соответствующие элементы, стоящие на диагонали. Таких операций будет ровно n штук. Подсчитаем общую сложность решения системы (2.1) с помощью метода Гаусса

$$\underbrace{\mathcal{O}\left(n^3\right)}_{\text{прямой ход}}+\underbrace{\mathcal{O}\left(n^2\right)}_{\text{обратный ход}}+\underbrace{n}_{\text{финальный этап}}=\mathcal{O}\left(n^3\right). \tag{2.8}$$

Заметим, что одной из подзадач для (2.1) является нахождение определителя матрицы системы. Видно, что в данном случае определитель системы легко найти сразу после завершения прямого хода метода Гаусса, так как определитель верхнетреугольной матрицы равен произведению диагональных элементов. Сложность этой подзадачи оказывается равна $\mathcal{O}\left(n^3\right)$.

В этой лекции мы подробно подсчитали сложность для метода Гаусса решения системы линейных алгебраических уравнений. Эта информация пригодится в будущем, когда мы будем сравнивать метод Гаусса с другими методами решения данной задачи, например, с градиентными методами.

С помощью метода Гаусса можно найти обратную матрицу к A из (2.1). Рассмотрим следующую формулу

$$AA^{-1} = E. \tag{2.9}$$

Видно, что для поиска столбцов обратной матрицы возникает система линейных алгебраических уравнений с исходной матрицей и правой частью, в которой стоит столбец единичной матрицы. На первый взгляд кажется, что сложность вычисления обратной матрицы составляет $n\mathcal{O}\left(n^3\right) = \mathcal{O}\left(n^4\right)$, но это не так! Так как матрица система каждый раз будет одинаковая, то прямой ход метода Гаусса можно реализовать одновременно для всех столбцов единичной матрицы, то есть для всех правых частей. Таким образом, трудоёмкость прямого хода метода Гаусса не увеличивается (в смысле главного члена), а трудоёмкость обратного хода метода Гаусса станет пропорциональна $\mathcal{O}\left(n^3\right)$. Суммарная сложность задачи по вычислению обратной матрицы составляет $\mathcal{O}\left(n^3\right)$.

Плохообусловленные системы.

Рассмотрим систему уравнений для двух уравнений.

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$$

Геометрически эта система задает точку пересечения прямых (Рис. 2.1). Естественно у нас могут быть ошибки либо в самой матрице системы, либо в ее правой части, если речь идет об эксперименте. Наличие ошибки в правой части приводит к параллельному сдвигу,

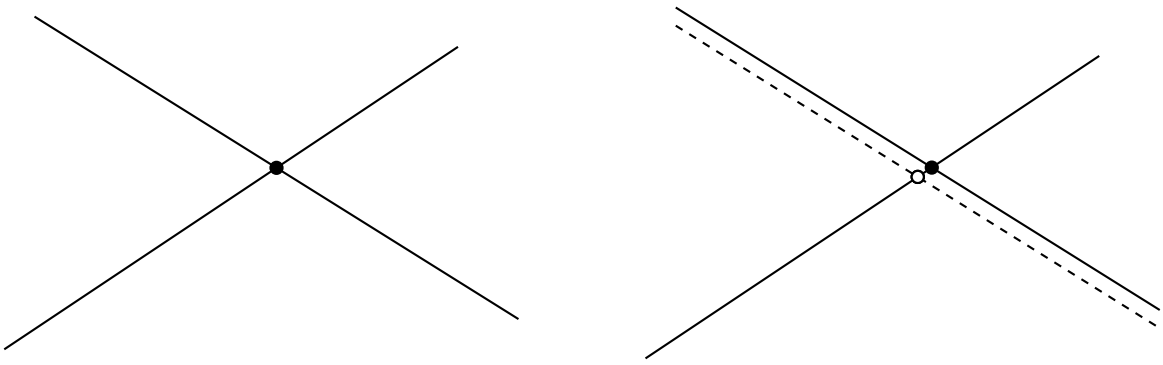


Рис. 2.1: геометрическая интерпретация решения системы двух уравнения в случае задания правой части без погрешности (слева) и в случае наличия погрешности (справа)

маленькое значение погрешности соотвествует к малому изменению решения (Рис. 2.1).

Однако может произойти следующая ситуация, если прямые пересекаются под очень острым углом.

Рис. 2.2: иллюстрация плохой обусловленности системы двух уравнений

Небольшой параллельный сдвиг в этом случае дает достаточно сильное изменение решения (Рис. 2.2). Такие системы называются плохообусловленными.

Возникает вопрос, если у нас разрешимая система, мы все вычисления делаем точно, то откуда возьмутся ошибки? Идея заключается в следующем. Допустим, мы реализуем метод Гаусса, все вычисления делаются на компьютере, это ключевой момент. При вычислениях на компьютере есть такая проблема, как проблема ошибок машинного округления, вычисления делаются не точно, а раз так, то уже ошибка может возникать в процессе счета, поэтому некоторые системы и обладают плохой обусловленностью. Пример такой системы построил Гильберт. Рассмотрим так называемую матрицу Гильберта, элементы которой задаются следующей формулой

$$a_{ij} = \frac{1}{i+j-1}, \quad i,j = \overline{1,n} \tag{2.10}$$

умножим эту матрицу (для случая $n = 4$) на вектор из единиц

$$Gx = \begin{pmatrix} 1 & 1/2 & 1/3 & 1/4 \\ 1/2 & 1/3 & 1/4 & 1/5 \\ 1/3 & 1/4 & 1/5 & 1/6 \\ 1/4 & 1/5 & 1/6 & 1/7 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} \square \\ \square \\ \square \\ \square \end{pmatrix} = y$$

Умножили на компьютере, получили какой-то вектор y. Теперь с помощью метода Гаусса, найдём решение такой системы (x считаем неизвестным):

$$Gx = y$$

Мы ожидаем, что x получится вектором из единиц, но оказывается, что если порядок матрицы Гильберта становится примерно $n = 10 \sim 12$, то значения в векторе x сильно отличаются от единиц (может получится 3, -7 и так далее). Это как раз связано с тем, что в процессе реализации метода Гаусса, возникают ошибки машинного округления, они накапливаются и приводят к этим эффектам. Конечно, если проделать все эти операции вручную, ничего не округляя, то на выходе мы получим тот же самый x.

Метод прогонки.

При решение физических задач чрезвычайно часто встречаются системы у которых матрицы имеют 3 диагональный вид, такие системы выглядят следующим образом

$$\begin{pmatrix} -b_1 & c_1 & \dots & \dots & 0 \\ a_2 & -b_2 & c_2 & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & a_{n-1} & -b_{n-1} & c_{n-1} \\ 0 & \dots & \dots & a_n & -b_n \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \dots \\ x_{n-1} \\ x_n \end{pmatrix} = \begin{pmatrix} d_1 \\ d_2 \\ \dots \\ d_{n-1} \\ d_n \end{pmatrix}$$

Знак минус перед элементами b_i не играет принципиальной роли, это всего лишь дань традиции. Конечно, такую матрицу можно решать методом, Гаусса, но тогда придется выполнить n^3 операций. Однако можно написать частный случай метода Гаусса. Давайте напишем произвольное уравнение i-ой строки

$$a_i x_{i-1} - b_i x_i + c_i x_{i+1} = d_i \quad i = \overline{1,n} \tag{2.11}$$

Заметим, что строки при $i = 1$ и $i = n$ имеют только два ненулевых элемента, т.е. $a_1 \equiv 0$, $c_n \equiv 0$.

Идея метода прогонки

Мысленно проделаем прямой ход Метода Гаусса, то есть с помощью линейных преобразований аннулируем коэффициенты a_i для всех $i = \overline{1,n}$, тогда, конечно, каким-то образом изменятся коэффициенты b_i, c_i и d_i. После этого действия каждое уравнение системы имеет две неизвестных — x_i и x_{i+1}, $i = \overline{1,n}$. Выразим x_i через x_{i+1}.

$$x_i = \xi_{i+1} x_{i+1} + \eta_{i+1} \tag{2.12}$$

Коэффициенты ξ_i и η_i нам неизвестны, после прямого хода метода гаусса коэффициенты изменились, но как именно, пока не знаем. Это уравнение называется *формулой обратного хода* и оно справедливо при $i = \overline{1,n}$ (формально, при $i = n$ возникает x_{n+1}, но это лишь означает, что $\xi_n = 0$, но об этом позднее), напишем его с индексом на единицу меньше

$$x_{i-1} = \xi_i x_i + \eta_i \tag{2.13}$$

подставим в (2.11) и выразим x_i

$$a_i (\xi_i x_i + \eta_i) - b_i x_i + c_i x_{i+1} = d_i \quad \Rightarrow \quad x_i = \frac{c_i}{b_i - a_i \xi_i} x_{i+1} + \frac{a_i \eta_i - d_i}{b_i - a_i \xi_i} \tag{2.14}$$

Теперь сравним полученное выражение с формулой (2.12), нетрудно видеть, что

$$\xi_{i+1} = \frac{c_i}{b_i - a_i\xi_i} \quad \eta_{i+1} = \frac{a_i\eta_i - d_i}{b_i - a_i\xi_i} \tag{2.15}$$

Это так называемые *формулы прямого хода*, они рекуррентные, для того чтобы найти ξ_{i+1} нужно знать ξ_i, для которого, в свою очередь — ξ_{i-1} и так далее. Однако, нам известно, что $a_1 = 0$, а значит и произведение $a_1\xi_1 = 0$, независимо от того, чему равно ξ_1, оно может быть любым, а уже далее можно найти все остальные ξ_i, аналогично η_1 можно взять любым. Таким образом, вопрос старта итерационного процесса решен.

Сейчас мы знаем все ξ_i и η_i при $i = \overline{1,n}$. Теперь нужно реализовать *обратный ход* прогонки. Еще раз выпишем формулы обратного хода

$$x_i = \xi_{i+1}x_{i+1} + \eta_{i+1}, \quad i = \overline{n,1} \tag{2.16}$$

они тоже рекуррентные и записываются в обратном порядке, т.е. с $i = n$. Как уже упоминалось выше при $i = n$ в уравнении возникает x_{n+1}, но перед ним стоит ξ_{n+1}, которое равно нулю в силу того, что $c_n = 0$ (см. ур-я (2.15)), а значит x_{n+1} можно брать любым.

Формулы метода прогонки появились в середины 50-ых годов прошлого столетия, они использовались в разных ядерных и термоядерных проектах СССР и поначалу даже были секретны. Уже потом поняли, что эти формулы являются частным случаем метода Гаусса, собственно мы эти формулы так и строили.

Сколько необходимо операций нужно выполнить, чтобы найти решение методом прогонки? Прямой ход: вычисление всех ξ_i и η_i занимает $6n$ операций. Обратный ход: для вычисление x_i нужно $2n$ операций. Итого, необходимо совершить $8n = O(n)$ операций. Их количество сильно меньше, чем в общем случае метода Гаусса, который требует $O(n^3)$ операций. Немного терминологии. Метод прогонки также называют экономичным методом, т.е. таким, число операций для вычисления каждой неизвестной которого, не зависит от размерности системы. Другими словами, если мы имеем систему размерности n, то вектор неизвестных состоит из n компонент. Метод прогонки выполнит для этого $8n$ операций, то есть на каждую компоненты приходится 8 итераций, т.е. не зависит от n.

Условие разрешимости системы

Если выполнено условие диагонального преобладания, т.е.

$$|b_i| \geq |a_i| + |c_i|\,, \; i = \overline{1,n} \tag{2.17}$$

при этом для какого-то i-го неравенство выполняется строго, то в этом случае система разрешима. Покажем это, оценим по модулю выражение для ξ_i

$$|\xi_{i+1}| = \frac{|c_i|}{|b_i - a_i\xi_i|} \leq \frac{|c_i|}{|b_i| - |a_i\xi_i|} \leq \frac{|c_i|}{|a_i| + |c_i| - |a_i\xi_i|} = \frac{|c_i|}{|c_i| + |a_i|\,(1 - |\xi_i|)} \tag{2.18}$$

$|\xi_{i+1}| < 1$ только в том случае, если $|a_i|\,(1-|\xi_i|) > 0$, это возможно только в случае $|\xi_i| < 1$. Получаем следующий результат, если $|\xi_k| < 1$, то $\forall i > k$ выполняется $|\xi_i| < 1$, а так как мы можем взять ξ_1 любым, например, $\xi_1 = 1$, то это неравенство будет выполнено для всех последующих ξ_i. Отсюда получается

$$|b_i - a_i\xi_i| > |c_i| \geq 0 \tag{2.19}$$

Выпишем еще раз формулы прямого хода

$$\xi_{i+1} = \frac{c_i}{b_i - a_i\xi_i} \quad \eta_{i+1} = \frac{a_i\eta_i - d_i}{b_i - a_i\xi_i} \tag{2.20}$$

То есть мы получили, что знаменатели этих формул отличны от нуля, значит не возникнет деления на 0, а когда он может возникнут? - если определитель матрицы системы равен нулю, т.е. задача поставлена некорректно, поэтому условие диагонального преобладания по сути является условием разрешимости, которое легко проверяется на практике.

Лекция 3

Основной вопрос, который будет рассмотрен на этой лекции, как приближённо вычислить определенный интеграл вида

$$U = \int_a^b u(x)\,dx \tag{3.1}$$

здесь предполагается, что подынтегральная функция $u(x)$ непрерывная. Если можно вычислить интеграл аналитически, то никаких численных методов применять не нужно, их необходимо использовать только в том случае, если подынтегральная функция достаточно «неприятная» и первообразную найти невозможно, либо это очень затруднительно.

Геометрический смысл определенного интеграла.

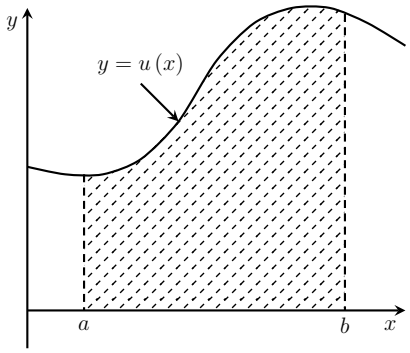


Рис. 3.1: геометрический смысл определенного интеграла.

Пусть нам дана интегрируемая функция $u(x)$ в пределах от a до b (рис.3.1). Геометрический смысл определенного интеграла — это площадь заштрихованной криволинейной трапеции. Идея приближенного интегрирования заключается в следующем. Введем на оси абсцисс так называемую сетку на отрезке $[a, b]$, будем обозначать сетку Ω_N:

$$\Omega_N = \left\{x_n,\ n = \overline{0, N},\ a = x_0 < x_1 < \ldots < x_N = b\right\}$$

$h_n = x_n - x_{n-1}$ — шаг неравномерной сетки

$h_n = h = \dfrac{b-a}{N}$ — шаг равномерной сетки

Обратите внимание, если у нас N интервалов, то сетка состоит из $N + 1$ узла, нумерация узлов начинается с нулевого.

Теперь введём интегральную сумму следующим образом:

$$U_N = \sum_{n=1}^{N} u(\tilde{x}_n) h_n, \qquad \tilde{x}_n \in [x_{n-1}, x_n] \tag{3.2}$$

Из курса математического анализа известно, что

$$\exists \lim_{\max |h_n| \to 0} U_N = U, \qquad \tilde{x}_n \in [x_{n-1}, x_n] \tag{3.3}$$

предел равняется точному значению интеграла U независимо от выбора $\tilde{x}_n$ из $[x_{n-1}, x_n]$. Посмотрим, что получится в некоторых простейших случая выбора $\tilde{x}_n$.

Некоторые конкретные схемы численного интегрирования.

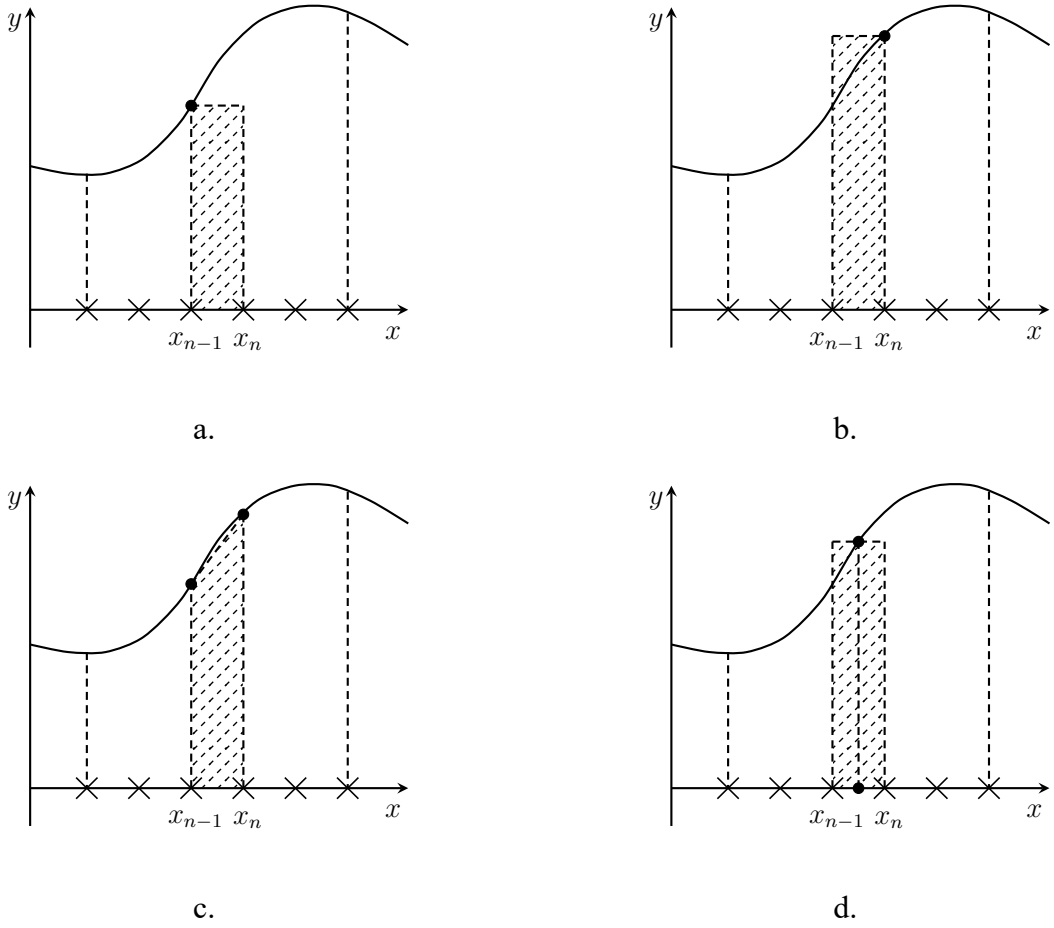


Рис. 3.2: геометрическая интерпретация формул: a) левых прямоугольников, b) правых прямоугольников, c) трапеций, d) средних прямоугольников.

Формула левых прямоугольников. Выбираем $\tilde{x}_n = x_{n-1}$, тогда U_N имеет вид

$$U_N = \sum_{n=1}^{N} u(x_{n-1})h_n \tag{3.4}$$

она представляет собой сумму площадей прямоугольников. Основание n-го прямоугольника есть отрезок $[x_{n-1}, x_n]$, а в качестве высоты выступает значение функции на *левой* границе этого отрезка. (см. рис. 3.2a.)

Формула правых прямоугольников. Выбираем $\tilde{x}_n = x_n$, тогда U_N имеет вид

$$U_N = \sum_{n=1}^{N} u(x_n)h_n \tag{3.5}$$

она представляет собой сумму площадей прямоугольников. Основание n-го прямоугольника есть отрезок $[x_{n-1}, x_n]$, а в качестве высоты выступает значение функции на *правой* границе этого отрезка. (см. рис. 3.2b.)

Формула трапеций. Из геометрических соображений можно выписать некоторое обобщение предыдущих формул

$$U_N = \sum_{n=1}^{N} \frac{u(x_{n-1}) + u(x_n)}{2} h_n \tag{3.6}$$

Эта формула не вытекает непосредственно из формулы (3.2), но из геометрических соображений понятно, что сумма таких трапеций (см. рис. 3.2c.) стремится к исходному интегралу при $N \to \infty$.

Формула средних прямоугольников. Выбираем $\tilde{x}_n = x_{n-\frac{1}{2}}$ — это середина отрезка $[x_{n-1}, x_n]$, тогда U_N имеет вид

$$U_N = \sum_{n=1}^{N} u(x_{n-\frac{1}{2}})h_n \tag{3.7}$$

она представляет собой сумму площадей прямоугольников. Основание n-го прямоугольника есть отрезок $[x_{n-1}, x_n]$, а в качестве высоты выступает значение функции на *середине* этого отрезка. (см. рис. 3.2d.). На равномерной сетке $x_n = a + hn$, в частности, можно подставить $n - \frac{1}{2}$, тем самым, получив середину соответствующего отрезка.

Все выписанные формулы для U_N из геометрических соображений являются «хорошими»: при стремлении шага сетки к нулю (или $N \to \infty$) они вырождаются в площадь исходной криволинейной трапеции, образованной подынтегральной функцией. На практике мы никогда не возьмем N равному бесконечности, потому что в этом случае пришлось бы просуммировать бесконечное число чисел, компьютер с этим никогда не справится, поэтому

возникают следующие важные вопросы. Во-первых, какое брать N? Понятно, что при одном и том же фиксированном N формулы для вычисления U_N дают результаты с разной точностью. Поэтому второй очень важный вопрос, а как сравнить эти формулы и какая формула более точная? Сейчас на этот вопрос мы с вами и ответим.

Сравнение точности.

Вопрос о том, как сравнить различные формулы, можно решить следующим образом. Надо просто вычислить ошибку, которую они дают. Понятно, что если N не бесконечность, а какое-то фиксированное число, то численный приближенный результат отличается от точного. Если вычесть из точного результата приближённый численный, то мы получим ошибку соответствующей схемы. Сравнивая ошибки различных схем, можно сделать вывод о том, какая из них более точная, а какая менее.

