Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Владимирский государственный университет
имени Александра Григорьевича и Николая Григорьевича Столетовых»

В. Н. ЛОБКО

МАТЕМАТИЧЕСКИЕ МЕТОДЫ В ХИМИИ И ХИМИЧЕСКОЙ ТЕХНОЛОГИИ

Численные методы решения алгебраических задач и обработки функций

Учебное пособие

Владимир 2019

УДК 519.61:54
ББК 22.193
Л68

Рецензенты:
Кандидат физико-математических наук, доцент
доцент кафедры физики и прикладной математики
Владимирского государственного университета
имени Александра Григорьевича и Николая Григорьевича Столетовых
А. Ю. Лексин

Кандидат химических наук
старший научный сотрудник Федерального государственного
бюджетного учреждения Всероссийского научно-исследовательского
института защиты животных
(Федерального центра охраны здоровья животных)
Д. С. Большаков

Издаётся по решению редакционно-издательского совета ВлГУ

Лобко, В. Н.

Л68 Математические методы в химии и химической технологии. Численные методы решения алгебраических задач и обработки функций : учеб. пособие / В. Н. Лобко ; Владим. гос. ун-т им. А. Г. и Н. Г. Столетовых. – Владимир : Изд-во ВлГУ, 2019. – 144 с.
ISBN 978-5-9984-1033-8

Учебное пособие посвящено использованию средств программирования и численных методов для решения вычислительных задач в химии и химических науках в соответствии с программой вуза. Рассмотрены методы решения нелинейных уравнений, представлены средства обработки экспериментальных данных (аппроксимация и интерполяция), методы решения систем уравнений, численные методы взятия интегралов.

Предназначено для студентов химических и химико-технологических специальностей 04.03.01 – Химия, 18.03.01 – Химическая технология.

Рекомендовано для формирования профессиональных компетенций в соответствии с ФГОС ВО.

Табл. 9. Ил. 52. Библиогр.: 11 назв.

УДК 519.61:54
ББК 22.193

ISBN 978-5-9984-1033-8

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

В химии, физической химии и химической технологии широко используются математические методы для проведения теоретических и технологических расчётов. В последнее время некоторыми научными коллективами, работающими в области химических наук, составлялись комплексы компьютерных программ для решения целого класса специфических задач.

При этом использование средств только высшей математики не позволяет успешно решать многие вычислительные задачи, такие как взятие интегралов, решение дифференциальных уравнений, обработка результатов экспериментов и др. Все эти проблемы легко разрешимы методами прикладной математики, т. е. использованием численных методов. Почти все эти методы приближённые и могут быть реализованы только на компьютерах с использованием языков программирования высокого уровня как средств решения вычислительных задач.

В предыдущем пособии «Математические методы в химии и химической технологии. Основы программирования вычислительных задач» освещались вопросы теории программирования и изложены основы языка программирования высокого уровня Pascal – базового для таких современных языков, как Delphi, Lasarus и пр. Выбор языка был обусловлен его высоким профессиональным статусом и сравнительной лёгкостью в освоении.

В предлагаемом пособии представлены методы решения нелинейных уравнений, которые часто встречаются в теории химических наук; методы решения систем уравнений как задача вспомогательного характера, к которой сводится применение многих вычислительных методик; численные методы взятия интегралов как часто встречающаяся при расчётах в химии и химической технологии проблема.

В следующем учебном пособии, готовящемся к изданию, будут рассмотрены другие вычислительные задачи и методы их решения в химии, физической химии и химической технологии, а также основы моделирования состояний и процессов в этих областях.

Поскольку профессиональные математики и программисты всегда значительно хуже химиков понимают ту или иную конкретную задачу, нельзя полностью полагаться на них при решении математических задач в химической области из-за риска возникновения трудновыявляемых ошибок. С другой стороны – высокопрофессиональный химик, освоивший на необходимом уровне университетский курс численных методов, может вполне самостоятельно решать специфические задачи химии или контролировать решения профессиональных математиков и программистов.

Профессионализм химиков при этом должен проявляться в умении чётко сформулировать задачу, наметить способы её решения, подобрать тот или иной численный метод со знанием дела, осуществить тем или иным способом решение на компьютере, выявить и исправить возможные ошибки.

Глава 1. РЕШЕНИЕ НЕЛИНЕЙНЫХ УРАВНЕНИЙ

§ 1. Введение

При решении химико-технологических задач, связанных, в частности, с термодинамикой и химической кинетикой, в теоретических расчётах при обработке экспериментальных данных очень часто бывает необходимо решать нелинейные уравнения для нахождения того или иного параметра. При решении дифференциальных уравнений во многих областях химии, физической химии и химической технологии получаются функции, которые могут представлять собой нелинейные уравнения. При этом параметры и константы уравнения становятся, как правило, известными, но требуется возможность вычисления функции при задаваемых значениях аргумента.

В теоретической математике известны способы аналитического решения линейных уравнений, квадратного трёхчлена, некоторые способы решения уравнения с кубической параболой, а также решения некоторых других уравнений для малого числа частных случаев.

Таким образом, подавляющее большинство нелинейных уравнений может быть решено только численными методами с использованием компьютерных вычислений.

§ 2. Классификация уравнений и их решение. Особенности численных решений

Нелинейные уравнения с одним неизвестным, решение которых будет рассмотрено, имеют общий вид

$$\boxed{f(x)=0}, \qquad (1)$$

где $f(x)$ – нелинейная функция аргумента x. При решении уравнений необходимо соблюдать этот канонический вид и приводить к нему рассматриваемые уравнения (справа должен быть 0), так как в конкретных методах работа проводится с полученной после этого функцией $f(x)$.

Например, заданное уравнение

$$5\sin x - 4x^3 = \ln(2x) + 3x - 8$$

перед тем, как его решать, нужно привести к виду

$$5\sin x - \ln(2x) - 4x^3 - 3x + 8 = 0\,.$$

При этом функция $f(x)$ из (1) представляет собой

$$f(x) = 5\sin x - \ln(2x) - 4x^3 - 3x + 8,$$

с ней-то и производятся все действия.

Нелинейные функции вида $f(x)$ подразделяются на алгебраические и трансцендентные (соответственно этому **уравнения** бывают **алгебраические и трансцендентные**). Алгебраические функции имеют каноническую форму полинома степени n:

$$\boxed{f(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \ldots + a_n x^n}, \qquad (2)$$

где a_i – известные коэффициенты. К такому виду приводятся и некоторые другие функции, в частности – рациональные дроби. Трансцендентные функции (показательные, логарифмические, тригонометрические и др.) нельзя точно привести к каноническому виду. Такое деление нелинейных функций представляет собой математические нюансы и не имеет значения при решении уравнений численными методами.

Решить уравнение (1) означает **найти его корни**. **Корень уравнения – это число, при подстановке которого в уравнение оно превращается в тождество**. Будем обозначать корень уравнения x^*. При этом из формы уравнения (1) следует, что при $x = x^*$ функция обращается в нуль, т. е. график функции должен пересекать ось абсцисс (рис. 1).