Точное значение интеграла мы всегда будем обозначать U, приближённое значение интеграла, вычисленное на сетке с N интервалами, мы будем обозначать U_N. В качестве примера рассмотрим интегральную формулу трапеций:

$$U = \int_a^b u(x)\,dx; \quad U_N = \sum_{n=1}^{N} \frac{u(x_{n-1}) + u(x_n)}{2} h_n \tag{3.8}$$

Посчитаем ошибку вычисления интеграла на сетке с N интервалами $R_N = U - U_N$. Заметим, что эту ошибку можно записать как

$$R_N = \sum_{n=1}^{N} r_n \tag{3.9}$$

где r_n — это ошибки полученные при вычислении интеграла на каждом отдельном отрезке $[x_{n-1}, x_n]$, её еще называют локальной ошибкой. Если мы сможем оценить r_n на каждом таком отрезке, а затем их сложим, то получим искомую ошибку R_N.

$$r_n = \int_{x_{n-1}}^{x_n} u(x)\,dx - \frac{u(x_{n-1}) + u(x_n)}{2} h_n \tag{3.10}$$

Возникает проблема со взятием интеграла, ведь функцию $u(x)$ мы не знаем, а значит первообразную найти не можем, поэтому разложим подынтегральную функцию в ряд Тейлора. Здесь можно раскладывать в окрестностях любой из точек отрезка $[x_{n-1}, x_n]$, но проще раскладывать либо в x_{n-1}, либо в x_n, так как площадь трапеции зависит как раз

от этих точек, для определенности выберем x_n, тогда раскладывать нужно $u(x)$ и $u(x_{n-1})$:

$$\begin{aligned} u(x) &= u(x_n) + u'(x_n)(x - x_n) + u''(x_n)\frac{(x - x_n)^2}{2} + \dots \\ u(x_{n-1}) &= u(x_n) + u'(x_n)(x_{n-1} - x_n) + u''(x_n)\frac{(x_{n-1} - x_n)^2}{2} + \dots \end{aligned} \tag{3.11}$$

Учтем, что $x_{n-1} - x_n = -h_n$ и введём обозначения $u_n := u(x_n)$ — это сеточные значения функции

$$\begin{aligned} &\int\limits_{x_{n-1}}^{x_n} u_n + u'_n(x - x_n) + u''_n\frac{(x - x_n)^2}{2} + \dots \, dx - \left(2u_n - u'_n h_n + \frac{u''_n}{2}h_n + \dots\right)\frac{h_n}{2} = \\ &= \left(u_n x + u'_n\frac{(x - x_n)^2}{2} + \frac{u''_n}{2}\frac{(x - x_n)^3}{3} + \dots\right)\Bigg|_{x_{n-1}}^{x_n} - u_n h_n + \frac{u'_n}{2}h_n^2 - \frac{u''_n}{4}h_n^3 + \dots = \\ &= u_n h_n - \frac{u'_n}{2}h_n^2 + \frac{u'_n}{6}h_n^3 + \dots - u_n h_n + \frac{u'_n}{2}h_n^2 - \frac{u''_n}{4}h_n^3 + \dots = -\frac{1}{12}u''_n h_n^3 + \dots \end{aligned} \tag{3.12}$$

т.е. мы получили выражение для локальной ошибки. Как мы видим эта ошибка раскладывается в ряд Тейлора по всем степеням начиная с 3-ей и равна всему, подчеркиваем, всему этому «хвостику». Члены 0-, 1- и 2-по порядков передались точно.

Общая ошибка вычисления точного значения интеграла по формуле трапеций:

$$R_N = U - U_N = \sum_{n=1}^{N} r_n = \sum_{n=1}^{N}\left(-\frac{1}{12}u''_n h_n^3 + \dots\right) = -\frac{h_n^2}{12}\sum_{n=1}^{N} u''_n h_n + \dots \tag{3.13}$$

На этом моменте мы сознательно сделали ошибку, мы вынесли из-под суммы h_n^2, так делать, конечно, нельзя, ведь суммирование ведется по n. Так сделать можно только в том случае, если сетка является равномерной и $h_n = h$. Если сетка неравномерная, так сделать уже нельзя. Кроме того предположим, что N очень велико, соответственно h очень мало и мы можем заменить сумму на интеграл

$$R_N = -\frac{h^2}{12}\int\limits_a^b u''(x)dx + \dots = -\frac{h^2}{12}\left(u'(b) - u'(a)\right) + \dots \tag{3.14}$$

Если h достаточно мало, то главный член разложения ошибки есть $-\dfrac{h^2}{12}\left(u'(b) - u'(a)\right)$, остальные слагаемые имеют более высокий порядок малости (h^3, h^4 и т.д.), поэтому ими мы пренебрегаем. Еще раз акцентируем внимание на то, что это сделать возможно только лишь в случае достаточно малого h.

Оценка ошибки R_N асимптотически точная, это означает, что формула для ошибки даёт правильный ответ только в том случае, если h стремится к нулю. Следующий момент,

если мы интегрируем конкретную функцию, то для нее $u'(a)$ и $u'(b)$ от сетки никак не зависит, результат вычислений будет зависеть только от h, поэтому формулу для ошибки можно записать как

$$R_N = O(h^2) \tag{3.15}$$

То есть ошибка пропорциональна h^2. Степень при h называют порядком точности схемы, мы получили, что у формулы трапеций 2-ой порядок точности. Чем выше порядок точности, тем точнее схема. Схема с порядком точности p передает первые p членов разложения в ряд Тейлора точного решения (члены разложения, напоминаем, начинаются с нулевого порядка).

При аналогичном исследовании можно получить, что формула средних также имеет 2-ой порядок точности, а формулы левых и правых прямоугольников — только 1-ый порядок точности. Выпишем их ошибки, считая, что h достаточно мало и можно пренебречь остаточным «хвостиком».

$$R_N^{\text{средн.}} = \frac{h^2}{24}\left(u'(b) - u'(a)\right), \quad R_N^{\text{л./п.}} = \pm\frac{h}{2}\left(u(b) - u(a)\right) \tag{3.16}$$

Мы видим, что формулы трапеций и средних имеют второй порядок точности, но абсолютная величина главного члена разложения ошибки в ряд Тейлора меньше у формулы средних, это означает, что она более точная, чем формула трапеций. Формулы левых и правых прямоугольников еще менее точные, чем две предыдущие формулы, так как имеют первый порядок точности.

Оценка ошибки R_N является не только асимптотически точной, но еще и априорной оценкой. Эту оценку мы можем посчитать еще до того, как найдем U_N. В самом деле, допустим нам нужно посчитать интеграл от какой-то известной функции. Еще до вычисления приближенного значения интеграла, можно указать оценку погрешности, достаточно найти производную подынтегральной функции на границах и шаг той сетки, на которой будем считаться интеграл. Такую оценку называют априорной, если же формула оценки погрешности можно найти только после нахождения U_N, то такая оценка называется апостериорной.

Уточнение формул приближенного интегрирования.

Обсудим еще один важный момент. Для формулы трапеций мы можем записать

$$U = U_N - \frac{h^2}{12}\left(u'(b) - u'(a)\right) + O(h^{(2+1)}) \tag{3.17}$$

Получается, что если мы вычислили приближенное значение интеграла U_N и к нему добавили главный член разложения ошибки в ряд Тейлора, то мы получим уже приближенное значение

с более высоким порядком точности. Как видим, используя формулу трапеций, можно получить 3-ий порядок точности. Уточненное значение будем обозначать $\widetilde{U}_N$, для которого справедливо соотношение

$$\widetilde{U}_N = U_N - \frac{h^2}{12}\left(u'(b) - u'(a)\right), \quad U = \widetilde{U}_N + O(h^3) \tag{3.18}$$

На самом деле, далее мы покажем, что полученный порядок точности будет равен 4-м. Опять же здесь подразумевается что h достаточно мало. И здесь стоит задаться вопросом: «h должна быть достаточно мала для чего?» — чтобы точно посчитать, но как мы узнаем, что посчитали интеграл достаточно точно, если мы не знаем точного значения интеграла? Об этом будет идти речь на следующей лекции.

Так вот, h считаем достаточно малым, тогда R_N можно воспринимать как абсолютную оценку погрешности, либо можно ввести величину $\frac{R_N}{\widetilde{U}_N}100\%$ — относительную оценку погрешности, которая опять же асимптотически точная.

В более общем случае

$$U = U_N + R_N + O(h^{p+q}) \tag{3.19}$$

В частности, $q = 1$ для формул левых и правых прямоугольников и $q = 2$ для формул средних и трапеций, и этот момент сейчас будет пояснен.

Уточнение порядка точности.

В формуле для локальной ошибки (3.12) можно явно выписать несколько последующих слагаемых, в этом случае окажется что коэффициенты перед нечетными степенями h окажутся равными нулю, откуда следует, что следующее слагаемое будет порядка $O(h^4)$.

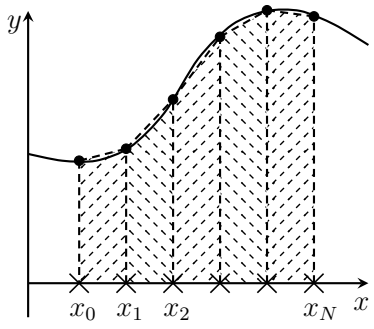


Рис. 3.3: формула трапеций с узлами, упорядоченными слева направо.

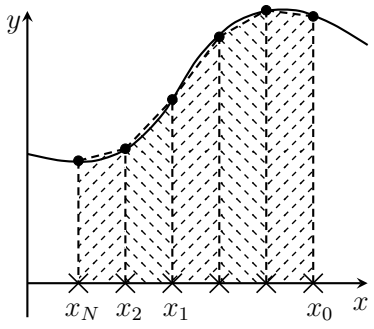


Рис. 3.4: формула трапеций с узлами, упорядоченными справа налево.

Показать, что коэффициенты перед нечетными степенями равны нулю можно из следующих соображений. Вернемся к геометрической интерпретации формулы трапеций (см. Рис. 3.3). Здесь узлы сетки пронумерованы слева направо. Что будет, если мы проведем вычисления того же интеграла на сетке с тем же самым числом интервалов, но только узлы будут упорядочены справа налево (см. Рис. 3.4)? Картинка, как мы видим, не поменялась. В первом случае величина шага была положительна $x_n - x_{n-1} =: h$, во втором случае мы сделали шаг $x_{n-1} - x_n = -h$. То есть мы h поменяли на $-h$, а результат не поменялся. В разложении в ряд Тейлора присутствуют как четные степени, так и нечетные. При замене h на $-h$ коэффициенты перед четными степенями не поменяются, но перед нечетными изменят знак. Ошибка при такой замене, не меняется, значит можем сделать вывод, что коэффициенты при нечетных должны быть равны нулю. Для формулы средних ситуация аналогичная.

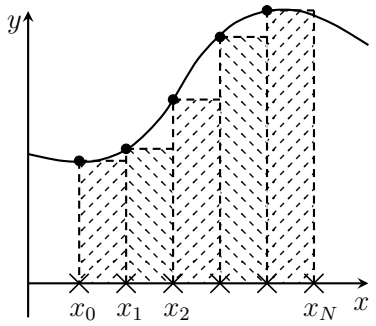


Рис. 3.5: формула левых прямоугольников с узлами, упорядоченными справа налево.

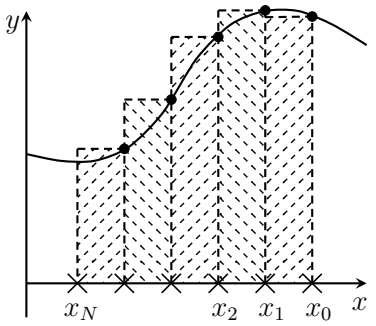


Рис. 3.6: формула левых прямоугольников с узлами, упорядоченными слева направо.

Для формул левых и правых прямоугольников такое утверждение неверно. На n-ом интервале используется значение функции на $n-1$-ом узле (см. Рис. 3.5) Поменяем шаг h на $-h$ (см. Рис. 3.6). Картинка поменялась, значит и ошибка тоже, то есть в разложении ошибки в ряд Тейлора, в общем случае, будут присутствовать степени как четные, так и нечетные.

Подытожим, допустим мы посчитали приближенное значение интеграла по схеме с p-ым порядком точности

$$U = U_N + O(h^p) \tag{3.20}$$

Если у нас имеется априорная, асимптотически точная оценка для главного члена разложения ошибки в ряд Тейлора, то

$$U = U_N + R_N + O(h^{p+q}) \tag{3.21}$$

и для формулы трапеций или средних учет этого члена дает уже 4-ый порядок точности.

Пример численного вычисления интеграла.

С помощью формулы трапеций найти приближенное значение интеграла U, который берется аналитически

$$U = \int_{-1}^{1} \frac{dx}{1+x^2} = \frac{\pi}{2} \simeq 1.57079\ldots \tag{3.22}$$

Вычисления будем проводить на различных сетках, для сравнения результата. Приведём таблицу результатов ниже

На сетке	N	$=$	1	интеграл	U_N	$=$	1.0000
На сетке	N	$=$	2	интеграл	U_N	$=$	1.5000
На сетке	N	$=$	4	интеграл	U_N	$=$	1.5500
На сетке	N	$=$	8	интеграл	U_N	$=$	1.5656
На сетке	N	$=$	16	интеграл	U_N	$=$	1.5695
На сетке	N	$=$	32	интеграл	U_N	$=$	1.5705
На сетке	N	$=$	64	интеграл	U_N	$=$	1.5707
На сетке	N	$=$	128	интеграл	U_N	$=$	1.5707
На сетке	N	$=$	256	интеграл	U_N	$=$	1.5708

Эта маленькая программа позволяет нам убедиться в том, что при более густой сетке точность вычисления растёт и мы приближаемся к более точному значению. Вопросами контроля точности мы займемся уже на следующей лекции.

Лекция 4

Вычисления с контролем точности – одна из ключевых тем курса, и полученные в этой лекции формулы могут быть применены не только при вычислении интегралов, но и, например, при решении систем обыкновенных дифференциальных уравнений и уравнений в частных производных.

Формула Рунге-Ромберга

Рассмотрим формулу, связывающую точное значение интеграла и приближённое.

$$U = U_N + \mathcal{O}(h^p), \tag{4.1}$$

где $U = \int\limits_a^b u(x)dx$ – точное значение интеграла, N – количество интервалов сетки (сетка считается равномерной), тогда шаг сетки вычисляется по формуле $h = \frac{b-a}{N}$.

В данном случае p – порядок точности, то есть ошибка раскладывается в ряд Тейлора по степеням, начиная с p.

В прошлой лекции было получено значение порядка точности p для формулы трапеций при условии, что сетка равномерная. Если сетка будет неравномерная, оценить p не получится.

Запишем формулу (4.1) в следующем виде:

$$U = U_N + c \cdot h^p + \mathcal{O}(h^{p+q}), \tag{4.2}$$

то есть выделим главный член разложения ошибки. Как обсуждалось на прошлой лекции, в случае формулы средних и формулы трапеций ошибка раскладывается только

по чётным степеням h, то есть $q = 2$, а в случае левых и правых прямоугольников – по всем степеням, то есть $q = 1$.

В выражении (4.2) две неизвестных величины – U и c. Чтобы их найти, необходимо записать два уравнения. Проведём вычисления интеграла на сгущённой сетке с количеством интервалов $\tilde{N} = rN$. Тогда шаг сетки уменьшится в r раз по сравнению с исходным:

$$\tilde{h} = \frac{b-a}{\tilde{N}} = \frac{b-a}{rN} = \frac{h}{r}.$$

Получим систему

$$\begin{cases} U = U_N + c \cdot h^p + \mathcal{O}(h^{p+q}) \\ U = U_{rN} + c \cdot \left(\frac{h}{r}\right)^p + \mathcal{O}\left(\left(\frac{h}{r}\right)^{p+q}\right) \end{cases} \tag{4.3}$$

Почему в обоих уравнениях одинаковое c? Если вспомнить предыдущую лекцию, слагаемое ch^p называется асимптотически точной оценкой главного члена разложения в ряд Тейлора. Для формулы трапеций:

$$R_N = -\frac{h^2}{12}(u'(b) - u'(a)),$$

то есть $c = -\frac{1}{12}(u'(b) - u'(a))$, не зависит от числа интервалов и является постоянной, так как при сгущении сетки не меняются подынтегральная функция, её производные, пределы интегрирования.

Из второго уравнения системы (4.3) вычтем первое:

$$0 = U_{rN} - U_N + ch^p \left(\frac{1}{r^p} - 1\right) + \mathcal{O}(h^{p+q})$$

и выразим из этого выражения

$$c = \frac{U_{rN} - U_N}{r^p - 1} \cdot \frac{r^p}{h^p} + \mathcal{O}(h^q).$$

Так как наша цель – уточнить выражение для численного значения интеграла, подставим значение c во второе уравнение системы (4.3):

$$U = U_{rN} + \underbrace{\frac{U_{rN} - U_N}{r^p - 1} + \mathcal{O}(h^{p+q})}_{R_{rN}} + \mathcal{O}(h^{p+q}).$$

В литературе эта формула обычно записывается в виде

$$R_{rN} = \frac{U_{rN} - U_N}{r^p - 1} \tag{4.4}$$

и называется *формулой Рунге-Ромберга*. Обратите внимание, что, на самом деле, формула является асимптотически-точной, так как она правильно передаёт главный член разложения ошибки в ряд Тейлора только при $h \to 0$.

Полученное выражение $U_{rN} + R_{rN}$ приближает абсолютно-точное значение U уже с порядком точности $p + q$, а R_{rN} можно считать абсолютной оценкой ошибки этого выражения.

Методика Ричардсона

Пусть на практике необходимо вычислить интеграл $U = \int\limits_a^b u(x)dx$ с абсолютной точностью ε. Вначале вычислим значение U_N. Затем посчитаем значение на сетке с rN интервалами U_{rN}. U_N и U_{rN} посчитаны с порядком точности p, по ним можно посчитать оценку ошибки R_{rN} и уточненное значение $U^{(1)}_{rN} = U_{rN} + R_{rN}$, порядок точности которого $p + q$. Будем записывать вычисленные значения следующим образом:

$$\begin{array}{cc|c} p & p+q & \\ \hline U_N & & \\ U_{rN} & U^{(1)}_{rN} & R_{rN} \end{array}$$

Если $|R_{rN}| < \varepsilon$, то значение $\tilde{U}_{rN}$ можно воспринимать как приближенное значение U с ошибкой не более ε. Если же это не так, сгущаем сетку ещё в r раз: вычисляем значение U_{r^2N}, по значениям U_{r^2N} и U_{rN} получаем значение R_{r^2N}, вычисляем $U^{(1)}_{r^2N} = U_{r^2N} + R_{r^2N}$. Продолжаем, пока не найдётся такое R_{r^sN}, что $|R_{r^sN}| < \varepsilon$.

$$\begin{array}{ll|l} p & p+q & \\ \hline U_N & & \\ U_{rN} & U^{(1)}_{rN} & R_{rN} \\ U_{r^2N} & U^{(1)}_{r^2N} & R_{r^2N} \\ \vdots & \vdots & \vdots \\ U_{r^sN} & U^{(1)}_{r^sN} & R_{r^sN} \end{array}$$

Эта процедура в западной литературе чаще всего называется процедурой Ричардсона (Richardson extrapolation), то есть однократное применение формулы Рунге-Ромберга. Однако в нашем курсе под *методикой Ричардсона* мы будем понимать рекуррентное применение формулы Рунге-Ромберга.

Дело в том, что значения $U^{(1)}_{rN}$ и $U^{(1)}_{r^2N}$ также имеют одинаковый порядок точности – $p+q$, а значит, по формуле Рунге-Ромберга (4.4) можно вычислить

$$R^{(1)}_{r^2N} = \frac{U^{(1)}_{r^2N} - U^{(1)}_{rN}}{r^{p+q}-1}.$$

Далее можно уточнить значение $U^{(1)}_{r^2N}$:

$$U^{(2)}_{r^2N} = U^{(1)}_{r^2N} + R^{(1)}_{r^2N},$$

полученное значение будет верно с точностью $p+2q$. Рекуррентное применение этих формул, как видно, может дать ответ с любым порядком точности. Например, вместо формулы Симпсона (4 порядок точности) можно использовать формулу трапеций, сделав всего одно сгущение, и получить такой же порядок точности и оценку ошибки сразу.

Формула Рунге-Ромберга даёт *апостериорную* оценку ошибки, т.е. формулы, в отличие от априорных оценок, содержат приближённые значения интегралов, а значит, её можно посчитать только после вычисления соответствующих приближённых значений. С другой стороны, в отличие от априорных оценок, в формуле не присутствуют значения производных.

$$
\begin{array}{ccccc|cccc}
p & p+q & p+2q & \cdots & p+sq & & & & \\
\hline
U_N^{(0)} & & & & & & & & \\
U_{rN}^{(0)} & U_{rN}^{(1)} & & & & R_{rN}^{(0)} & & & \\
U_{r^2N}^{(0)} & U_{r^2N}^{(1)} & U_{r^2N}^{(2)} & & & R_{r^2N}^{(0)} & R_{r^2N}^{(1)} & & \\
\vdots & \vdots & \vdots & \ddots & & \vdots & \vdots & \ddots & \\
U_{r^sN}^{(0)} & U_{r^sN}^{(1)} & U_{r^sN}^{(2)} & \cdots & U_{r^sN}^{(s)} & R_{r^sN}^{(0)} & R_{r^sN}^{(1)} & \cdots & R_{r^sN}^{(s)}
\end{array}
$$

Чтобы написать программу, введём следующие обозначения:

$$
\begin{aligned}
U_{(s,l)} &= U_{r^sN}^{(l)} \\
R_{(s,l)} &= R_{r^sN}^{(l)}
\end{aligned}
$$

Тогда формулы примут вид (в общем случае):

$$
\begin{aligned}
R_{(s,l)} &= \frac{U_{(s,l)} - U_{(s-1,l)}}{r^{p+lq} - 1}, \\
U_{(s,l+1)} &= U_{(s,l)} + R_{(s,l)}
\end{aligned}
\tag{4.5}
$$

при $l \in \overline{0, L-1}$, $s \in \overline{0, S-1}$.

Пример

В качестве примера рассмотрим вычисление интеграла

$$
U = \int_{-1}^{+1} \frac{dx}{1+x^2} = 1.5707963...
$$

(реализация программы доступна в материалах к лекциям под номером 4-1). Возьмём $N = 1$, $S = 3$ и будем использовать формулу трапеций ($p = 2$, $q = 2$).

Таблица приближённых значений интеграла:

	p = 2	p = 4	p = 6
s = 0	1.0000		
s = 1	1.5000	1.6667	
s = 2	1.5500	1.5667	1.5600

Таблица оценок ошибок:

	p = 2	p = 4	p = 6
s = 0			
s = 1	0.1667		
s = 2	0.0167	−0.0067	

На данном примере можно заметить, что при $s = 1$ уточнённое значение получается хуже, чём не уточнённое. Это связано с асимптотической точностью формул, то есть формулы верны только при достаточно больших N (или, что то же самое, при достаточно малых h). Определению того, достаточно ли большое N, посвящён следующий раздел.

Эффективный порядок точности

Вернёмся к выводу формулы Рунге-Ромберга.

$$U = U_{rN} + \underbrace{c \cdot \left(\frac{h}{r}\right)^p}_{R_{rN}} + \mathcal{O}\left(h^{p+q}\right),$$

проведём ещё одно сгущение

$$U = U_{r^2N} + \underbrace{c \cdot \left(\frac{h}{r^2}\right)^p}_{R_{r^2N}} + \mathcal{O}\left(h^{p+q}\right)$$

и поделим одно выражение для ошибки на другое:

$$\frac{R_{rN}}{R_{r^2N}} = r^p.$$

Отсюда

$$p = \log_r \frac{R_{rN}}{R_{r^2N}}.$$

Это равенство верно только в том случае, если здесь стоят точные выражения для главного члена разложения ошибки в ряд Тейлора. Мы же будем подставлять приближенные

значения, а значит, равенство также будет носить асимптотический характер. Поэтому введём понятие *эффективного порядка точности*

$$p^{\mathit{eff}}{}_{r^2N} = \log_r \left| \frac{R_{rN}}{R_{r^2N}} \right|. \tag{4.6}$$

Эффективный порядок точности p^{eff} может отличаться от заданного порядка точности p, но $p^{\mathit{eff}}{}_{r^sN} \to p$ при $s \to \infty$. Наличие модуля связано с тем, что оценки ошибок могут быть как положительные, так и отрицательные.

Критерий достаточной малости h – близость эффективного порядка точности к теоретическому, так как значения оценок ошибок будут посчитаны уже достаточно точно.

Обозначим

$$p_{(s,l)} = \left(p^{\mathit{eff}}\right)^{(l)}_{r^sN} = \log_r \frac{R_{(s-1,l)}}{R_{(s,l)}}. \tag{4.7}$$

Программу с вычислением эффективного порядка точности можно найти в материалах к лекции под номером 4-2.

Пример

Посчитаем значения эффективного порядка точности для интеграла

$$U = \int_{-1}^{+1} \frac{dx}{1+x^2} = 1.5707963...$$

Таблица приближённых значений интеграла:

	p = 2	p = 4	p = 6	p = 8	p = 10
s = 0	1.0000				
s = 1	1.5000	1.6667			
s = 2	1.5500	1.5667	1.5600		
s = 3	1.5656	1.5708	1.5711	1.5712	
s = 4	1.5659	1.5708	1.5708	1.5708	1.5708

Таблица оценок ошибок:

	p = 2	p = 4	p = 6	p = 8	p = 10
s = 0					
s = 1	0.1667				
s = 2	0.0167	−0.0067			
s = 3	0.0052	0.0003	0.0002		
s = 4	0.0013	0.0000	−0.0000	−0.0000	

Таблица эффективных порядков точности:

	p = 2	p = 4	p = 6	p = 8	p = 10
s = 0					
s = 1					
s = 2	3.3219				
s = 3	1.6815	4.6020			
s = 4	1.9967	8.4302	5.4007		

Видно, что $p_{(4,0)} \approx p = 2$, а значит, значению $U_{(4,1)}$ можно доверять.

У эффективного порядка точности есть также геометрическая интерпретация. Если построить в двойном логарифмическом масштабе график ошибки $|R^{(0)}|$ как функцию N, то тангенс угла наклона должен быть равен $-p$ (см. рис. 4.1).

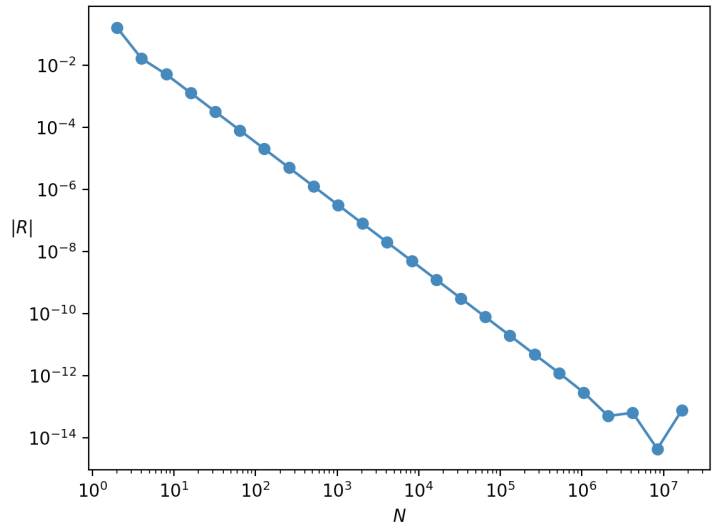


Рис. 4.1: График зависимости модуля ошибки $|R^{(0)}|$ от количества интервалов сетки N.

На графике видно, что при малых N график отклоняется от прямой – это происходит из-за недостаточной точности. При больших же N начинают сказываться ошибки округления. Сбой происходит примерно при $N \approx 10^6$, при котором $|R| \approx 10^{-13}$, дальнейшее сгущение сетки не имеет смысла.

Аналогичный график можно построить для $|R^{(1)}|$ (рис. 4.2).

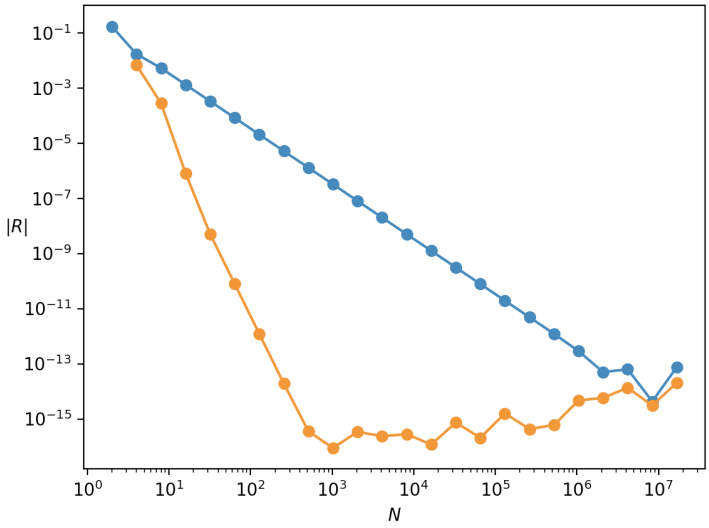


Рис. 4.2: График $|R^{(0)}|$, $|R^{(1)}|$ (синий и оранжевый соответственно).

Здесь видно, что не имеет смысла брать N больше, чем 1000.

Пример

Рассмотрим другой интеграл:

$$U = \int_0^1 x^2 dx = 0.3333...$$

Таблица приближённых значений интеграла:

	$p = 2$	$p = 4$	$p = 6$	$p = 8$	$p = 10$
$s = 0$	0.5000				
$s = 1$	0.3750	0.3333			
$s = 2$	0.3438	0.3333	0.3333		
$s = 3$	0.3359	0.3333	0.3333	0.3333	
$s = 4$	0.3340	0.3333	0.3333	0.3333	0.3333

Таблица оценок ошибок:

	$p = 2$	$p = 4$	$p = 6$	$p = 8$	$p = 10$
$s = 0$					
$s = 1$	-0.0417				
$s = 2$	-0.0104	0.0000			
$s = 3$	-0.0026	0.0000	0.0000		
$s = 4$	-0.0007	0.0000	0.0000	0.0000	

Таблица эффективных порядков точности:

	$p = 2$	$p = 4$	$p = 6$	$p = -8$	$p = 10$
$s = 0$					
$s = 1$					
$s = 2$	2.0000				
$s = 3$	2.0000	nan			
$s = 4$	2.0000	nan	nan		

nan-ы получаются из-за того, что интегрирование полинома второй степени по формулам с порядком точности выше 2-го даёт абсолютно точный результат. Так как все последующие члены разложения в ряд Тейлора содержат нули, формула передаёт их все, а значит, эффективный порядок точности равен ∞.

Аналогично можно убедиться, что для $U = \int\limits_0^1 \sqrt{x}dx$ эффективный порядок точности будет сходиться примерно к 1.5. Это связано с тем, что производная корня не непрерывна на отрезке интегрирования $[0, 1]$, а значит, в ряд Тейлора её раскладывать нельзя.

Методика Эйткена

Предполагаем, что существуют какие-то проблемы с непрерывностью производных интегрируемой функции, а значит, раскладывать функцию в ряд Тейлора мы не можем. Обозначим всю ошибку за ch^p:

$$U = U_N + c \cdot h^p.$$

В данном выражении три неизвестных: U, c и p. Запишем аналогичные выражения при сгущении сетки:

$$\begin{cases} U = U_N + c \cdot h^p \\ U = U_{rN} + c \cdot \left(\frac{h}{r}\right)^p \\ U = U_{r^2N} + c \cdot \left(\frac{h}{r^2}\right)^p \end{cases}$$

Из этой системы можно выразить

$$U = U_{r^2N} + \frac{U_{r^2N} - U_{rN}}{\frac{U_{rN} - U_N}{U_{r^2N} - U_{rN}} - 1} \tag{4.8}$$

– *формула Эйткена*. При выводе формулы мы использовали, что все значения c – одинаковые. Это – лишь допущение, гарантировать его мы не можем.

Также нужно обратить внимание на то, что применять эту формулу рекуррентно нельзя, так как, в отличие от формулы Рунге-Ромберга, она описывает всю ошибку, а не только главный член разложения.

Контрпример к методике Ричардсона

Рассмотрим интеграл $\int\limits_0^{8\pi} |\sin x| dx \neq 0$ (график подынтегральной функции изображён на рис. 4.3).

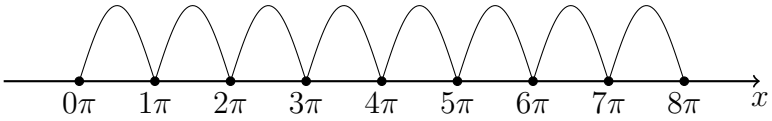


Рис. 4.3: График функции $u(x) = |\sin x|$.

Если начинать с очень малого количества интервалов, то значения U_N получатся равными 0 (так как узлы сетки попадут в точки, в которых функция зануляется), аналогично произойдёт и с оценками ошибок и может сложиться впечатление, что $p^{eff} = \infty$, т.е. передаются все члены разложения в ряд Тейлора. На самом деле это, очевидно, не так.

Тут нужно помнить, что к каждому численному методу есть свой контрпример, но при решении реальных практических задач такие случаи вряд ли встретятся.

Лекция 5

Тема этой лекции – вычисление интегралов на квазиравномерных сетках. По названию понятно, что, с одной стороны, такие сетки равномерными не являются, а с другой – что они обладают некоторыми свойствами равномерных сеток.

На предыдущих лекциях рассматривались только собственные интегралы. Рассмотрим несобственный интеграл первого рода:

$$U = \int_a^{\infty} u(x)dx. \tag{5.1}$$

На практике нередко заменяют бесконечность конечным отрезком, на котором вводят равномерную сетку:

$$U^{(A)} = \int_a^{A} u(x)dx.$$

Но этот подход является довольно грубым и необоснованным: например, исходный интеграл может расходиться. В случае, если исходный интеграл исследовался на предмет сходимости теоретически, область интегрирования, действительно, можно заменить конечным отрезком $[a, A]$, но тогда возникают проблемы выбора правого конца этого отрезка A, а при его увеличении довольно быстро растёт количество интервалов сетки. Ещё одним недостатком является невозможность применить формулу Рунге-Ромберга и методику Ричардсона, о которых говорилось в предыдущей лекции. Действительно, при применении этих формул мы оценим только интеграл на конечном отрезке, но не исходный.

Мы пойдём другим путём и будем вводить квазиравномерные сетки, которые смогут покрыть весь луч, а не конечный отрезок, при увеличении количества интервалов сетки

N. Квазиравномерные сетки также позволяют оценить порядок точности, а значит, на них можно будет осуществлять сгущение сеток.

Квазиравномерные сетки

Введём переменную $\xi \in [\alpha, \beta]$ и построим преобразование $x(\xi)$, которое каждому узлу равномерной сетки по ξ ставит в соответствие какое-то значение x.

Введём равномерную сетку по ξ:

$$\xi_n = \alpha + \underbrace{\frac{\beta - \alpha}{N}}_{\Delta} n, \quad n \in \overline{0, N}. \tag{5.2}$$

Тогда преобразование $x(\xi)$ будет задавать набор сеточных значений $x_n = x(\xi_n)$.

Потребуем:

1. $x(\xi)$ – достаточно гладкая;
2. $x'(\xi) \geq q > 0$;
3. $x(\alpha) = a, \quad x(\beta) = b$.

Тогда $\{x_n\}_{n \in \overline{0,N}} \subset [a, b]$ – *квазиравномерная сетка*.

Шаг квазиравномерной сетки вычисляется по формуле

$$h_n = x_n - x_{n-1} = x(\xi_n) - x(\xi_{n-1}) = \text{// по формуле конечных приращений Лагранжа //} =$$

$$= x'(\xi^*)(\xi_n - \xi_{n-1}) \approx x'(\xi_{n-1/2})\Delta = x'_{n-1/2}\Delta = \mathcal{O}(N^{-1}) \to 0 \text{ при } N \to \infty.$$

Отношение шагов на разных интервалах

$$\frac{h_m}{h_n} \approx \frac{x'_{m-1/2}}{x'_{n-1/2}}$$

– различные интервалы могут отличаться сколь угодно сильно за счёт выбора $x(\xi)$, а вот соседние интервалы будут близки друг к другу (при $N \to \infty$):

$$\frac{h_{n+1}}{h_n} \approx \frac{x'_{n+1/2}}{x'_{n-1/2}} = \frac{x'(\xi_{n+1/2})}{x'(\xi_{n-1/2})} \approx \frac{x'(\xi_{n-1/2}) + x''(\xi_{n-1/2})\Delta}{x'(\xi_{n-1/2})} = 1 + \frac{x''(\xi_{n-1/2})}{x'(\xi_{n-1/2})}\Delta = 1 + \mathcal{O}(N^{-1}).$$

Примеры квазиравномерных сеток

Пример

Рассмотрим следующее преобразование:

$$x(\xi) = a + (b-a)\frac{e^{c\xi}-1}{e^c-1}, \tag{5.3}$$

которое переводит отрезок $\xi \in [0,1]$ в отрезок $x \in [a,b]$.

c – управляющий параметр, который определяет густоту соответствующей квазиравномерной сетки. На графиках изображены случаи $c > 0$ и $c < 0$ (см. рис. 5.1).

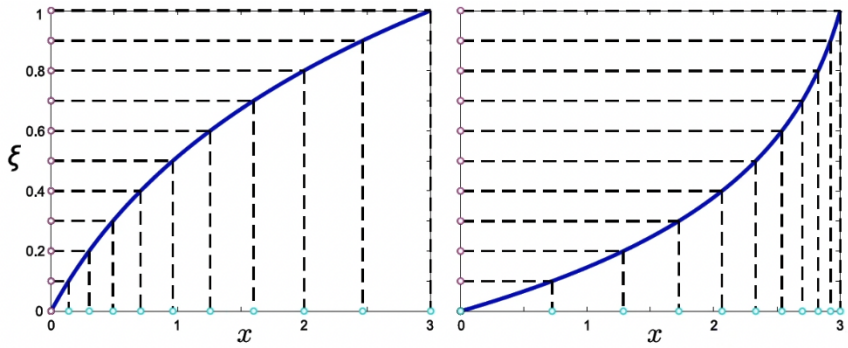


Рис. 5.1: Примеры квазиравномерный сеток.

С помощью c можно регулировать, насколько подробно учитываются те или иные особенности функции.

Пример

Рассмотрим другое преобразование (см. рис. 5.2):

$$x(\xi) = a + \frac{c\xi}{(1-\xi)^m}, \tag{5.4}$$

которое переводит отрезок $\xi \in [0,1]$ в луч $x \in [a,\infty)$. Действительно,

$$x_N = a + \frac{c\xi_N}{(1-\xi_N)^m} = a + \frac{c \cdot 1}{(1-1)^m} = \infty.$$

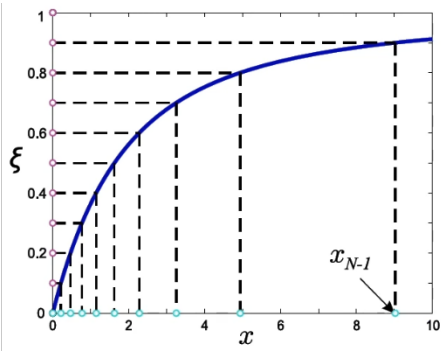


Рис. 5.2: Пример квазиравномерной сетки.

Пример

Покажем, что преобразование

$$x(\xi) = c\xi^2$$

не порождает квазиравномерную сетку на отрезке $x \in [0, c]$ при $\xi \in [0, 1]$.

Вычислим значения шагов интервалов:

$$\xi_n = \frac{1}{N}n,$$

$$h_1 = x_1 - x_0 = x(\xi_1) - x(\xi_0) = \frac{1}{N^2} - 0 = \frac{1}{N^2},$$

$$h_2 = x_2 - x_1 = \frac{2^2}{N^2} - \frac{1}{N^2} = \frac{3}{N^2},$$

$$\frac{h_2}{h_1} = 3 \nrightarrow 1 \text{ при } N \to \infty,$$

нарушается условие на близость соседних шагов сетки. Это происходит потому, что производная $x'(\xi) = 2c\xi$ обращается в ноль при $\xi = 0$.

Вычисление интегралов на квазиравномерных сетках

Запишем формулу средних для вычисления интеграла:

$$U_N = \sum_{n=1}^{N} u(x_{n-1/2})h_n, \tag{5.5}$$

Ошибка имеет вид

$$R_N = \sum_{n=1}^{N} \left(\frac{h_n^3}{24} u''_{n-1/2} \right) \approx \frac{1}{24} \sum_{n=1}^{N} \left(x'_{n-1/2} \Delta \right)^2 u''_{n-1/2} h_n = \frac{\Delta^2}{24} \sum_{n=1}^{N} \left(x'_{n-1/2} \right)^2 u''_{n-1/2} h_n \approx$$

// сумму приближённо заменяем интегралом, так как N достаточно большое //

$$\approx \frac{\Delta^2}{24} \int_a^b (x')^2 u'' dx = \frac{1}{N^2} \frac{(\beta - \alpha)^2}{24} \int_a^b (x')^2 u'' dx = \mathcal{O}(N^{-2}).$$

Получаем, что на квазиравномерной сетке для формулы средних порядок точности остаётся $p = 2$, а значит, можно проводить вычисления с контролем точности.

Пусть нам необходимо вычислить несобственный интеграл на $[a, \infty)$:

$$U = \int_a^{\infty} u(x) dx,$$

$$U_N = \sum_{n=1}^{N} u(x(\xi_{n-1/2})) h_n.$$

В N-ом слагаемом сталкиваемся с проблемой:

$$h_N = x_N - x_{N-1}, \text{ при } x_N = \infty.$$

Чтобы справиться с этой неприятностью, перепишем формулу в другом виде:

$$U_N = \sum_{n=1}^{N} u(x(\xi_{n-1/2})) x'(\xi_{n-1/2}) \Delta. \tag{5.6}$$

По сути, мы просто сделали замену переменных:

$$U = \int_a^{\infty} u(x) dx = \int_{\alpha}^{\beta} u(x(\xi)) x'(\xi) d\xi.$$

В формуле трапеций встречается слагаемое $u(x_N)$, которое может быть невозможно найти, поэтому мы эту формулу использовать не будем.

$$U_N = \sum_{n=1}^{N} \frac{u(x_n) + u(x_{n-1})}{2} h_n$$

По аналогичной причине формула правых прямоугольников тоже неприменима.

Примеры вычисления несобственных интегралов

Программа доступна в материалах к лекциям под номером 5-1.

Пример

Рассмотрим интеграл

$$U = \int_0^\infty e^{-x} dx = 1.$$

Вычисления проводим на сетке, порождённой преобразованием

$$x(\xi) = \frac{\xi}{(1-\xi)},$$

вычисления проводим, начиная с количества интервалов $N = 1$.

Таблица приближённых значений интеграла:

	$\mathrm{p} = 2$	$\mathrm{p} = 4$	$\mathrm{p} = 6$	$\mathrm{p} = 8$	$\mathrm{p} = 10$
$\mathrm{s} = 0$	1.4715				
$\mathrm{s} = 1$	1.0352	0.8898			
$\mathrm{s} = 2$	0.9847	0.9678	0.9730		
$\mathrm{s} = 3$	1.0018	1.0075	1.0101	1.0107	
$\mathrm{s} = 4$	1.0002	0.9996	0.9991	0.9989	0.9989

Таблица оценок ошибок:

	p = 2	p = 4	p = 6	p = 8	p = 10
s = 0					
s = 1	−0.1454				
s = 2	−0.0168	0.0052			
s = 3	0.0057	0.0026	0.0006		
s = 4	−0.0005	−0.0005	−0.0002	−0.0000	

Таблица эффективных порядков точности:

	p = 2	p = 4	p = 6	p = 8	p = 10
s = 0					
s = 1					
s = 2	3.1098				
s = 3	1.5623	0.9764			
s = 4	3.3930	2.3322	1.7481		

Видно, что на сетке с $N = 16$ интервалами ($s = 4$) значение уже довольно близко к аналитическому, но эффективные порядки точности к аналитическому ещё не сходятся (если бы мы не знали точное значение, этим результатам доверять бы было нельзя).

Как и в лекции 4, построим большее количество значений на графике $|R^{(0)}|$ от N (рис. 5.3).

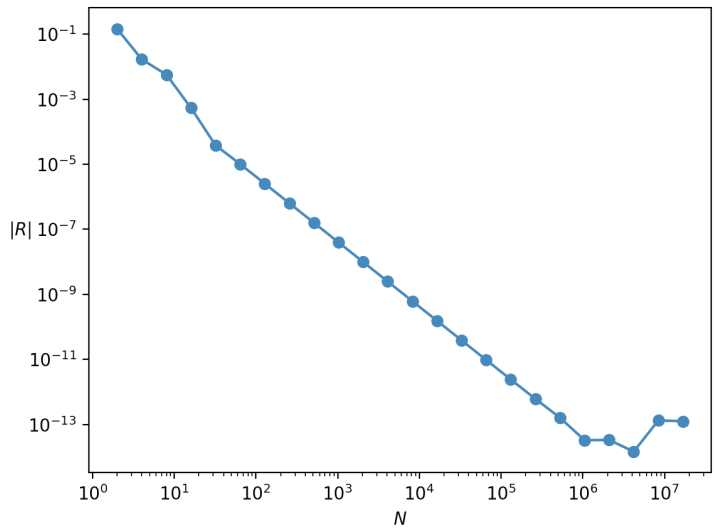


Рис. 5.3: График зависимости модуля ошибки $|R^{(0)}|$ от количества интервалов сетки N.

Видно, что значениям можно доверять, начиная, примерно, с $N = 50$ и до $N = 10^6$, где начинают влиять ошибки округления.

Можно также построить график для $|R^{(1)}|$ и последующих оценок (рис. 5.4).

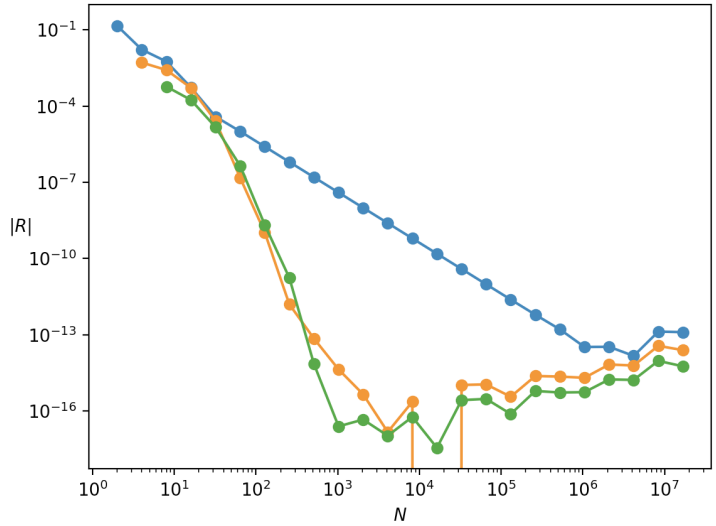


Рис. 5.4: График $|R^{(0)}|$, $|R^{(1)}|$, $|R^{(2)}|$ (синий, оранжевый и зелёный соответственно).

Пример

Рассмотрим теперь расходящийся интеграл:

$$U = \int_0^\infty x dx = +\infty.$$

Таблица приближённых значений интеграла:

	p = 2	p = 4	p = 6	p = 8	p = 10
s = 0	4.				
s = 1	24.	31.			
s = 2	115.	145.	153.		
s = 3	499.	627.	659.	667.	
s = 4	2075.	2600.	2732.	2765.	2773.

Таблица оценок ошибок:

	$\mathrm{p}=2$	$\mathrm{p}=4$	$\mathrm{p}=6$	$\mathrm{p}=8$	$\mathrm{p}=10$
$\mathrm{s}=0$					
$\mathrm{s}=1$	6.7				
$\mathrm{s}=2$	30.3	7.6			
$\mathrm{s}=3$	128.0	32.1	8.0		
$\mathrm{s}=4$	525.3	131.5	32.9	8.2	

Таблица эффективных порядков точности:

	$\mathrm{p}=2$	$\mathrm{p}=4$	$\mathrm{p}=6$	$\mathrm{p}=8$	$\mathrm{p}=10$
$\mathrm{s}=0$					
$\mathrm{s}=1$					
$\mathrm{s}=2$	-2.1662				
$\mathrm{s}=3$	-2.0762	-2.0707			
$\mathrm{s}=4$	-2.0366	-2.0341	-2.0335		

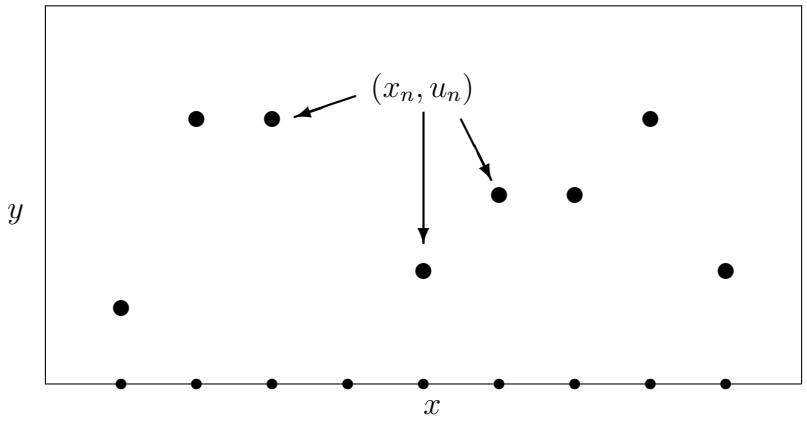
Видно, что значения U_N растут с каждым шагом, а эффективный порядок точности $p^{\mathit{eff}} \to -2$. В данном случае реально передаётся $-2 < 0$ члена разложения в ряд Тейлора, то есть решение в ряд Тейлора не раскладывается, а значит, не существует.

Получается, методика Ричардсона помогает определить, что интеграл расходится.

Лекция 6

Интерполяция функций.

Пусть нам задана функцией не аналитически, а в виде набора сеточных значений, то есть имеются узлы сетки по x, и в этих узлах задана функция $y(x_n)$, поэтому, фактически, у нас есть таблица чисел с координатами (x_n, u_n) для $n = \overline{0, N}$. **Задача интерполяции**:



найти приближенное значение функции в точке, не совпадающей с узлом сетки. Такие задачи часто встречаются на практике, например, когда функция $y(x)$ определена в эксперименте. Для решения задачи интерполяции введем базисные функции $\varphi_k(x)$, $k = \overline{0, N}$, и построим обобщенный многочлен:

$$\Phi_N(x) = \sum_{k=0}^{N} c_k \varphi_k(x).$$

Коэффициенты c_k будем определять следующим образом:

$$\Phi_N(x_n) = u(x_n), \quad n = \overline{0, N}.$$

В результате мы получаем следующую систему:

$$\sum_{k=0}^{N} c_k \varphi_k(x_n) = u(x_n), \quad n = \overline{0, N}.$$

Таким образом, мы получаем систему из $N+1$ уравнения с $N+1$ неизвестными c_k. Перепишем эту систему в матричном виде:

$$\underbrace{\begin{pmatrix} \varphi_0(x_0) & \varphi_1(x_0) & \cdots & \varphi_N(x_0) \\ \varphi_0(x_1) & \varphi_1(x_1) & \cdots & \varphi_N(x_1) \\ \vdots & \vdots & \ddots & \vdots \\ \varphi_0(x_N) & \varphi_1(x_N) & \cdots & \varphi_N(x_N) \end{pmatrix}}_{A} \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_N \end{pmatrix} = \begin{pmatrix} u_0 \\ u_1 \\ \vdots \\ u_N \end{pmatrix}$$

Эта система имеет решение только в том случае, если $\det A \neq 0$. В зависимости от выбора базисных функций и сетки, определитель может быть как отличен от нуля, так и равен нулю. Поэтому базисные функции $\varphi_k(x)$ выбираются такими, чтобы определитель на данной сетке был отличен от нуля. Один из простейших вариантов следующий:

$$\varphi_k(x) = x^k, \quad k = \overline{0, N}.$$

В этом случае определитель системы примет следующий вид:

$$\det A = \begin{vmatrix} 1 & x_0 & \cdots & x_0^N \\ 1 & x_1 & \cdots & x_1^N \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_N & \cdots & x_N^N \end{vmatrix}$$

Этот определитель называется определителем Вандермонда, и из курса линейной алгебры известно, что он отличен от нуля при условии, что $x_n \neq x_m \; \forall n, m$. Поэтому в произвольном случае для простоты в качестве базисных функций выбирается система степеней. После нахождения коэффициентов c_k можем найти значение функции в произвольной точке x^*:

$$u(x^*) \cong \sum_{k=0}^{N} c_k (x^*)^k.$$

При этом из построения обобщенного многочлена понятно, что если x^* совпадает с одним из узлов сетки, то равенство будет точное. Если же x^* отличается от узлов, то соответствующее значение будет всего лишь приближенным. Интуитивно понятно, что приближение степенями достаточно разумно, так как чем больше у нас точек, тем более высокой степенью полинома приближается функция. А это аналог разложения функции в ряд Тейлора: чем больше членов разложения учитываем, то есть, фактически, приближаем функцию полиномом более высокой степени, тем точнее передаем.

Самая вычислительно-емкая операция в данном алгоритме это нахождение набора коэффициентов c_k из системы уравнений. Так как система имеет произвольный вид, то мы применяем метод Гаусса, для которого, как было показано на второй лекции, количество операций $\sim \frac{2}{3}(N+1)^3 = O(N^3)$.

Интерполяционный многочлен Ньютона.

Запишем следующую таблицу и разделенные разности функции u в узле n порядка k:

$$\begin{array}{cc} x_0 & u_{0,0} \\ x_1 & u_{1,0} \\ x_2 & u_{2,0} \\ x_3 & u_{3,0} \\ \vdots & \vdots \\ x_N & u_{N,0} \end{array} \qquad u_{n,k} = k\frac{u_{n,k-1} - u_{n+1,k-1}}{x_n - x_{n+k}}, \text{ при этом } \quad u_{n,0} \equiv u_n, n = \overline{0, N}.$$

Например, разделенные разности в нулевом узле первого и второго порядка выглядят следующим образом:

$$u_{0,1} = 1 \cdot \frac{u_{0,0} - u_{1,0}}{x_0 - x_1}, \quad u_{0,2} = 2 \cdot \frac{u_{0,1} - u_{1,1}}{x_0 - x_2}$$

Данные конструкции похожи на первую и вторую производную, соответственно. Заметим, что про их построении не накладывается условие того, что узлы должны быть упорядочены. Теперь запишем произвольный полином $P(x)$ степени N, который проходит через набор точек (x_n, u_n), $n = \overline{0, N}$, и посмотрим на его разделенные разности:

$$\begin{array}{cc} x & P(x) \equiv P_{x,0} \\ x_0 & P(x_0) \equiv P_{0,0} \\ x_1 & P(x_1) \equiv P_{1,0} \\ x_2 & P(x_2) \equiv P_{2,0} \\ x_3 & P(x_3) \equiv P_{3,0} \\ \vdots & \vdots \\ x_N & P(x_N) \equiv P_{N,0} \end{array}$$

$$\begin{aligned} &P_{x,1} = 1 \cdot \frac{P_{x,0} - P_{0,0}}{x - x_0} = \frac{P(x) - P(x_0)}{x - x_0} \\ &P_{x,2} = 2 \cdot \frac{P_{x,1} - P_{0,1}}{x - x_1} = \frac{2}{x - x_1}\Big(\frac{P(x) - P(x_0)}{x - x_0} - \frac{P(x_0) - P(x_1)}{x_0 - x_1}\Big) \\ &\vdots \\ &P_{x,N} = const \\ &P_{x,N+k} = 0, \quad k \geq 1 \end{aligned}$$

Заметим, что если подставить точку x_0 в $P_{x,1}$, то числитель будет равен нулю, то есть в выражении $P_{x,0} - P_{0,0}$ можно вынести за скобки $(x - x_0)$, поэтому $P_{x,1}$ является полиномом степени $N - 1$. Аналогичная ситуация происходит с разделенными разностями более высоких порядков. Теперь выразим $P(x)$ из выражения для $P_{x,1}$:

$$P(x) = P(x_0) + P_{x,1}(x - x_0),$$

где неизвестное $P_{x,1}$ выразим из выражения для $P_{x,2}$. В результате получаем

$$P(x) = P_{0,0} + (x - x_0)(P_{0,1} + \frac{x - x_1}{2}P_{x,2}).$$

Проделывая все следующие подстановки, в итоге получаем выражение

$$P(x) = \sum_{n=0}^{N} \frac{P_{0,n}}{n!} \prod_{k=0}^{n-1}(x - x_k) = \sum_{n=0}^{N} \frac{u_{0,n}}{n!} \prod_{k=0}^{n-1}(x - x_k) \cong u(x),$$

так как $u_n = P(x_n) = P_{n,0}$.

Таким образом, алгоритм построения интерполяционного многочлена Ньютона:

1. Известны узлы сетки и сеточные значения неизвестной функции в узлах этой сетки (x_n, u_n), $n = \overline{0, N}$
2. Определяем разделенные разности в каждом из узлов как $u_{n,0} = u_n, n = \overline{0, N}$
3. Вычисляем разделенные разности $u_{n,k} = k\dfrac{u_{n,k-1} - u_{n+1,k-1}}{x_n - x_{n+k}}, k = \overline{1, N}, n = \overline{0, N-k}$
4. Находим значение функции в произвольной точке $u(x) = \sum\limits_{n=0}^{N} \frac{u_{0,n}}{n!} \prod\limits_{k=0}^{n-1}(x - x_k)$

Заметим, что последняя формула похожа на разложение в ряд Тейлора. Прикинем количество операций для этого алгоритма. Для вычисления разделенных разностей используется двойной цикл, в котором выполняется четыре операции: два вычитания, деление и умножение. Итого получаем

$$\sum_{k=1}^{N}\sum_{n=0}^{N-k} 4 = \sum_{k=1}^{N} 4 \cdot (N - k + 1) = \sum_{k=1}^{N}(4N - 4) - 4\sum_{k=1}^{N} k = (4N - 4) \cdot N - 4\frac{1+N}{2} \cdot N =$$

$$= 4N^2 - 4N - 2N - 2N^2 = 2N^2 - 6N$$

Подсчитаем количество операций, необходимое для вычисления $u(x)$:

$$\sum_{n=0}^{N}\Big[(1{+}1){+}\sum_{k=0}^{n-1} 2\Big] = \sum_{n=0}^{N}\Big[2{+}2{\cdot}n\Big] = 2{\cdot}(N{+}1){+}2{\cdot}\frac{0+N}{2}(N{+}1) = 2N{+}2{+}N^2{+}N = N^2{+}3N{+}2$$

Итого, количество операций: $O(N^2)$.

Априорная погрешность интерполяционного многочлена Ньютона.

Для выполнения оценок погрешности $u(x) - P(x)$ интерполяционного многочлена Ньютона будем предполагать, что у функции $u(x)$ существуют все производные до $N + 1$ порядка включительно, и известно, что производная порядка $N{+}1$ ограничена: $|u^{(N+1)}(x)| \leq M_{N+1}$. Будем искать погрешность в виде произведения двух функций:

$$u(x) - P(x) = q(x)\omega_{N+1}(x), \quad \omega_{N+1}(x) = \prod_{k=0}^{N}(x - x_k),$$

так как в узлах сетки ошибка должна равняться нулю. Теперь оценим $q(x)$, для этого введем вспомогательную функцию $r(x)$:

$$r(x) = u(x) - P(x) - q(\xi)\omega_{N+1}(x), \quad \xi \in [a,b], a = \min(x_0, x_1, \ldots, x_N, x), b = \max(x_0, x_1, \ldots, x_N, x).$$

Заметим, что функция $r(x)$ дифференцируема $N+1$ раз, так как $q(\xi)$ – это число. $r(x)$ на отрезке $[a,b]$ имеет $N+2$ нуля: $N+1$ узлов и точка ξ. Из курса математического анализа известно, что между парой нулей функции находится нуль ее производной, поэтому функция $r'(x)$ имеет $N+1$ нуль, и, аналогично, $r''(x)$ имеет N нулей. В итоге, $r^{(N+1)}(x)$ имеет ровно один один нуль, который обозначим как $r^{(N+1)}(x^*) = 0$. Запишем производную $N+1$ порядка функции $r(x)$ в точке x^*:

$$r^{(N+1)}(x^*) = u^{(N+1)}(x^*) - q(\xi)(N+1)! = 0 \quad \Rightarrow$$

$$\Rightarrow q(\xi) = \frac{u^{(N+1)}(x^*)}{(N+1)!} \quad \text{и} \quad |q(x)| \leq \frac{M_{N+1}}{(N+1)!}.$$

В результате получаем формулу для априорной погрешности вычисления приближенного значения неизвестной функции $u(x)$:

$$|u(x) - P(x)| \leq \frac{M_{N+1}}{(N+1)!} \prod_{k=0}^{N}(x - x_k).$$

Эта формула называется априорной, так как ей можно воспользоваться до того, как вычислено приближенное значение функции. Однако на практике оценка для производной M_{N+1} неизвестна. Но с помощью этой формулы можно оценить порядок точности на равномерной сетке(ЗДЕСЬ КАРТИНКА РАВНОМЕРНОЙ СЕТКИ, ТАЙМИНГ 1:11:30):

$$|u(x) - P(x)| \leq \frac{M_{N+1}}{(N+1)!} \prod_{k=0}^{N}(x - x_k) \leq \frac{M_{N+1}}{(N+1)!}(Nh)^{N+1} = O(h^{N+1}).$$

Поэтому на равномерной сетке порядок точности будет $N+1$. Может сложиться впечатление, что имеет смысл использовать как можно больше узлов N для повышения точности интерполяции. Однако правильный ответ заключается в том, что нужно использовать, наоборот, меньше узлов, и точность повышается, если эти узлы лежат достаточно близко друг к другу. Если число узлов большое, то у нас возрастает коэффициент $\frac{N^{N+1}}{(N+1)!}$, причем он может расти чрезвычайно сильно, и ошибка может стать неприемлемо большой. **Практические наблюдения**: при интерполяции произвольного набора данных не имеет смысла брать больше 7-8 узлов, но всё зависит от задачи. Если x берем на отрезке, который определяется набором $(x_0, x_1, \ldots, x_N)$, то это задача интерполяции. А если x брать вне этого отрезка, то это задача экстраполяции, и она уже поставлена **некорректно**, так как с увеличением узлов и с удалением от отрезка ошибка становится колоссальной, то есть проявляется неустойчивость – малое изменение входных данных можно привести к большой ошибке.

Апостериорная погрешность интерполяционного многочлена Ньютона.

Выпишем еще раз интерполяционный многочлен Ньютона:

$$u(x) = \sum_{n=0}^{N} \frac{u_{0,n}}{n!} \prod_{k=0}^{n-1} (x - x_k).$$

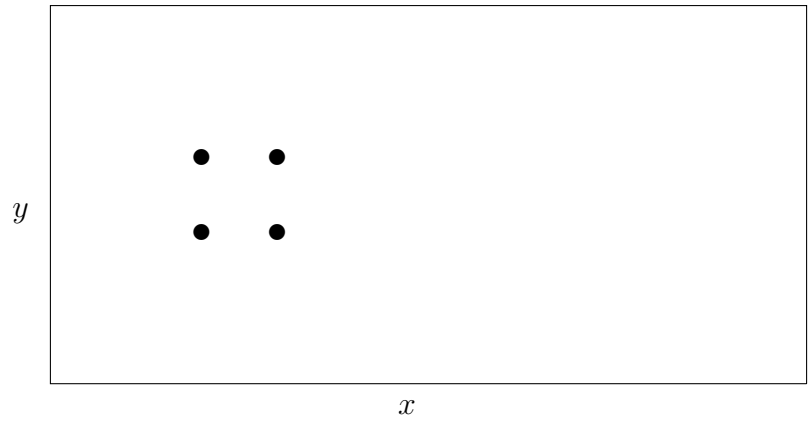
Допустим, дано много точек, например, сто, и необходимо найти приближенное значение в каком-то x, не совпадающим ни с одним узлом сетки. Сто точек для интерполяции использовать бессмысленно, так как ошибки будут огромными, поэтому нужно ограничиться только какой-то их частью. Из вида априорной погрешности можно сделать вывод, что лучше выбирать ближайшие к x точки. Возникает вопрос: сколько точек нужно выбрать? Распишем несколько слагаемых:

$$u(x) = u_{0,0} + u_{0,1}(x - x_0) + \frac{u_{0,2}}{2}(x - x_0)(x - x_1) + \cdots$$

В качестве точки (x_0, u_0) выбираем ту точку, координата которой ближе всего к координате x, тогда $u_{0,0} = u_0$. Затем возьмем еще одну точку (x_1, u_1), координата которой как можно ближе к x, $u_{1,0} = u_1$. По этим двум точкам можем посчитать разделенную разность $u_{0,1}$. Теперь очень грубо можно считать, что $u_{0,1}(x - x_0)$ – главный член разложения ошибки в ряд Тейлора, если все x_k близки. Таким образом точки добавляем до тех пор, пока не вышли либо на заданную точность, либо на наилучшую. Так как интерполяционный многочлен Ньютона всего лишь напоминает формулу Тейлора, поэтому с каждой прибавленной точкой $\prod_{k=0}^{n-1}(x - x_k)$ будет всё меньше похоже на $(x - x_0)^n$, и значит, отклонение от формулы Тейлора будет всё больше и больше, и в какой-то момент очередная добавка будет не меньше, чем предыдущая, а вдруг возрастет. Как только эта добавка возросла по сравнению с предыдущей, значит, сходство с формулой Тейлора нарушилось, и продолжать добавлять точки не имеет смысла. Это и будет наилучшая точность. Такой алгоритм позволяет делать апостериорные оценки точности, то есть, после вычислений. Итак, в цикле последовательно выбираются точки, ближайшие к точке x, и соответствующие слагаемые добавляются до тех пор, пока не выйдем на заданную, либо наилучшую точность. Так как узлы не обязательно должны упорядоченными, можно брать точки то слева, то справа от x.

Параметрическая интерполяция кривых.

Допустим, дано четыре точки. Интерполяционный многочлен Ньютона через них нельзя провести, так как он является функцией, то есть одному значению x ставится в соответствие одно значение функции. В этом случае ставится задача об интерполяции кривой, которая проходит через эти точки. Рассмотрим вопрос о том, как построить



кривую в данном случае. Введем параметр, значения которого упорядочены в такой же последовательности, как будет проводиться кривая:

t	?	?	t
t_0	x_0	u_0	t_0
t_1	x_1	u_1	t_1
t_2	x_2	u_2	t_2
t_3	x_3	u_3	t_3

Вариантов параметризации много, например, один из простейших способов это t_0 взять равным нулю, а t_1 взять как сумму t_0 и расстояния между точками (x_0, u_0) и (x_1, u_1), t_2 взять как сумму t_1 и расстояния между точками (x_1, u_1) и (x_2, u_2), и так далее. Теперь можем для любого значения параметра t определить соответствующее значение x, и, с помощью обычной интерполяции, в соответствующей точке t найти приближенной значение u. Таким образом, получим точку с координатами (x, u), соответствующую этому значению параметра t. Поэтому, если в качестве t будем выбирать значение, находящееся на отрезке, содержащем (t_0, t_1, t_2, t_3), то получаем задачу интерполяции, а если вне этого отрезка, то получится задача экстраполяции.

Решение нелинейных уравнений с помощью интерполяции.

На практике может быть ситуация, что функция задана в виде набора сеточных значений (x_n, u_n), $n = \overline{0, N}$, и есть потребность найти нуль соответствующей функции: $u(x) = 0$. Мы не можем применить метод дихотомии, потому что если возьмем какие-то два узла и значения функции в этих узлах, то уже после первого деления получим точку, которая не совпадает ни с одним из узлов, значит, не можем посчитать значение функции в этой точке. Мы не можем применять метод Ньютона, потому что, опять же, после первой итерации получим следующее приближение для x, которое не совпадает ни с одним из узлов, и считать производную дальше не получится. Но можно сделать следующее. У

нас, фактически, есть таблица

$$\begin{array}{cc} x_0 & u_0 \\ x_1 & u_1 \\ \vdots & \vdots \\ x_N & u_N \end{array}$$

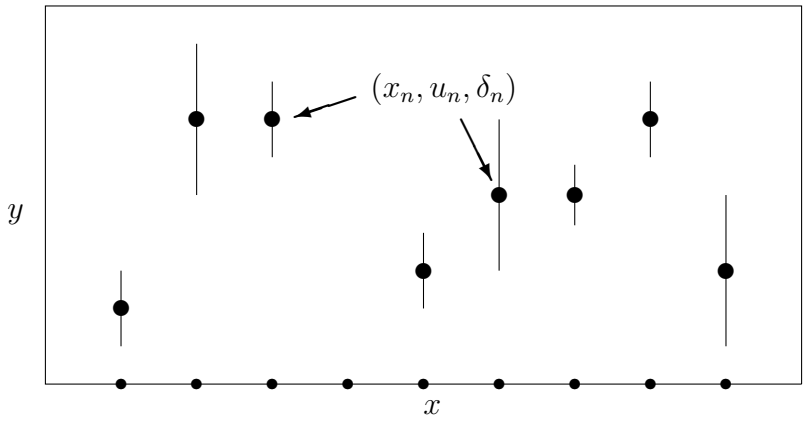
В качестве аргументов рассматриваем массив $(u_0, u_1, \cdots, u_N)$, а в качестве значений функции рассматривать $(x_0, x_1, \cdots, x_N)$. Далее задаемся вопросом: чему равно приближенное значение функции $x(u)$ при значении аргумента, равном нулю. В результате получим приближенное значение x, в котором функция равна нулю.

Стоит отметить, что если известна дополнительная априорная информация о функции, например, значения производной в сеточных узлах, то можно построить более точный метод вычисления приближенного значения. В данном случае, можно использовать интерполяционный многочлен Эрмита, который при построении использует информацию не только о значении функции, но и о значении производной в каждом узле.

Лекция 7

Среднеквадратичная аппроксимация обобщённым многочленом.

Вернемся к функции, которая задана на сетке соответствующим набором сеточных значений. Отличие аппроксимации от интерполяции заключается в следующем. Зачастую на практике точки измерены не точно, а с некоторой погрешностью δ_n. В этом случае



не имеет смысла строить приближенную функцию, которая проходит через эти точки точно, а нужно провести функцию как-то приближенно, при этом желательно, чтобы она попадала в коридоры погрешностей. По аналогии с предыдущей лекцией, введем набор базисных функций и запишем обобщенный многочлен:

$$\varphi_k(x), \quad k = \overline{0, K}, \quad \Phi_K(x) = \sum_{k=0}^{K} c_k \varphi_k(x), \quad K < N,$$

где, в отличие от интерполяции, $K \neq N$. Также на практике часто известен закон для аппроксимирующей функции, например, если известно, что точки должны быть расположены на графике по линейному закону, то K имеет смысл брать равным единице, если закон квадратичный, то $K = 2$, это в случае системы степеней в качестве базисных функций. И в этом случае аппроксимирующую функцию можно проводить через сколь угодно большое число точек N, в то время как в случае интерполяции уже при $N \approx 7-8$ интерполяционный

многочлен Ньютона имел достаточно большие осцилляции. Сейчас же мы знаем, что если у нас линейный закон, то независимо от числа точек пройдет прямая линия.

Итак, **задача аппроксимации**: каким-то образом выбрать базисные функции и их количество, и наилучшим образом провести аппроксимирующий обобщенный многочлен рядом с набором точек, то есть, нужно выбрать c_k. Последовательно ответим на три вопроса:
1) Как найти коэффициенты c_k=?
2) Как оптимальным образом выбирать $\varphi_k(x)$ = ?
3) Как выбирать количество базисных функций, если нет никакой априорной информации о поведении функции, K = ?
Для ответа на первый вопрос обратимся к методу наименьших квадратов. Запишем квадрат нормы следующей разности и будем подбирать коэффициенты c_k таким образом, что она стремится к нулю:

$$\|\Phi_K(x) - u(x)\|^2 \to 0.$$

Одним из свойств нормы является следующее:

$$\|y\| = 0 \Leftrightarrow y = \theta,$$

поэтому чем норма $\|\Phi_K(x) - u(x)\|$ ближе к нулю, тем разность $\Phi_K(x) - u(x)$ ближе к нулевому элементу, то есть, тем лучше $\Phi_K(x)$ удовлетворяет неизвестной функции $u(x)$. На практике норма никогда не обратится в нуль, так как есть ошибки. Почему необходимо минимизировать не норму разности, а квадрат нормы разности? Приведем простейший пример из математического анализа. Выражение $\|\Phi_K(x) - u(x)\|^2$ – это квадратичный функционал, то есть элементу пространства ставится в соответствие число. Квадратичная функция одной переменной имеет одну точку, реализующую минимум. В случае кубической функции такой точки нет, так как функция уходит на $-\infty$. Функция четвертой степени в общем случае имеет три точки локального экстремума, поэтому сначала надо найти эти точки, потом определить, какие из них являются локальными минимумами, а потом из них выбрать глобальный минимум. Поэтому можно было бы вводить и метод четвертых степеней или шестых степеней, просто у метода наименьших квадратов более простая реализация на практике. Возникает вопрос: зачем мы добавили квадрат, и почему бы не ограничиться нормой? В случае просто нормы, по аналогии с математическим анализом, функция будет выглядеть как модуль. Точка, реализующая минимум одна, как и в случае с квадратом нормы, но в этой точке производной не существует, и возникают проблемы с практическим критерием нахождения точки минимума. В связи с этим, в случае квадрата нормы, достаточно наложить условия на коэффициенты c_k, при которых будет реализовываться минимум.

Для простоты рассматриваем достаточно общий случай – пространство L_2:

$$\|z\|_{L_2} = \sqrt{(z,z)}, \quad (u,\upsilon) = \int\limits_a^b u(x)\upsilon(x)dx.$$

С учетом этого можно переписать квадрат нормы в следующем виде:

$$\Big(\sum_{k=0}^{K} c_k\varphi_k(x) - u(x), \sum_{k=0}^{K} c_k\varphi_k(x) - u(x)\Big).$$

Если раскрыть все скобки, то получится квадратичная форма относительно коэффициентов c_k, и поэтому условие минимума будет следующее:

$$\frac{d}{dc_m}\Big(\sum_{k=0}^{K} c_k\varphi_k(x) - u(x), \sum_{k=0}^{K} c_k\varphi_k(x) - u(x)\Big) = 0, \quad m = \overline{0,K}.$$

Так это симметричная квадратичная форма, то брать производную можно только от левой части скалярного произведения. В результате получается

$$\Big(\varphi_m(x), \sum_{k=0}^{K} c_k\varphi_k(x) - u(x)\Big) = 0, \quad m = \overline{0,K}.$$

Раскрываем скобки и получаем

$$\sum_{k=0}^{K} c_k\Big(\varphi_m(x), \varphi_k(x)\Big) = \Big(\varphi_m(x), u(x)\Big), \quad m = \overline{0,K}.$$

Таким образом, имеем систему линейных алгебраических уравнений относительно коэффициентов c_k, при которых аппроксимирующая функция наилучшим образом приближает неизвестную функцию $u(x)$ в пространстве L_2. Перепишем эту систему в матричной форме:

$$\begin{pmatrix} (\varphi_0,\varphi_0) & (\varphi_0,\varphi_1) & \cdots & (\varphi_0,\varphi_K) \\ (\varphi_1,\varphi_0) & (\varphi_1,\varphi_1) & \cdots & (\varphi_1,\varphi_K) \\ \vdots & \vdots & \ddots & \vdots \\ (\varphi_K,\varphi_0) & (\varphi_K,\varphi_1) & \cdots & (\varphi_K,\varphi_K) \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_K \end{pmatrix} = \begin{pmatrix} (\varphi_0,u) \\ (\varphi_1,u) \\ \vdots \\ (\varphi_K,u) \end{pmatrix}$$

Получается, что все элементы матрицы и элементы правой части известны, поэтому можно решить эту систему относительно c_k, при условии, что определитель матрицы отличен от нуля. Решаем эту систему методом Гаусса и находим коэффициенты c_k, $k = \overline{0,K}$, с помощью которых можно выписать аппроксимирующую функцию. Обратим внимание на то, что матрица системы является матрицей Грама.

Как вычислять необходимые скалярные произведения? Элементы матрицы Грама можно вычислить и по аналитической формуле, однако $\varphi_k(x)$ в общем случае могут иметь

достаточно неприятный вид, поэтому эти элементы можно считать численно. Элементы же правой части можно считать только численно, потому что функция $u(x)$ задана набором своих сеточных значений. Эти скалярные произведения можно посчитать, например, с помощью формулы трапеций, следующим образом:

$$\Big(\varphi_k(x), u(x)\Big) = \sum_{n=1}^{N} \frac{\varphi_k(x_n)u(x_n) + \varphi_k(x_{n-1})u(x_{n-1})}{2} \cdot (x_n - x_{n-1}) =$$

$$= \sum_{n=1}^{N} \frac{\varphi_k(x_n)u_n + \varphi_k(x_{n-1})u_{n-1}}{2} \cdot (x_n - x_{n-1}).$$

Итак, коэффициенты c_k подобрали.

Неортогональные базисы и обусловленность алгоритма (система степеней).

Теперь поговорим об особоенностях выбора базисных функций. Первое, что приходит в голову, по аналогии с построением интерполяционного многочлена Ньютона, в качестве базисных функций выбирать систему степеней: $\varphi_k(x) = x^k$, $k = \overline{0, K}$. С одной стороны стороны, этот выбор интуитивно понятен, потому что чем больше K выберем, тем аппроксимация будет полиномом более высокой степени, а значит, будет более точной. Но рассмотрим частный случай на отрезке $[0, 1]$ и вычислим следующее скалярное произведение:

$$(\varphi_k, \varphi_m) = \int_0^1 x^k x^m dx = \int_0^1 x^{k+m} dx = \frac{1}{k+m+1} x^{k+m+1}\Big|_0^1 = \frac{1}{k+m+1},$$

то есть элементы матрицы Грама имеют такой вид. Вспомним, что элементы матрицы Гильберта тоже имеют такой вид. То есть матрица Грама в случае степенных базисных функций и отрезка $[0, 1]$ превращается в матрицу Гильберта:

$$\begin{pmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \cdots \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & & \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & & \\ \vdots & \vdots & & & \end{pmatrix},$$

которая является плохо обусловленной, то есть в процессе реализации метода Гаусса для решения системы ошибки машинного округления сильно влияют на решение, и решение становится неадекватным. Если размерность матрицы Гильберта равна 12, то ошибки присутствуют во всех знаках решения. Таким образом, степенные базисные функции вроде бы являются достаточно физичными, однако их имеет смысл использовать только в случае небольшого K, то есть $K \geq 6$ брать не стоит, так как ошибки аппроксимации будут колоссальными.

Ортогональные базисы.

Тогда какой выбор базисных функций является наиболее оптимальным? Если все базисные функции попарно ортогональны, то все скалярные произведения, не лежащие на главной диагонали, обратятся в нуль. В этом случае коэффициенты находятся как $c_k = \dfrac{(\varphi_k, u)}{(\varphi_k, \varphi_k)}$, и количество операций уже не $O(K^3)$, а $O(K^1)$. Но главное заключается в

```
from numpy import zeros, array, linspace, dot, linalg
from matplotlib.pyplot import plot, axes, xlim, ylim

def Integration(x,u) :
    N = len(x) - 1
    int = 0.
    for n in range(1,N+1) :
        int = int + (u[n] + u[n-1])/2*(x[n] - x[n-1])
    return int

def phi(x,k) :
    return x**k

def Approximation(x,u,phi,K,a) :
    N = len(x)-1; A = zeros((K+1,K+1)); B = zeros((K+1,1))
    for m in range(K+1) :
        for k in range(K+1) :
            A[m,k] = Integration(x,phi(x,m)*phi(x,k))
        B[m,0] = Integration(x,phi(x,m)*u)
    C = linalg.solve(A,B)
    sum = 0.
    for k in range(K+1) :
        sum = sum + C[k,0]*phi(a,k)
    return(sum)

x = array([1, 2, 3, 5, 6, 7, 8, 9])
u = array([1, 4, 4, 2, 3, 3, 4, 2])

K = 4

plot(x,u,'go',MarkerSize = 7)

x_approx = linspace(1,9,100)
u_approx = Approximation(x,u,phi,K,x_approx)

plot(x_approx,u_approx,'-r')
xlim([0,10]); ylim([0,6]); axes().set_aspect(1)

# Листинг программы, реализующей построение
# аппроксимирующей функции
```

Рис. 7.1: Пример программной реализации 7-1

том, что уже не будут накапливаться ошибки машинного округления, как в случае метода Гаусса, который в случае плохой обусловленности системы приводит к катастрофическим ошибкам. Стоит отметить, что в случае аппроксимации требуется упорядоченность узлов для функции интегрирования. Из примера программной реализации 7-1 видно, что при изменении K существенно меняется аппроксимирующая функция. Таким образом, остался важный вопрос о том, как выбирать K, который обсудим позднее. При $K = N - 1$ аппроксимация должна выродиться в интерполяцию, и аппроксимирующая функция должна пройти ровно через сеточные точки, однако как раз из-за плохой обусловленности матрицы

Гильберта проходить ровно через точки она не будет.

Можно использовать следующий приём. У нас $x \in [a, b]$, введем $x^* = \dfrac{a+b}{2}$, расположим начало координат в середине отрезка и далее отмасштабируем этот отрезок в следующий: $x \in [-1, 1]$. В этом случае

$$(\varphi_k, \varphi_m) = \int_{-1}^{1} x^{k+m} dx = \frac{1}{k+m+1} x^{k+m+1} \Big|_{-1}^{1} = \begin{cases} \dfrac{2}{m+k+1} & \text{, если } m+k \text{ – четное} \\ 0 & \text{, если } m+k \text{ – нечетное} \end{cases}$$

Тогда матрица будет иметь следующий вид

$$\begin{pmatrix} 2 & 0 & \frac{2}{3} & 0 & \cdots \\ 0 & \frac{2}{3} & 0 & & \\ \frac{2}{3} & 0 & \frac{2}{5} & & \\ \vdots & \vdots & & & \end{pmatrix},$$

которая похожа на матрицу Гильберта, но у нее чередуются нулевые диагонали, а так как в матрице встречается достаточно много нулей, то ошибки будут накапливаться медленнее, поэтому обусловленность матрицы такой системы будет лучше. В таком случае, как показано в обновленной программной реализации 7-2, аппроксимация действительно вырождается в интерполяцию, и аппроксимирующая кривая проходит ровно через набор точек.

Обработка экспериментов. Выбор весов и оптимального числа коэффициентов.

Теперь обсудим вопрос: как выбирать K? Обычно он выбирается двумя способами:
1) Если имеется априорная информация о том, какому закону должны удовлетворять точки, набор которых аппроксимируется, то сразу используется соответствующее значение K
2) В случае, если никакой априорной информации о законе нет, могут помочь значения погрешностей δ_n. **Практический алгоритм**:

$$\text{берем } K = 0: \quad \Phi_0(x) = c_0 \varphi_0(x)$$

Пусть в качестве базисных функций взяли систему степеней, то есть аппроксимировали набор точек константой. Теперь вопрос: удовлетворяет ли аппроксимирующая функция значениям, заданным с погрешностью? Посчитаем среднеквадратичное отклонение аппроксимирующег многочлена от соответствующего сеточного значения и сравним со среднеквадратичной ошибкой задания входных данных:

$$\sqrt{\frac{1}{(N+1)} \sum_{n=0}^{N} \Big(\Phi_0(x_n) - u_n \Big)^2} \quad \leq^? \quad \sqrt{\frac{1}{(N+1)} \sum_{n=0}^{N} \delta_n^2}.$$

```
from numpy import zeros, array, linspace, dot, linalg
from matplotlib.pyplot import plot, axes, xlim, ylim

def Integration(x,u) :
    N = len(x) - 1
    int = 0.
    for n in range(1,N+1) :
        int = int + (u[n] + u[n-1])/2*(x[n] - x[n-1])
    return int

def phi(x,k) :
    return x**k

def Approximation(x,u,phi,K,a) :
    N = len(x)-1; A = zeros((K+1,K+1)); B = zeros((K+1,1))
    for m in range(K+1) :
        for k in range(K+1) :
            A[m,k] = Integration(x,phi(x,m)*phi(x,k))
        B[m,0] = Integration(x,phi(x,m)*u)
    C = linalg.solve(A,B)
    sum = 0.
    for k in range(K+1) :
        sum = sum + C[k,0]*phi(a,k)
    return(sum)

x = array([1, 2, 3, 5, 6, 7, 8, 9])
u = array([1, 4, 4, 2, 3, 3, 4, 2])

plot(x,u,'go',MarkerSize = 7)

a = min(x); b = max(x)
x_0 = (a + b)/2; x = (x - x_0)/((b - a)/2)

K = 7

x_approx = linspace(-1,1,100)
u_approx = Approximation(x,u,phi,K,x_approx)

x_approx = x_approx*((b - a)/2) + (a + b)/2
plot(x_approx,u_approx,'-r')
xlim([0,10]); ylim([0,6]); axes().set_aspect(1)

# Листинг программы, реализующей построение
# аппроксимирующей функции
```

Рис. 7.2: Пример программной реализации 7-2

Если среднеквадратичное отклонение аппроксимирующего многочлена от соответствующего сеточного значения меньше либо равно среднеквадратичной ошибке задания входных данных, то считаем, что это хороший аппроксимирующий многочлен, который попадает в коридор среднеквадратичной ошибки задания входных данных. Если больше, то увеличиваем K на единицу:

$$K = 1: \quad \Phi_1(x) = c_0\varphi_0(x) + c_1\varphi_1(x),$$

и повторяем процедуру. То есть с помощью программы снова нашли коэффициенты, и опять делаем сравнение. Если неравенство не выполняется, то еще раз увеличиваем K на единицу:

$$K = 2: \quad \Phi_K(x) = \sum_{k=0}^{K} c_k\varphi_k(x).$$

То есть каждый раз прибавляем очередную базисную функцию, находим соответствующие коэффициенты разложения и делаем проверку, насколько соответствующий аппроксимирующий многочлен соотносится со среднеквадратичной погрешностью задания входных данных. Делаем это до тех пор, пока не попадем в в коридор среднеквадратичной погрешности

задания входных данных. В этом случае сразу же останавливаемся, так как продолжать процесс не имеет смысла, потому что в случае системы степеней $\varphi_k(x) = x^k$ получается предельный случай, когда $K \equiv N$, при котором аппроксимация выродится в интерполяцию, и среднеквадратичное отклонение аппроксимирующего многочлена от соответствующего сеточного значения станет равным нулю. Тогда, начиная с какого-то K^*, при котором впервые выполнилось неравенство, с прибавлением очередной базисной функции выражение для среднеквадратичного отклонения продолжит уменьшаться, тогда вопрос, зачем останавливаться на K^*, если дальше погрешность становится всё меньше? Вспомним, что аппроксимация используется в том числе, когда количество точек огромно, а интерполяционный многочлен Ньютона с увеличением числа точек сильно осциллирует между узлами. И в этом случае получается, что погрешность уменьшается, а осцилляции между узлами сильно увеличиваются, поэтому выбираем такое K^*, которое удовлетворяет погрешности с минимальными осцилляциями между узлами. Таким образом, выбор оптимального K согласуется с погрешностью задания входных данных. Еще раз заметим, что это актуально только в том случае, если никакой априорной информации о характере закона, которому должны удовлетворять точки, нет.

Также возникает вопрос, почему рассматривается среднеквадратичная ошибка, а не поточечная, то есть почему бы не проводить такую проверку поточечно:

$$|\Phi_0(x_n) - u_n| \quad \leq^? \quad \delta_n, \quad n = \overline{0, N},$$

и увеличивать K на единицу, пока это неравенство не будет выполнено во всех узлах. Ответ довольно простой: обычно в эксперименте ошибки δ_n определяются статистически, значит, есть вероятность того, что для некоторой точки оценка погрешности неадекватная, поэтому поточечно значениям δ_n доверять нельзя, и поточечная проверка является излишне строгой, которая приведет к тому, что итерационный процесс прервется при большем K, чем нужно, что означает более сильные осцилляции между узлами.

Лекция 8

Тема этой лекции — численное дифференцирование.

Пусть функция задана на какой-либо сетке набором своих сеточных значений, задача ставится следующим образом: найти в какой-то точке, например, первую производную или другого порядка по сеточным значениям функции.

$$(x_n, u_n), \quad n = \overline{0, N}; \quad u^{(k)}(x) = ? \tag{8.1}$$

Проблема состоит в том, что нам дана не сама функция $u(x)$, а только ее значения u_n в некоторых точках x_n. Разумно соответствующий набор точек интерполировать, либо аппроксимировать, то есть построить либо обобщенный интерполяционный многочлен, который проходит через эти точки, либо, если известны погрешности, с которыми вычислены, либо измерены в эксперименте эти сеточные значения, построить аппроксимирующей обобщенный многочлен, и уже их продифференцировать и соответственно найти выражение для производной нужного порядка. На этой лекции мы будем предполагать, что все сеточные значения известны точно и соответственно через набор вот этих точек

Мы будем предполагать, что все сеточные значения известны точно и соответственно через набор этих точек будем проводить интерполяционный многочлен Ньютона и уже его дифференцировать необходимое число раз. Таким образом, будем получать численные формулы для приближенного вычисления производных необходимого нам порядка и еще с необходимым порядком точности.

Приближенная формула для производной 1-го порядка.

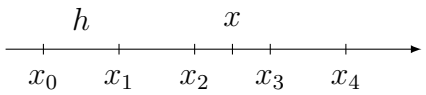
Напомним, как выглядит интерполяционный многочлен Ньютона

$$u(x) \simeq \sum_{n=0}^{N} \frac{u_{0,n}}{n!} \prod_{k=0}^{n-1} (x - x_k) \tag{8.2}$$

Выпишем несколько слагаемых явно

$$u(x) \simeq u_{0,0} + u_{0,1}(x - x_0) + \frac{u_{0,2}}{2}(x - x_0)(x - x_1) + O(h^{\square})$$

Какую степень поставить у h? Для простоты объяснений, будем предполагать, что у нас равномерная сетка с шагом h, тогда справедливо $(x - x_k) = c_k h$, где c_k, некоторая константа, которая зависит от расположения точки x, например:



$x - x_0 = 2.5h,\ x - x_3 = -0.5h$ и т.д.

Поэтому $(x - x_0)(x - x_1) = c_0 c_1 h^2$, т.е. пропорционально h^2, следующее слагаемое будет пропорционально уже h^3

$$u(x) \simeq u_{0,0} + u_{0,1}(x - x_0) + \frac{u_{0,2}}{2}(x - x_0)(x - x_1) + O(h^3) \tag{8.3}$$

Отметим, что мы хоть и пишем $O(h^3)$, равенство все равно остается приближенным, так как интерполяционный многочлен Ньютона даёт верное равенство только в узлах сетки. Продифференцируем, $u_{0,0}$ является числом, оно даст 0, дифференцируя квадратичное слагаемое, мы получим полином первой степени, который по порядку величины $O(h)$

$$u'(x) \simeq u_{0,1} + O(h^1) \tag{8.4}$$

Мы получили формулу для приближенного вычисления производной, кроме того, нам известен порядок точности этой формулы, он равен единице. Порядок точности нам нужен для проведения методики Ричардсона. Выписывая большее число слагаемых и дифференцируя их, мы можем заранее получить необходимый порядок точности.

Это существенно отличается от численного интегрирования. Напомним, что при получении формул приближенного вычисления интегралов для получения порядка точности нужно было раскладывать ошибку в ряд Тейлора и на основе того, каков порядок у главного члена разложения ошибки, мы делали вывод о значении порядка точности. Здесь мы делаем наоборот: хотим формулу с заданным порядком точности и уже потом получаем её.

Выпишем явно разделенную разность $u_{0,1}$

$$u'(x) \simeq \frac{u_{0,0} - u_{1,0}}{x_0 - x_1} + O(h^1) = \frac{u_0 - u_1}{x_0 - x_1} + O(h^1), \quad x \in [x_0, x_1] \tag{8.5}$$

Подчеркнем, что эта формула верна именно внутри отрезка $[x_0, x_1]$, так как интерполяционный многочлен Ньютона даёт большие погрешности вне рассматриваемого отрезка, т.е. при экстраполяции. В литературе формулу для разностной производной пишут следующим образом:

$$u'_n = \frac{u_n - u_{n-1}}{h} + O(h^1) \tag{8.6}$$

Обращаем ваше внимание, что вместо u'_n можно было писать u'_{n-1} или при любом другом $n \in [n-1, n]$, при этом правая часть формулы не поменяется. Однако есть один нюанс, если мы возьмём $n - \frac{1}{2}$, то формула будет иметь повышенный порядок точности. Все дело в том, что если честно брать производную от квадратичного слагаемого в формуле (8.3), то мы получим

$$u'(x) \simeq u_{0,1} + u_{0,2}\left(x - \frac{x_0 + x_1}{2}\right) + O(h^2), \quad \longrightarrow \quad u'_{n-\frac{1}{2}} = \frac{u_n - u_{n-1}}{h} + O(h^2) \tag{8.7}$$

Обычно пишут немного по-другому. Так как $n - \frac{1}{2}$ является средней точкой отрезка $[x_{n-1}, x_n]$, то используя n вместо $n+1$, то есть отрезок $[x_{n-1}, x_{n-1}]$, средней точкой которого служит точка n, формула примет вид

$$u'_n = \frac{u_{n+1} - u_{n-1}}{2h} + O(h^2) \tag{8.8}$$

Приближенная формула для производной 2-го порядка.

Продифференцируем еще раз следующее выражение

$$u'(x) \simeq u_{0,1} + u_{0,2}\left(x - \frac{x_0 + x_1}{2}\right) + O(h^2), \quad \longrightarrow \quad u''(x) \simeq u_{0,2} + O(h^1) \tag{8.9}$$

Напомним, как считается разделенная разность в n-м узле k-го порядка (Лекция 5):

$$u_{n,k} = k\frac{u_{n,k-1} - u_{n+1,k-1}}{x_n - x_{n+k}} \tag{8.10}$$

По этой формуле получим

$$u''(x) \simeq \frac{2}{x_0 - x_2}\left(\frac{u_{0,0} - u_{1,0}}{x_0 - x_1} - \frac{u_{1,0} - u_{2,1}}{x_1 - x_2}\right) + O(h^1) \simeq \frac{2}{x_0 - x_2}\left(\frac{u_0 - u_1}{x_0 - x_1} - \frac{u_1 - u_2}{x_1 - x_2}\right) + O(h^1) \tag{8.11}$$

сразу введём переобозначения $x_0 \leftrightarrow x_{n-1}$, $x_1 \leftrightarrow x_n$, $x_2 \leftrightarrow x_{n+1}$, сетка по-прежнему равномерная и имеет шаг h

$$u''_\square = \frac{u_{n+1} - 2u_n + u_{n-1}}{h^2} + O(h^1) \tag{8.12}$$

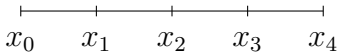
индекс у u'' можно выбрать любой из отрезка $[n-1, n+1]$, но можно показать, что выбрав серединку отрезка, порядок точности возрастет и будет равен 2

$$u''_n = \frac{u_{n+1} - 2u_n + u_{n-1}}{h^2} + O(h^2) \tag{8.13}$$

В литературе формулы (8.8,8.13) называют симметричными, либо центральными.

Несимметричные формулы для производных.

Рассмотрим следующую сетку



Мы можем посчитать первую производную со вторым порядком точности в узлах x_1, x_2 и x_3. В крайних узлах порядок точности будет первым. Возникает вопрос, а можно ли для нецентральных узлов высчитать приближенное значение производной со вторым порядком точности? Оказывается, можно.

$$\begin{aligned} u'_0 &\simeq \frac{u_0 - u_1}{-h} + \frac{2}{-2h}\left(\frac{u_0 - u_1}{-h} - \frac{u_1 - u_2}{-h}\right)\left(x_0 - \frac{x_0 + x_1}{2}\right) + O(h^2) \\ u'_0 &\simeq \frac{u_1 - u_0}{h} + \frac{u_2 - 2u_1 + u_0}{h^2}\frac{-h}{2} + O(h^2) \simeq \frac{3u_0 + 4u_1 - u_2}{2h} + O(h^2) \end{aligned} \tag{8.14}$$

введя переобозначения $x_0 \leftrightarrow x_n$, $x_1 \leftrightarrow x_{n+1}$, $x_{n+2} \leftrightarrow x_{n+1}$, получим формулу, которая выписывается в учебниках

$$u'_n \simeq \frac{3u_{n1} + 4u_{n+1} - u_{n+2}}{2h} + O(h^2) \tag{8.15}$$

Видим, что она имеет второй порядок точности, является несимметричной, иногда ее называют формулой для правой производной. Аналогично можно получить формулу для левой производной.

Таким образом, вы можете получить необходимые вам формулы для вычисления фактически любой производной с любым порядком точности. Они могут быть как симметричными, которые обладают обычно повышенным порядком точности, так и несимметричными, которые при своих вычислениях используют сеточные значения функции только с одной стороны от данного узла.

Обратим ваше внимание, что чем выше порядок точности, тем большее число узлов используется для вычисления производной.

Вычисления с контролем точности.

Естественно, программная реализация чрезвычайно простая, но мы с вами сразу же поговорим о вычислениях с контролем точности. Для вычисления производных применима все та же формула Рунге-Ромберга. Напоминаем ее смысл: проводим вычисления на

одной сетке, получаем какой-то результат U_N, сгущаем сетку в r раз, снова вычисляем приближенное значение U_{rN}. Далее считаем

$$R_{rN} = \frac{U_{rN} - U_N}{r^p - 1} \tag{8.16}$$

Это формула Рунге-Ромберга, она является апостериорной, асимптотически точной оценкой главного члена разложения в ряд Тейлора ошибки приближенного вычисления. То есть, если вы к вашему приближенному результату прибавите результат применения формулы Рунге-Ромберга, то вы получите уточненное значение с более высоким порядком точности. Но для того, чтобы применять эту формулу, нужно знать порядок точности p, который мы можем оценить только на равномерной сетке. Если у вас, например, неограниченная область, то можно проводить вычисления на квазиравномерных сетках, особенность которых здесь такая же, как и в случае вычисления интегралов. С одной стороны сетки являются неравномерными, например, они могут покрыть всю бесконечную прямую, но соседние интервалы близкие, за счет чего получается обосновать порядок точности. Этот момент мы в лекциях не рассматриваем, потому что это достаточно громоздко, но, например, у Николая Николаевича Калиткина есть учебник «Вычисления на квазиравномерных сетках», в нем приведены как преобразования порождающие различные квазиравномерные сетки, так и приближенные формулы для вычисления производных.

Программная реализация конкретных примеров.

Для тестирования воспользуемся следующим примером с точным ответом

$$(e^x)'\big|_{x=0} = 1. \tag{8.17}$$

Вычислим производную на серии сгущающихся сеток по несимметричной формуле $(q = 1)$ с первым порядком точности $(p = 1)$, и проведём её рекуррентное уточнение по Ричардсону.

Таблица приближённых значений производной:

	$p = 1$	$p = 2$	$p = 3$	$p = 4$	$p = 5$
$s = 0$	0.9516				
$s = 1$	0.9754	0.9992			
$s = 2$	0.9876	0.9998	1.0000		
$s = 3$	0.9938	0.9999	1.0000	1.0000	
$s = 4$	0.9969	1.0000	1.0000	1.0000	1.0000

Таблица оценок ошибок:

	$p=1$	$p=2$	$p=3$	$p=4$	$p=5$
$s=0$					
$s=1$	0.0238				
$s=2$	0.0122	0.0002			
$s=3$	0.0062	0.0001	0.0000		
$s=4$	0.0031	0.0000	0.0000	0.0000	

Таблица эффективных порядков точности:

	$p=1$	$p=2$	$p=3$	$p=4$	$p=5$
$s=0$					
$s=1$					
$s=2$	0.9642				
$s=3$	0.9820	1.9686			
$s=4$	0.9910	1.9843	2.9731		

Или можно построить график, отклонения при достаточно больших N связаны с некорректностью численного дифференцирования, а именно, т.к. компьютер хранит 16 знаков, при достаточно густой сетке u_n и u_{n-1} на компьютере просто совпадут, в числителе формулы разностной производной получится машинный 0, при делении которого на очень маленькое h получится не пойми что

Разберём сразу несколько типовых ошибок. Допустим, что при переписывании программы из лекции 4 (пример 4.2.), мы забыли изменить p и q. Таким образом, имеем $p=2,\ q=2$. Рассмотрим результаты работы программы.

Таблица приближённых значений производной:

	$p=2$	$p=4$	$p=6$	$p=8$	$p=10$
$s=0$	0.9516				
$s=1$	0.9754	0.9833			
$s=2$	0.9876	0.9917	0.9922		
$s=3$	0.9938	0.9958	0.9961	0.9962	
$s=4$	0.9969	0.9979	0.9981	0.9981	0.9981

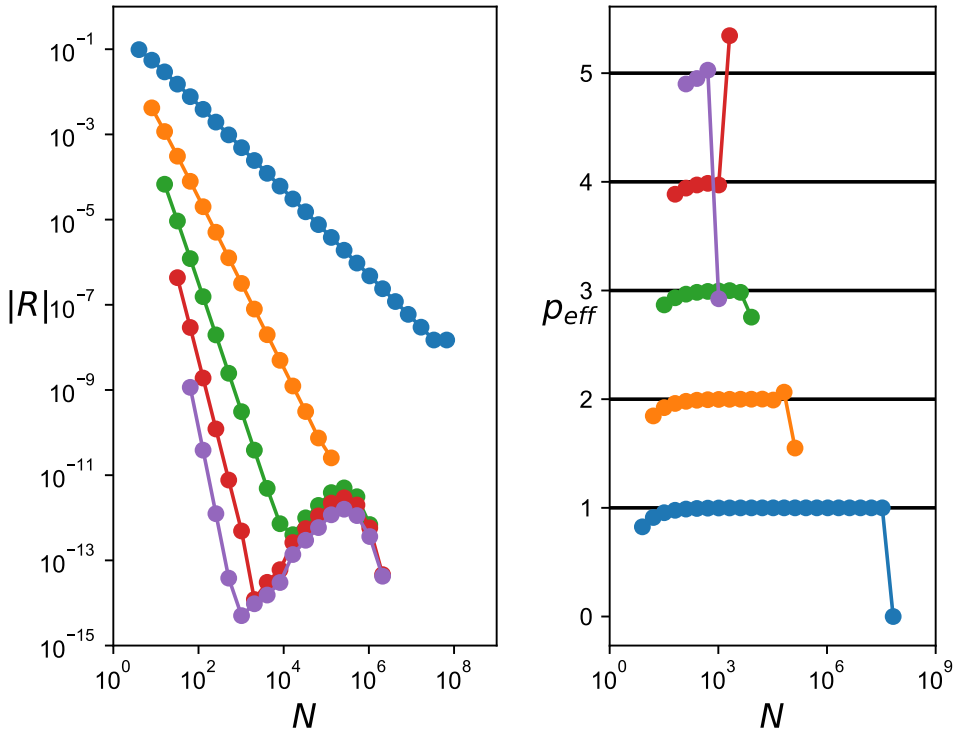


Рис. 8.1: Графики сходимости (слева) и соответствующие им эффективные порядки точности (справа).

Таблица оценок ошибок:

	$p=2$	$p=4$	$p=6$	$p=8$	$p=10$
$s=0$					
$s=1$	0.0079				
$s=2$	0.0041	0.0006			
$s=3$	0.0021	0.0003	0.0001		
$s=4$	0.0010	0.0001	0.0000	0.0000	

Таблица эффективных порядков точности:

	$p=2$	$p=4$	$p=6$	$p=8$	$p=10$
$s=0$					
$s=1$					
$s=2$	0.9642				
$s=3$	0.9820	0.9992			
$s=4$	0.9910	0.9998	0.9999		

Как видим, эффективный порядок точности, который показывает сколько членов разложения в ряд Тейлора схема передала точно не совпадает $p = 2$, кроме того оказалось, что уточненная схема также имеет лишь первый порядок точности ($q = 0$).

Теперь, допустим, что $p = 1$, а $q = 2$. Рассмотрим результаты выполнения программы.

Таблица приближённых значений производной:

	$p = 1$	$p = 3$	$p = 5$	$p = 7$	$p = 9$
$s = 0$	0.9516				
$s = 1$	0.9754	0.9992			
$s = 2$	0.9876	0.9998	0.9999		
$s = 3$	0.9938	0.9999	1.0000	1.0000	
$s = 4$	0.9969	1.0000	1.0000	1.0000	1.0000

Таблица оценок ошибок:

	$p = 1$	$p = 3$	$p = 5$	$p = 7$	$p = 9$
$s = 0$					
$s = 1$	0.0238				
$s = 2$	0.0122	0.0001			
$s = 3$	0.0062	0.0000	0.0000		
$s = 4$	0.0031	0.0000	0.0000	0.0000	

Таблица эффективных порядков точности:

	$p = 1$	$p = 3$	$p = 5$	$p = 7$	$p = 9$
$s = 0$					
$s = 1$					
$s = 2$	0.9642				
$s = 3$	0.9820	1.9686			
$s = 4$	0.9910	1.9843	1.9994		

Здесь мы видим что первая колонка совпадает с теоретическим порядком точности, но во второй, где мы ожидаем $p = 3$, получился только второй порядок точности.

Рассмотрим другой наглядный пример, имеющий аналитическое решение

$$\left(x^3\right)'\Big|_{x=0} \equiv 3x^2\big|_{x=0} = 0. \tag{8.18}$$

Рассмотрим результаты работы программы.

Таблица приближённых значений производной:

	$p=1$	$p=2$	$p=3$	$p=4$	$p=5$
$s=0$	0.0100				
$s=1$	0.0025	−0.0050			
$s=2$	0.0006	−0.0013	0.0000		
$s=3$	0.0002	−0.0003	0.0000	0.0000	
$s=4$	0.0000	−0.0001	0.0000	0.0000	0.0000

Таблица оценок ошибок:

	$p=1$	$p=2$	$p=3$	$p=4$	$p=5$
$s=0$					
$s=1$	−0.0075				
$s=2$	−0.0019	0.0013			
$s=3$	−0.0005	0.0003	0.0000		
$s=4$	−0.0001	0.0001	0.0000	0.0000	

Таблица эффективных порядков точности:

	$p=1$	$p=2$	$p=3$	$p=4$	$p=5$
$s=0$					
$s=1$					
$s=2$	2.0000				
$s=3$	2.0000	2.0000			
$s=4$	2.0000	2.0000	nan		

Напомним, что мы используем формулу для вычисления приближенного значения производной с первым порядком точности. Почему в первой колонке эффективный порядок точности равен 2? Все дело в явном виде функции, это полином второй степени, его ряд Тейлора содержит только квадратичное слагаемое. По этой же причине в третьей колонке мы получили «nan» — все члены разложения в ряд Тейлора передались абсолютно точно.

Положим порядок точности $p=2$, и посмотрим на изменения в выходных данных.

Таблица приближённых значений производной:

	$p=2$	$p=3$	$p=4$	$p=5$	$p=6$
$s=0$	0.0100				
$s=1$	0.0025	0.0000			
$s=2$	0.0006	0.0000	0.0000		
$s=3$	0.0002	0.0000	0.0000	0.0000	
$s=4$	0.0000	0.0000	0.0000	0.0000	0.0000

Таблица оценок ошибок:

	$p=2$	$p=3$	$p=4$	$p=5$	$p=6$
$s=0$					
$s=1$	0.0025				
$s=2$	-0.0006	0.0000			
$s=3$	-0.0002	0.0000	0.0000		
$s=4$	-0.0000	0.0000	0.0000	0.0000	

Таблица эффективных порядков точности:

	$p=2$	$p=3$	$p=4$	$p=5$	$p=6$
$s=0$					
$s=1$					
$s=2$	2.0000				
$s=3$	2.0000	nan			
$s=4$	2.0000	nan	nan		

Как видим, здесь уже во второй колонке появились значения «nan», объяснение этого эффекта все то же — наша функция есть полином.

Рассмотрим ещё один пример, но теперь такой, что у функции не существует производной

$$\left(\sqrt{-x}\right)'\Big|_{x=0} = \emptyset. \tag{8.19}$$

Рассмотрим результаты работы программы.

Таблица приближённых значений производной:

	$p=1$	$p=2$	$p=3$	$p=4$	$p=5$
$s=0$	-3.2				
$s=1$	-4.5	-5.8			
$s=2$	-6.3	-8.2	-9.0		
$s=3$	-8.9	-11.6	-12.7	-13.1	
$s=4$	-12.6	-16.4	-18.0	-18.7	-19.1

Таблица оценок ошибок:

	$p=1$	$p=2$	$p=3$	$p=4$	$p=5$
$s=0$					
$s=1$	-1.30				
$s=2$	-1.86	-0.8			
$s=3$	-2.6	-1.1	-0.5		
$s=4$	-3.7	-1.6	-0.8	-0.4	

Таблица эффективных порядков точности:

	$p=1$	$p=2$	$p=3$	$p=4$	$p=5$
$s=0$					
$s=1$					
$s=2$	-0.5000				
$s=3$	-0.5000	-0.5000			
$s=4$	-0.5000	-0.5000	-0.5000		

Программа что-то считает, как интерпретировать результат? Мы видим, что эффективный порядок точности отрицательный, это означает что ни одного члена разложения в ряд Тейлора не передалось точно, то есть по этим данным можно понять, что производной у функции в этой точке не существует.

Лекция 9

Методы минимизации

Существует чрезвычайно много различных методов минимизации и эти методы крайне важны при обработке физических экспериментов. Поэтому, в данной лекции будут рассмотрены только самые базовые методы.

Начнем с поиска минимума функции одной переменной. Рассмотрим произвольную функцию $f(x)$, которую будем считать кусочно-непрерывной. И ставится задача о поиске минимума этой функции на каком то отрезке $[a, b]$, $\min\limits_{x\in[a,b]} f(x) = ?$.

Метод золотого сечения

Это самый базовый метод поиска локального минимума функции одной переменной. Рассмотрим отрезок $[a, b]$, на котором функция $f(x)$ кусочно-непрерывна, и выберем две точки x_0 и x_1, в этих точках вычисляется значения функции $f(x_0)$ и $f(x_1)$ (см. рис.9.1), таким образом, локальный минимум функции находится в одном из двух отрезков $[a, x_1]$ или $[x_0, b]$. Выберем точку в которой значение функции наименьшее (пусть это будет точка x_0) и выберем отрезок соответствующий этой точке, в нашем случае это отрезок $[a, x_1]$. Далее, выбираем на отрезке $[a, x_1]$ две точки и вычисляем значение функции, снова выбирая такой отрезок, внутри которого лежит точка, в которой функция принимает наименьшее значение. Таким образом, отрезок, в котором лежит точка минимума функции уменьшается, и в пределе, при бесконечном количестве итераций, отрезок стягивается в точку, в которой реализуется локальный минимум функции $f(x)$. Обратим внимание, что этот метод сходится лишь к одному из локальных минимумов, поэтому возможен случай когда метод сошелся к локальному минимуму на отрезке $[a, x_1]$, а глобальный минимум находится на отрезке $[x_1, b]$, поэтому при наличии нескольких локальных минимумов заранее

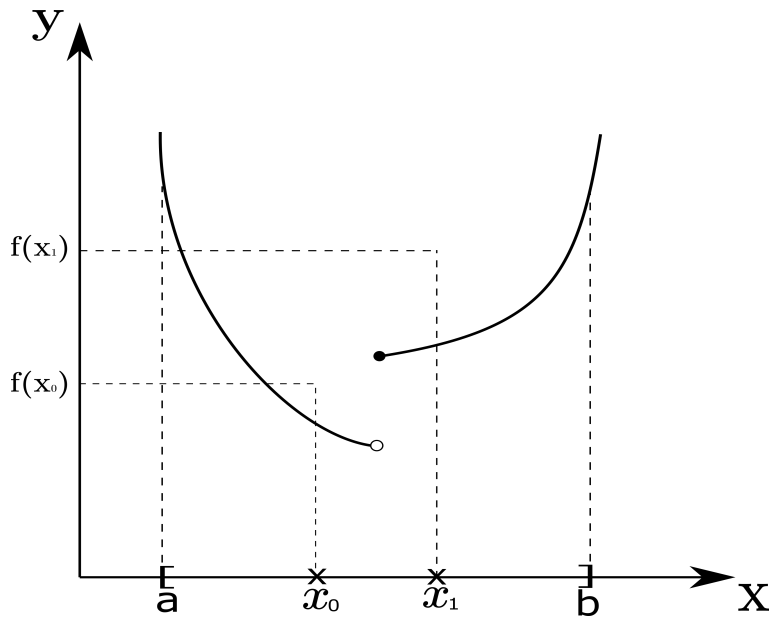


Рис. 9.1: Кусочно-непрерывная функция

неизвестно к какому именно сойдется метод и результат зависит от способа выбора точек x_0 и x_1.

Метод золотого сечения позволяет оптимизировать количество операций. В рассмотренном выше методе на каждой итерации значение функции вычисляется в двух точках, на практике, вычисление значения функции в точке может быть время затратным.
Примем наш отрезок за единицу и определим пропорцию (ξ) по которой будем класть на него две точки. Расположим точки симметрично на расстоянии ξ от концов отрезка (см. рис.9.2). Далее выбираем отрезок, содержащий точку, в которой функция принимает наименьшее значение. На следующей итерации мы выбираем точки так, чтобы они располагались в той же пропорции, и чтобы одна из точек совпадала с точкой на предыдущей итерации, что позволяет сократить количество вычислений. Таким образом, на следующей итерации получаем пропорцию:

$$\frac{1-2\xi}{1-\xi} = \frac{\xi}{1}$$

Найдем значения ξ для этой пропорции:

$$1-2\xi = \xi - \xi^2$$
$$\xi^2 - 3\xi + 1 = 0$$
$$D = (-3)^2 - 4 \cdot 1 \cdot 1 = (\sqrt{5})^2$$
$$\xi = \begin{cases} \frac{3+\sqrt{5}}{2} > 1 \\ \frac{3-\sqrt{5}}{2} \approx 0.38 \end{cases}$$

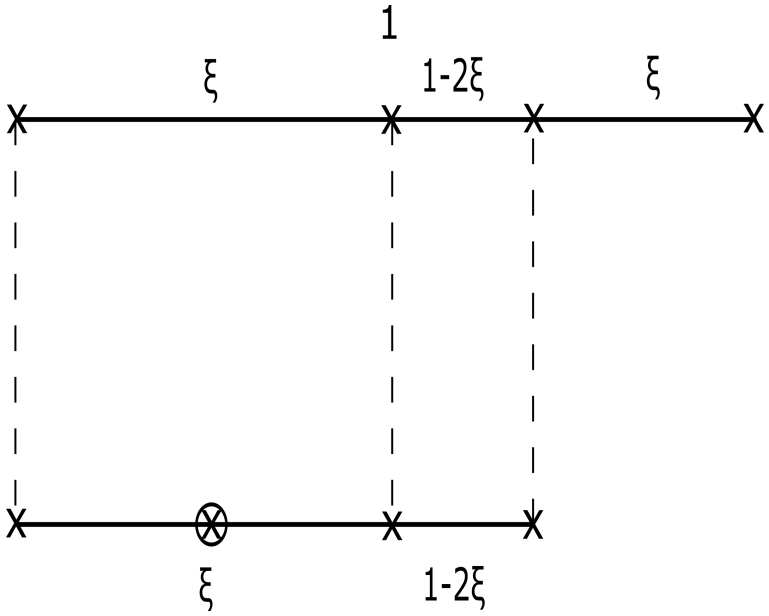


Рис. 9.2: Разбиение отрезка для метода золотого сечения

Поскольку $\xi > 1$ не подходит из геометрических соображений, выберем $\xi \approx 0.38$. Причем, в программной реализации используется приблизительное значение $\xi \approx 0.38$, а выражение $\xi = \frac{3-\sqrt{5}}{2}$.

Таким образом числа $\xi \approx 0.38$ и $1 - \xi \approx 0.62$ определяют пропорцию золотого сечения. Если на каждой итерации мы будем выбирать точки всегда с такой пропорцией ξ, то одна из точек на текущей итерации будет совпадать с одной из точек на предыдущей итерации, что дает возможность на каждой итерации вычислять значение функции только в одной точке, сокращая количество операций в два раза.

Заметим, что метод золотого сечения обеспечивает уменьшение количества операций по вычислению значения $f(x)$, но он не обеспечивает меньшего количества итераций, чем аналогичный метод с произвольным выбором значения ξ.

Поиск одномерного минимума с помощью метода Ньютона

Задача: $\min\limits_{x \in [a,b]} f(x) = ?$
Пусть функция $f(x) \in C^2[a,b]$.
Необходимое условие минимума это $f'(x) = 0$. Теперь воспользуемся методом ньютона и построим следующий итерационный процесс:
0. $\forall x_0$

1. $x_{n+1} = x_n - \frac{f'(x_n)}{f''(x_n)}, n = 0, 1, 2...$
Если функция $f(x)$ имеет несложный вид, то её производную можно найти аналитически или численно. Но, возможны случаи, когда не удается вычислить производную.
Введем определение:

Определение 1. ***Порядком метода*** *называется максимальной порядок производной минимизируемой функции, который используется в соответствующем методе.*

Поэтому, метод золотого сечения это метод нулевого порядка, метод с использованием метода Ньютона это метод второго порядка.

Метод координатного спуска

Теперь функция зависит от нескольких аргументов $f(x_1, ..., x_n)$ и нужно найти набор аргументов при котором реализуется минимум и соответствующее значение функции. Обозначим $x_1, .., x_n$ как $\vec{x}$, поэтому задачу можно поставить как поиск минимума функционала $\min f(\vec{x})$

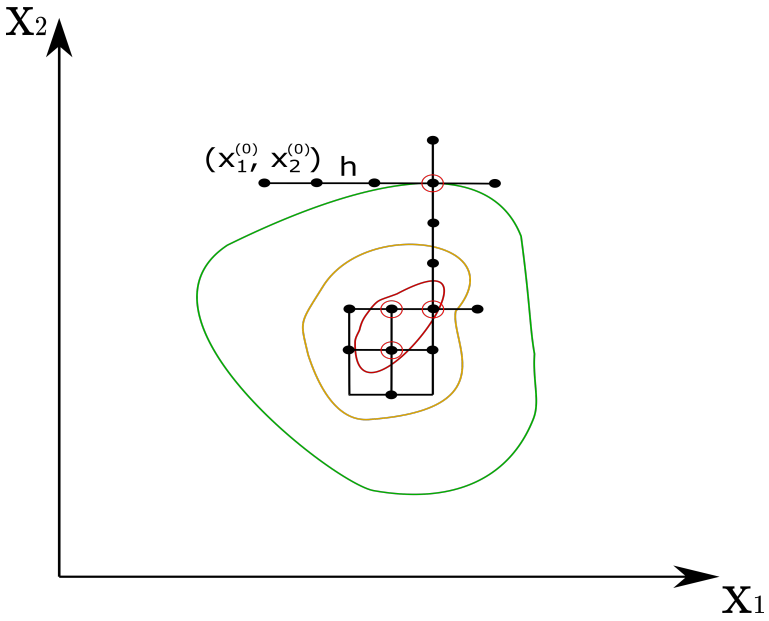


Рис. 9.3: Метод координатного спуска. Цветные линии – линии уровня функции. Точки – итерации.

Рассмотрим метод координатного спуска на двумерном примере:
1. Выбираем произвольное начальное приближение (x_1^0, x_2^0);

2. Определим шаг h:
$\vec{x}^{(0)} = \binom{x_1^{(0)}+h}{x_2^{(0)}}$
Предположим, что в новой точке значение функционала уменьшилось (см. рис. 9.3). Тогда снова прибавим к координате $x_1^{(0)}$ h и получим $\vec{x}^{(2)} = \binom{x_1^{(0)}+2h}{x_2^{(0)}}$. Продолжаем операцию до тех пор, пока значение функционала не начнет увеличиваться. После этого считаем что достигнут минимум функционала по координате x_1;
3. Далее проводим аналогичные операции по координате x_2. Прибавляем к ней h, предположим, что значение функционала увеличилось, это означает, что выбрано неправильное направление и его надо изменить. После чего вычитаем из $x_2^{(0)}$ h. Продолжаем до тех пор, пока в какой то точке значение функционала не увеличится;
4. Теперь снова сдвигаем координату по оси x_1, пока не найдем минимум на этой оси;
5. Продолжаем итерации до момента, пока сдвиг координаты в любом направлении приведет к увеличению функционала.
Таким образом, метод определяет минимум функционала $f(\vec{x})$ с гарантированной погрешностью, равной h по каждой из координат. Для увеличения точности на шаге 5 можно уменьшить h и продолжить итерации.

Минусом метода является его долгая реализация, поэтому его использование рекомендуется только в случаях, когда вектор $\vec{x}$ содержит небольшое число компонент.

Нахождение многомерного минимума градиентным методом

Возможны случаи, когда функционал зависит от количества сеточных элементов, которое может достигать больших значений, при которых сходимость метода координатного спуска будет крайне медленной. Поэтому на практике чаще применяются градиентные методы.

Рассмотрим идею градиентного метода на двумерном примере:
1. Выберем произвольную точку $\vec{x}^{(0)}$
2. Найдем градиент ($f'(\vec{x^{(0)}})$) функционала, который указывает на направление наибольшего роста. Далее будем проводить минимизацию вдоль направления, противоположного направлению градиента.
3. Минимизацию можно проводить двумя способами:
1) $\vec{x^{(0)}} - h \cdot f'(\vec{x^{(0)}})$
$h > 0$ – параметр спуска.
Продолжаем движение по направлению обратному направлению градиента до тех пор, пока значение не увеличится (см. рис. 9.4) 2) Вместо того, чтобы искать точку, в которой функционал начнет расти с помощью последовательных итераций с фиксированным h,

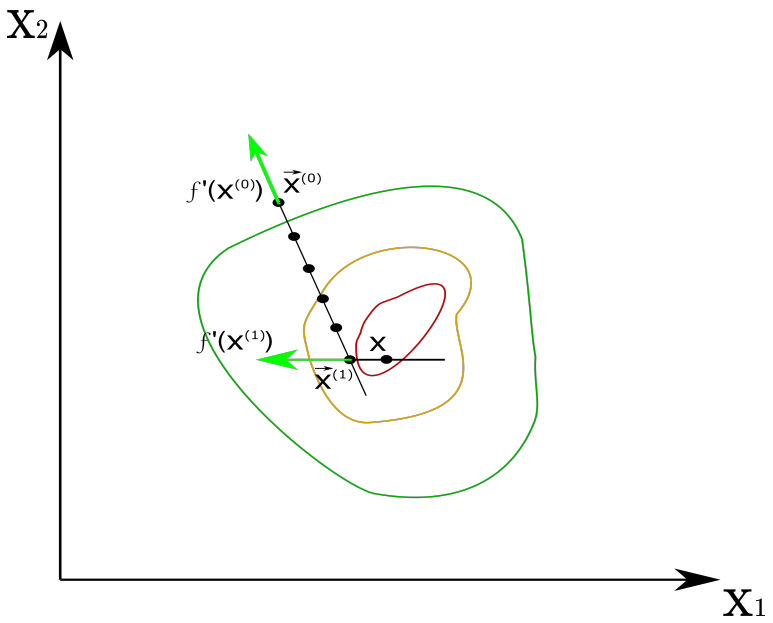


Рис. 9.4: Градиентный метод. Цветные линии – линии уровня функции. Точки – итерации.

можно найти оптимальный h, определяющий расстояние до этой точки, вычислив, например с помощью метода золотого сечения минимум следующего одномерного функционала:

$$\arg\min_{h\geq 0} f(x^{\vec{(n)}} - hf'(x^{\vec{(n)}}))$$

4. После того, как на выбранном направлении была найдена точка минимума, эта точка становится следующим приближением $x^{\vec{(1)}}$
5. Считаем $f'(x^{\vec{(0)}})$ и продолжаем итерации.

Градиентный метод сходится быстрее чем метод координатного спуска. Это связанно с тем, что градиентный метод является методом первого порядка, потому что задействованы производные первого порядка. Его недостатком является то, что необходимо вычислить первые производные функции $f(\vec{x})$.

Пример

Рассмотрим минимизацию квадратичного функционала

$$f(x) = \frac{1}{2}(Ax, x) + (b, x) + c$$

Матрица A является положительно определенной и симметричной. Такие функционалы чрезвычайно часто встречаются на практике.

Рассмотрим СЛАУ $Ax = b$, простейшим методом её решения является метод Гаусса. Предположим (см. рис. 9.5), что $v = ku + b$ и имеется набор точек (v_j, u_j), полученных из эксперимента, каждая из которых соответствует уравнению: $v_j = ku_j + b$. Чтобы найти

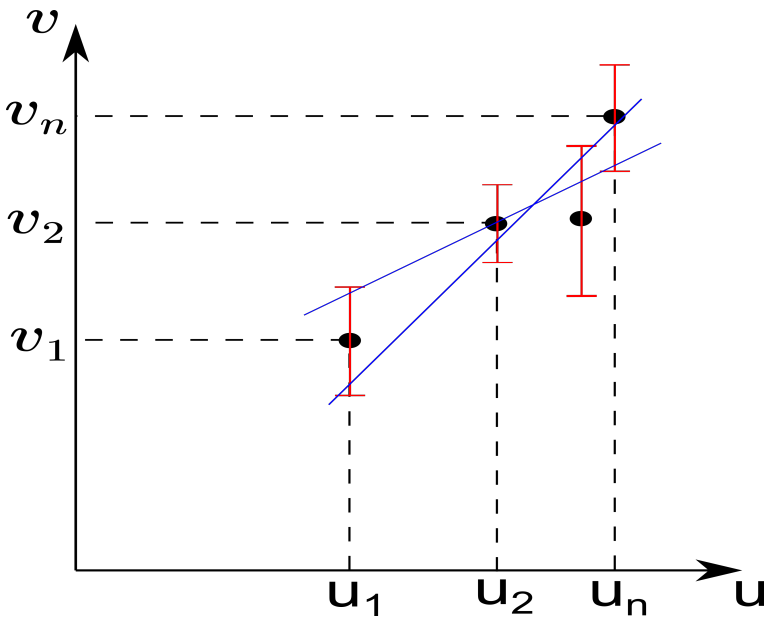


Рис. 9.5: Построение прямой через точки, заданные с ошибкой

два неизвестных коэффициента k и b достаточно двух уравнений, но, на практике (v_j, u_j) измерены с ошибкой, поэтому,тангенс угла наклона прямой, проведенной через две такие точки имеет большой разброс (см. рис. 9.5). По этой причине используется максимально большой набор точек (v_j, u_j). Таким образом получается система:

$$\begin{cases} v_1 = ku_1 + b \\ v_2 = ku_2 + b \\ \dots \\ v_n = ku_n + b \end{cases}$$

При этом система получается переопределенной. Эту систему можно переписать в матричном виде:

$$\begin{pmatrix} u_1 & 1 \\ u_2 & 1 \\ \dots & \dots \\ u_n & 1 \end{pmatrix} \begin{pmatrix} k \\ b \end{pmatrix} = \begin{pmatrix} v_1 \\ v_2 \\ \dots \\ v_n \end{pmatrix}$$

Где $A = \begin{pmatrix} u_1 & 1 \\ u_2 & 1 \\ \dots & \dots \\ u_n & 1 \end{pmatrix}, x = \begin{pmatrix} k \\ b \end{pmatrix}, b = \begin{pmatrix} v_1 \\ v_2 \\ \dots \\ v_n \end{pmatrix}$.
Метод Гаусса для такой системы работать не будет, поскольку он применим только для систем с квадратной матрицей. Более того, из-за того, что все измерения проведены с ошибкой, эта система не будет иметь классического решения, то есть, решение $x = (k, b)$ может не удовлетворять некоторым уравнения системы при подстановке.

В качестве решения этой системы можно искать минимум следующего квадратичного функционала:

$$\Phi(x) = ||Ax - b||^2 \to 0$$

Минимизируя этот функционал мы находим такой x, который наилучшим образом удовлетворяет данному уравнению $Ax - b \to \theta$.
Проведем преобразование:

$$\Phi(x) = (Ax - b, Ax - b) = (Ax, Ax) - 2(b, Ax) + (b, b) = (A^T Ax, x) + (2A^T b, x) + (b, b)$$

Пусть теперь:

$$\widetilde{A} = A^T A,\ \widetilde{b} = 2A^T b,\ \widetilde{c} = (b, b)$$

Получим:

$$\Phi(x) = (\widetilde{A}x, x) + 2(\widetilde{b}, x) + \widetilde{c}$$

Таким образом, к задаче минимизации функционала квадратичного вида сводится крайне распространенная прикладная задача по решению переопределенных СЛАУ $Ax = b$, полученных в эксперименте с ошибками. Причем, если матрица A является произвольной, то $\widetilde{A} = A^T A$ это положительно определенная и симметрической, при условии, что исходная матрица не вырождена.

Вернемся к минимизации функционала:

$$f(x) = \frac{1}{2}(Ax, x) + (b, x) + c$$

В курсе математического анализа градиент был равен:

$$f'(x) = \lim_{\Delta x \to 0} \frac{f(x + \Delta x) - f(x)}{\Delta x}$$

или

$$f(x + \Delta x) - f(x) = f'(x) \cdot \Delta x + o(\Delta x)$$

В математическом анализе оба варианта определения производной были эквивалентны.
В функциональном анализе эти производные называются слабой и сильной производной

соответственно.
Поэтому, будем искать производную в виде $f(x+h) - f(x) = (f^{'}(x), h) + o(||h||)$, где x и h являются векторами.

$$f(x+h) - f(x) = \frac{1}{2}(Ax + Ah, x + h) + (b, x + h) + \not{c} - \frac{1}{2}(Ax, x) - (b, x) - \not{c} =$$
$$= \frac{1}{\cancel{2}}\cancel{(Ax, x)} + \frac{1}{2}(Ax, h) + \frac{1}{2}(Ah, x) + \frac{1}{2}(h, h) + \cancel{(b, x)} + (b, h) - \frac{1}{\cancel{2}}\cancel{(Ax, x)} - \cancel{(b, x)} =$$
$$= (Ax + b, h) + \frac{1}{2}||h||^2$$

Где: $Ax+b = f^{'}(x),\ ||h||^2 = o(||h||)$. Таким образом можно найти производную функционала.

Выпишем задачу еще раз:

$$f(x) = \frac{1}{2}(Ax, x) + (b, x) + c$$
$$f^{'}(x) = Ax + b$$

Покажем как реализуется градиентный метод:
1. $x^{(0)}$ - любое, $n := 0$
2. $f^{'}(x^{(n)}) = Ax^{(n)} + b$
3. $x^{(n+1)} = x^{(n)} - \beta_n f^{'}(x^{(n)})$

Для квадратичного функционала оптимальное β_n можно найти аналитически, то есть найти β_n при котором значение функционала от выражения выше становится наименьшим:

$$\beta^* = \arg\min f(x^{(0)} - \beta_n f^{'}(x^{(0)}))$$

$$f(x^{(n)} - \beta_n f^{'}(x^{(n)})) =$$
$$= \frac{1}{2}(Ax^{(n)} - \beta_n A f^{'}(x^{(n)}), x^{(n)} - \beta_n f^{'}(x^{(n)})) + (b, x^{(n)} - \beta_n f^{'}(x^{(n)})) + c =$$
$$= \frac{1}{2}(Ax^{(n)}, x^{(n)}) - \underline{\underline{\frac{1}{2}(Ax^{(n)}, \beta_n f^{'}(x^{(n)}))}} - \underline{\underline{\frac{1}{2}(\beta_n A f^{'}(x^{(n)}), x^{(n)})}} +$$
$$+ \underline{\frac{1}{2}(\beta_n A f^{'}(x^{(n)}), \beta_n f^{'}(x^{(n)}))} + (b, x^{(n)}) - \underline{\underline{(b, \beta_n f^{'}(x^{(n)}))}} + c =$$
$$= \{\text{Это выражение является квадратичным относительно } \beta_n\} =$$
$$= \underline{\frac{1}{2}(Af^{'}(x^{(n)}), f^{'}(x^{(n)}))\beta_n^2} + \underline{\underline{(-(Af^{'}(x^{(n)}), x^{(n)}) - (f^{'}(x^{(n)}), b))\beta_n}} +$$
$$+ \frac{1}{2}(ax^{(n)}, x^{(n)}) + (b, x^{(n)}) + c$$
$$\beta_n = \frac{(Af^{'}(x^{(n)}), x^{(n)}) + (f^{'}(x^{(n)}), b)}{(Af^{'}(x^{(n)}), f^{'}(x^{(n)}))} = \frac{(Ax^{(n)} + b, f^{'}(x^{(n)}))}{(Af^{'}(x^{(n)}), f^{'}(x^{(n)}))}$$

Вспомним, что $Ax^{(n)} + b = f^{'}(x^{(n)})$, таким образом:

$$\beta_n = \frac{(f^{'}(x^{(n)}), f^{'}(x^{(n)}))}{(Af^{'}(x^{(n)}), f^{'}(x^{(n)}))}$$

4. $n = n + 1$, возвращаемся на шаг 2

Метод сопряженных градиентов

Ниже выписана схема метода:

$$Ax = b$$
$$p^{(0)} = 0$$
$$x^{(1)} - \text{любое}, n = 1$$
$$r^{(n)} = \begin{cases} A^T(Ax^{(n)} - b), \text{ если } n = 1 \\ r^{(n-1)} - \frac{q^{(n-1)}}{(p^{(n-1)}, q^{(n-1)})}, \text{ если } n \geq 2 \end{cases}$$
$$p^{(n)} = p^{(n-1)} + \frac{r^{(n)}}{(r^{(n)}, r^{(n)})}$$
$$q^{(n)} = A^T(Ap^{(n)})$$
$$x^{(n+1)} = x^{(n)} - \frac{p^{(n)}}{(p^{(n)}, q^{(n)})}, \ n = n + 1$$

$Ax = b$ это переопределенная СЛАУ, где правая часть в эксперименте измерена с ошибкой. Мы сопоставляем этой системе квадратичный функционал с положительно определенной симметрической матрицей $A^T A$, и последовательность, которая приводит нас к точному минимуму, строим по алгоритму, описанному выше. p и q это вспомогательные вектора, $p^{(0)}$ определяем нулем, то есть, это нулевой вектор, $x^{(1)}$ – произвольный вектор. Далее, на первой итерации ищем $r^{(1)} = A^T(Ax^{(n)} - b)$, при других n $r^{(n)} = r^{(n-1)} - \frac{q^{(n-1)}}{(p^{(n-1)}, q^{(n-1)})}$. Находим следующее приближение для вспомогательных векторов $p^{(n)} = p^{(n-1)} + \frac{r^{(n)}}{(r^{(n)}, r^{(n)})}$ и $q^{(n)} = A^T(Ap^{(n)})$. После чего находим следующее приближение вектора $x^{(n+1)} = x^{(n)} - \frac{p^{(n)}}{(p^{(n)}, q^{(n)})}$. В конце увеличиваем n на единицу и начинаем следующую итерацию.

Принципиальное отличие метода сопряженных градиентов от формул градиентной минимизации состоит в том, что в случае, если все вычисления производятся точно, то $x^{(N)}$ это точное решение, где N это размерность пространства, но, из-за ошибок машинного округления за N шагов возможно найти только решение лежащее очень близко к точному.

Рассмотрим еще одно преимущество метода сопряженных градиентов. Предположим, что в СЛАУ $Ax = b$ вектор правой части задан точно, для решение этой системы метод Гаусса требует $O(N^3)$ операций. Метод сопряженных градиентов, в свою очередь, так же требует $O(N^3)$ операций, но, на практике, метод Гаусса заканчивает работу только после выполнения всех $O(N^3)$ операций, из-за ошибок машинного округления даже хорошо

обусловленная матрица в процессе счета может стать плохо обусловленной. Градиентный метод же не накапливает ошибку в процессе счета, поскольку когда он с ошибкой находит точку минимума по какому либо направлению, расчет следующего направления позволяет её скорректировать, поскольку вычисляется уже для другой точки (см. рис.9.6). И при приближении к точному решению процесс можно прервать сильно раньше чем за $O(N^3)$ операций.

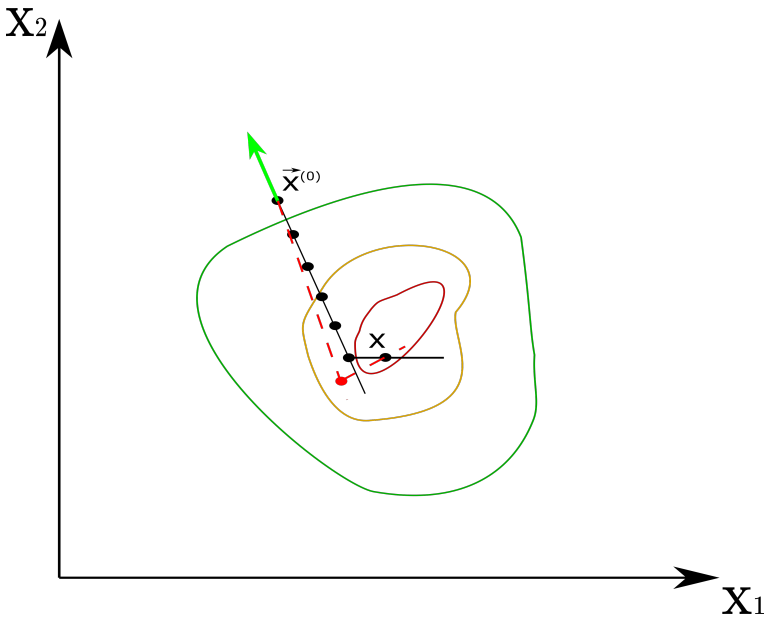


Рис. 9.6: Компенсация ошибки вычислений при использовании метода сопряжённых градиентов

Лекция 10

Алгебраическая проблема поиска собственных значений

В лекции 2, при рассмотрении вопроса решения СЛАУ $Ax = b$, было упомянуто, что одна из возникающих подзадач, это поиск собственных значений матрицы A.
Постановка задачи и поиске собственных значений и собственных векторов матрицы выглядит следущим образом:

$$Ay = \lambda y$$

Требуется найти такие числа λ, при которых существует нетривиальное решение этой системы и найти соответствующие решения. Тогда эти λ будут называться собственными значениями, а соответствующие им нетривиальные решения – собственными векторами.
Собственные значения матрицы A находятся следующим образом:

$$(A - \lambda E)y = \theta$$

Из курса линейной алгебры известно, что однородная система алгебраических уравнений будет иметь нетривиальные решения в том случае, если определитель матрицы системы равен нулю:

$$\det(A - \lambda y) = 0$$

Перейдём к вопросу о численном нахождении собственных значений. Если квадратная матрица A имеет размерность N соответствующий ей определитель $\det(A-\lambda y)$ превратится в полином порядка N. Таким образом, если N большое, то возникнут проблемы численного характера по поиску соответствующих корней полинома порядка N.
Приведём пример, обосновывающий утверждение того, что проблема поиска собственных

значений неустойчива. Рассмотрим матрицу A, называемую Жорданова Клетка:

$$A = \begin{pmatrix} 1 & 1 & 0 & \dots & \dots & 0 \\ 0 & 1 & 1 & 0 & \dots & \vdots \\ \vdots & 0 & \ddots & \ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \dots & \dots & 0 & 1 & 1 \\ 0 & \dots & \dots & \dots & 0 & 1 \end{pmatrix}$$

По виду матрицы сразу можно сделать вывод, что собственных значений N штук и все они равны единице. Предположим что в одном из элементов матрицы есть ошибка $\varepsilon \approx 10^{-16}$:

$$A = \begin{pmatrix} 1 & 1 & 0 & \dots & \dots & 0 \\ 0 & 1 & 1 & 0 & \dots & \vdots \\ \vdots & 0 & \ddots & \ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & \dots & 0 & 1 & 1 \\ \varepsilon & \dots & \dots & \dots & 0 & 1 \end{pmatrix}$$

Запишем характеристическое уравнение:

$$\det(A - \lambda E) = 0$$

$$\begin{vmatrix} 1-\lambda & 1 & 0 & \dots & \dots & 0 \\ 0 & 1-\lambda & 1 & 0 & \dots & \vdots \\ \vdots & 0 & \ddots & \ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & \dots & 0 & 1-\lambda & 1 \\ \varepsilon & \dots & \dots & \dots & 0 & 1-\lambda \end{vmatrix} = 0$$

Разложим определитель по первому столбцу:

$$(1-\lambda)(-1)^{1+1} \cdot \begin{vmatrix} 1-\lambda & 1 & \dots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ \vdots & \dots & 1-\lambda & 1 \\ 0 & \dots & \dots & 1-\lambda \end{vmatrix} + \varepsilon(-1)^{N+1} \cdot \begin{vmatrix} 1 & 0 & \dots & 0 \\ 1-\lambda & 1 & 0 & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 1-\lambda & 1 \end{vmatrix} = 0$$

$$(1-\lambda)(1-\lambda)^{N-1} \pm \varepsilon \cdot 1 = 0 \Rightarrow$$

$$\Rightarrow \lambda = 1 - \sqrt[N]{\pm\varepsilon}, \text{ где корни понимаются в комплексном смысле}$$

Рассмотрим это выражение внимательней. Мы знаем, что собственные значения у матрицы N штук и все они равны единице, то есть собственное значение равное единице кратности N. И если присутствует небольшая ошибка, то $\lambda = 1 - \sqrt[N]{\pm\varepsilon}$, и если например у нас $N = 5$, то $\sqrt[5]{10^{-16}} = 10^{-3}$, то есть ошибка порядка 0.1%, если размерность матрицы $N = 16$, то ошибка становится сопоставимой с самим решением.
Поэтому процесс поиска собственных значений неустойчив по отношению к ошибкам в элементах матрицы, которые могут возникать в том числе из-за технических особенностей вычисления на компьютерах.

В связи с этим разделяют две задачи. Одна задача называется полной задачей на собственные значения, которая предполагает найти все собственные значения матрицы A. И есть частичная задача на собственные значения, когда ставится задача найти не все собственные значения, а только например несколько минимальных, либо например минимальное и максимальное. Полная задача, как было показано выше, неустойчива в случае матриц большой размерности. Но, на практике, в задачах, где необходимо искать собственные значения, все собственные значения знать не необходимо. Обычно требуется знать либо минимальное и/или максимальное собственные значения, либо нужно знать несколько минимальных собственных значений, которые определяют характер решения в соответствующей задаче.

Метод прямой итерации

Построим итерационный процесс по следующей формуле:

$$y^{s+1} = Ay^s,\ y^{(0)} - \text{любое } (\neq \theta)$$

Будем предполагать, что в пространстве, в котором определены используемые вектора, существует базис, построенный из собственных векторов матрицы A. Это означает, что:

$$y^{(s)} = \sum_{n=1}^{N} \alpha_n y_n,\ y_n - \text{собственные вектора.}$$

Тогда:

$$y^{(s+1)} = Ay^{(s)} = A\sum_{n=1}^{N} \alpha_n y_n = \sum_{n=1}^{N} Ay_n = \sum_{n=1}^{N} \alpha_n \lambda_n y_n$$

Представим, что данный итерационный процесс выполняется множество раз. Таким образом, после каждой итерации остаются те же компоненты $\alpha_n y_n$ домноженные на λ_n. Пусть λ_1 это наименьшее собственное значение и λ_N – наибольшее собственное значение. Это означает, что если $s \to \infty$, то в пределе y^s и y^{s+1} становятся сонаправленными, поскольку

на каждой итерации множитель стоящий у y_n растёт и при $s \to \infty$ компонента y_N соответствующая наибольшему собственному значению λ_N является доминирующей.
Поэтому при больших s:

$$y^{(s+1)} = Ay^{(s)} = A\alpha_N y_N = \alpha_N \lambda_N y_N = \lambda_N y^{(s)}$$
$$y^{(s+1)} = \lambda_N y^{(s)}$$

Теперь рассмотрим вопрос как из полученного выражения найти λ_N. Для этого сделаем следующее преобразование, домножив выражение на вектор $y^{(s+1)}$:

$$(y^{(s+1)}, y^{(s+1)}) = \lambda_N (y^{(s)}, y^{(s+1)}) \Rightarrow$$
$$\Rightarrow \lambda_N = \frac{(y^{(s)}, y^{(s+1)})}{(y^{(s+1)}, y^{(s+1)})}$$

Таким образом, на каждой итерации считается новое приближение $\lambda_N^{(s)}$.
Последовательные итерации для алгоритма поиска максимального собственного значения выглядят следущим образом:

$$y^{(s+1)} = Ay^{(s)}, \ \forall y^{(0)} \neq \theta$$
$$\lambda_N^{(s)} = \frac{(y^{(s)}, y^{(s+1)})}{(y^{(s+1)}, y^{(s+1)})}$$
$$\lambda_N^{(s)} \xrightarrow{s \to \infty} \lambda_{max}$$

Метод обратной итерации

Теперь рассмотрим поиск минимального собственного значения. Построим итерационный процесс следующим образом:

$$Ay^{(s+1)} = y^{(s)}$$
$$y^{(s+1)} = A^{-1} y^{(s)}$$

При запуске такого итерационного процесса, как было показано в методе для поиска максимального собственного значения, будет найдено максимальное собственное значение для матрицы A^{-1}, а собственные значения обратной матрицы связанны с собственными значениями самой матрицы обратным соотношением. Поэтому такой итерационный процесс будет сходиться к обратному минимальному собственному значению матрицы A и:

$$\lambda^{(s)} = \frac{1}{\frac{(y^{(s)}, y^{(s)})}{(y^{(s+1)}, y^{(s+1)})}} \xrightarrow{s \to \infty} \lambda_{min}$$

Обратим внимание на трудоёмкость. В методе поиска максимального собственного значения получение $y^{(s+1)}$ это умножение матрицы на вектор, что требует количества операция N^2.
В методе поиска минимального собственного значения нахождение $y^{(s+1)}$ это решение СЛАУ, для метода Гаусса это составляет N^3 операций.

Пример поиска минимального и максимального собственного значения

Рассмотрим задачу Штурма-Лиувилля:

$$\begin{cases} u_{xx} - 9xu_x + \lambda u = 0,\ x \in (0,1) \\ u(0) = 0,\ u(1) = 0 \end{cases}$$

Нужно определить те λ, при которых краевая задача нетривиальное решение. Найдём минимальное собственное значение в данной задаче.
Сведём задачу к постановке вида: $Ay = \lambda y$. Для этого на отрезке (0, 1) введём равномерную сетку с шагом $h = \frac{1-0}{N}$ и узлы сетки обозначим как $x_n = 0 + h \cdot n,\ n = \overline{0, N}$. Запишем аппроксимацию этого уравнения по сетке:

$$\begin{cases} \frac{u_N n+1-2u_n+u_{n-1}}{h^2} - 9x_n \frac{u_{n+1}-u_{n-1}}{2h} + \lambda u_n = 0,\ n = \overline{1, N-1} \\ u_0 = 0,\ u_N = 0 \end{cases}$$

Эту систему можно переписать как:

$$\begin{cases} \frac{u_2-2u_1+0}{h^2} - 9x_1 \frac{u_2-0}{2h} + \lambda u_1 = 0,\ n = 1 \\ \frac{u_{n+1}-2u_n+u_{n-1}}{h^2} - 9x_n \frac{u_{n+1}-u_{n-1}}{2h} + \lambda u_n = 0,\ n = \overline{2, N-2} \\ \frac{0-2u_{N-1}+u_{N-2}}{h^2} - 9x_{N-1} \frac{0-u_{N-2}}{2h} + \lambda u_{N-1} = 0,\ n = N-1 \end{cases}$$

Сведём эту систему к виду $Ay = \lambda y$:

$$\begin{pmatrix} \frac{2}{h^2} & \frac{-1}{h^2} + \frac{9x_1}{2h} & 0 & \dots & 0 \\ \frac{1}{h^2} - \frac{9x_2}{2h} & \frac{2}{h^2} & -\frac{1}{h^2} + \frac{9x_2}{2h} & \ddots & 0 \\ 0 & \frac{1}{h^2} - \frac{9x_3}{2h} & \frac{2}{h^2} & -\frac{1}{h^2} + \frac{9x_3}{2h} & \ddots \\ \vdots & \ddots & \ddots & \ddots & \ddots \end{pmatrix} \cdot \begin{pmatrix} u_1 \\ u_2 \\ \dots \\ u_N - 1 \end{pmatrix} = \lambda \begin{pmatrix} u_1 \\ u_2 \\ \dots \\ u_N - 1 \end{pmatrix}$$

Для начала итераций:
1. Задаём произвольный вектор $y^{(0)}$
2. Решаем СЛАУ $Ay^{(s+1)} = y^{(s)},\ s = 0, 1, \dots$
3. Вычисляем $\lambda^{(s)} = \frac{1}{\frac{(y^{(s)}, y^{(s)})}{(y^{(s+1)}, y^{(s+1)})}}$

Обратим внимание на то, что, как известно из курса дифференциальных уравнений, приведённая выше задача обладает бесконечным числом собственных значений (т.е. не имеет максимального собственного значения). Мы же, введя сетку с N интервалами, был произведён переход от задачи в бесконечномерной постановке, мы перешли к конечномерному пространству. Поэтому, полученная матрица A является конечномерной аппроксимацией соответствующего дифференциального оператора действующего в бесконечномерном пространстве. По этой причине, решив численно эту задачу на собственные значения, на самом деле, мы найдём

минимальные собственные значения не исходной задачи, а матрицы A (и то, при предположении, что мы сделаем бесконечное число итераций). Из-за этого полученные собственные значения сходятся к истинным лишь при $N \to \infty$

Программа доступна в материалах к лекциям под номером 10-1, а так же была подробно разобрана в лекции номер 10.

В результате работы программы был получен следующий набор собственных значений:

$$\lambda_1 = 9.34695715$$
$$\lambda_2 = 10.09634356$$
$$\lambda_3 = 10.46763797$$
$$\lambda_4 = 10.58381322$$
$$\lambda_5 = 10.61639286$$
$$\lambda_6 = 10.62512865$$
$$\lambda_7 = 10.62742393$$
$$\lambda_8 = 10.62802135$$
$$\lambda_9 = 10.62817616$$
$$\lambda_{10} = 10.62821620$$

Видно, что метод сходится достаточно быстро к значению 10.62.

Определить оптимальное количество операций невозможно, поэтому итерации следует останавливать после того, как требуемая точность будет достигнута.

Для определения точности можно воспользоваться формулами Ричардсона. Для этого:
1. Задали желаемое число итераций метода обратной итерации, при котором мы считаем, что собственные значения матрицы A находятся достаточно точно
2. Разбиваем отрезок на N интервалов и решаем на нём задачу поиска минимального собственного значения
3. После этого, сгущаем сетку в r раз и находим собственные значения для матрицы оператора на этой сетке
4. После этого можем применить формулу Рунге-Ромберга (подробнее про метод Ричардсона см. Лекцию 4 и Лекцию 8)
Этот процесс можно повторять рекуррентно и по формуле Рунге-Ромберга получать очередные уточнения с более высоким порядком точности.

Программа доступна в материалах к лекциям под номером 10-2, а так же была подробно разобрана в лекции номер 10.

В результате работы программы с уточнением по ричардсону получены:

Таблица приближённых собственных значений λ:

	$p = 2$	$p = 4$	$p = 6$	$p = 8$	$p = 10$
$s = 0$	10.6282				
$s = 1$	10.6525	10.6606			
$s = 2$	10.6585	10.6605	10.6605		
$s = 3$	10.6600	10.6605	10.6605	10.6605	
$s = 4$	10.6604	10.6605	10.6605	10.6605	10.6605

Таблица оценок ошибок:

	$p = 2$	$p = 4$	$p = 6$	$p = 8$	$p = 10$
$s = 0$					
$s = 1$	0.0081				
$s = 2$	0.0020	-0.0000			
$s = 3$	0.0005	-0.0000	-0.0000		
$s = 4$	0.0001	-0.0000	-0.0000	-0.0000	

Таблица эффективных порядков точности:

	$p = 2$	$p = 4$	$p = 6$	$p = 8$	$p = 10$
$s = 0$					
$s = 1$					
$s = 2$	2.0041				
$s = 3$	2.0010	4.0197			
$s = 4$	2.0003	4.0049	6.0096		

Таблица эффективных порядков точности показывает, что порядок точности метода равен двум и что ошибка метода раскладывается в ряд Тейлора только по чётным степеням ($q = 2$).

Для задачи Штурма-Лиувиля Задача поиска максимального собственного значения не имеет смысла, поскольку максимального собственного значения этой задачи не существует. Если же запустить метод прямой итерации для этой задачи, то было бы найдено λ_{max} матрицы для любого N, что является следствием перехода из бесконечномерного пространства в конечномерное. Подобную ошибку можно определить по таблице эффективных порядков точности, в этому случае эффективные порядки точности окажутся меньше либо равными нулю.

Метод обратных итераций со сдвигом

Предположим, что есть априорная информация, которая является хорошим начальным приближением для какого то собственного значения λ. Тогда, построим следущий итерационный процесс:

$$(A - \lambda E)y^{(s+1)} = y^{(s)}$$

Если у матрицы A набор собственных значений λ_n, то у матрицы $(A - \lambda E)$ набор собственных значений это $(\lambda_n - \lambda)$

Поэтому, после запуска этого итерационно процесса, вычислить $\frac{(y^{(s+1)}, y^{(s+1)})}{(y^{(s)}, y^{(s+1)})}$, то мы знаем, что это отношение скалярных произведений сходится к максимальному собственному значению матрицы $(A - \lambda E)^{-1}$ равному $\frac{1}{(\lambda_n - \lambda)}$, то есть когда $(\lambda_n$ как можно ближе к λ. Поэтому, в этом случае итерационный процесс даёт возможность найти то λ_n, которое ближе к λ. Таким образом:

$$\frac{(y^{(s+1)}, y^{(s+1)})}{(y^{(s)}, y^{(s+1)})} = \frac{1}{\lambda_n^{s+1} - \lambda}$$

$$\lambda^{(s+1)} = \lambda + \frac{(y^{(s)}, y^{(s+1)})}{(y^{(s+1)}, y^{(s+1)})}$$

Проведём ещё одну модификацию:

$$(A - \lambda^{(s)} E)y^{(s+1)} = y^{(s)}$$

И будем считать, что $\lambda^{(0)}$ – хорошее начальное приближение для искомого λ, $y^{(0)}$ – любое ($\neq \theta$). Получаем, что:

$$\lambda^{(s+1)} = \lambda^{(s)} + \frac{(y^{(s)}, y^{(s+1)})}{(y^{(s+1)}, y^{(s+1)})} \frac{(y^{(s+1)}, y^{(s+1)})}{(y^{(s)}, y^{(s+1)})} = \frac{1}{\lambda_n^{s+1} - \lambda^{(s)}}$$

Итерационный процесс для метода обратной итерации со сдвигом выглядит следующим образом:

1. Зная $\lambda^{(0)}$ находим $y^{(1)}$ по решая СЛАУ:

$$(A - \lambda^{(s)} E)y^{(1)} = y^{(0)}$$

2. Находим $\lambda^{(1)}$ по формуле:

$$\lambda^{(s+1)} = \lambda^{(s)} + \frac{(y^{(s)}, y^{(s+1)})}{(y^{(s+1)}, y^{(s+1)})}$$

3. Возвращаемся к шагу 1, используя полученное приближение для λ.

Если не имеется никакого хорошего начального приближения для λ, то если мы задали $\lambda^{(0)}$ каким то числом, то этот метод сойдётся к тому собственному значению, которое ближе всего к этому $\lambda^{(0)}$.

ФИЗИЧЕСКИЙ
ФАКУЛЬТЕТ
МГУ ИМЕНИ
М.В. ЛОМОНОСОВА

teach-in

ЛЕКЦИИ УЧЕНЫХ МГУ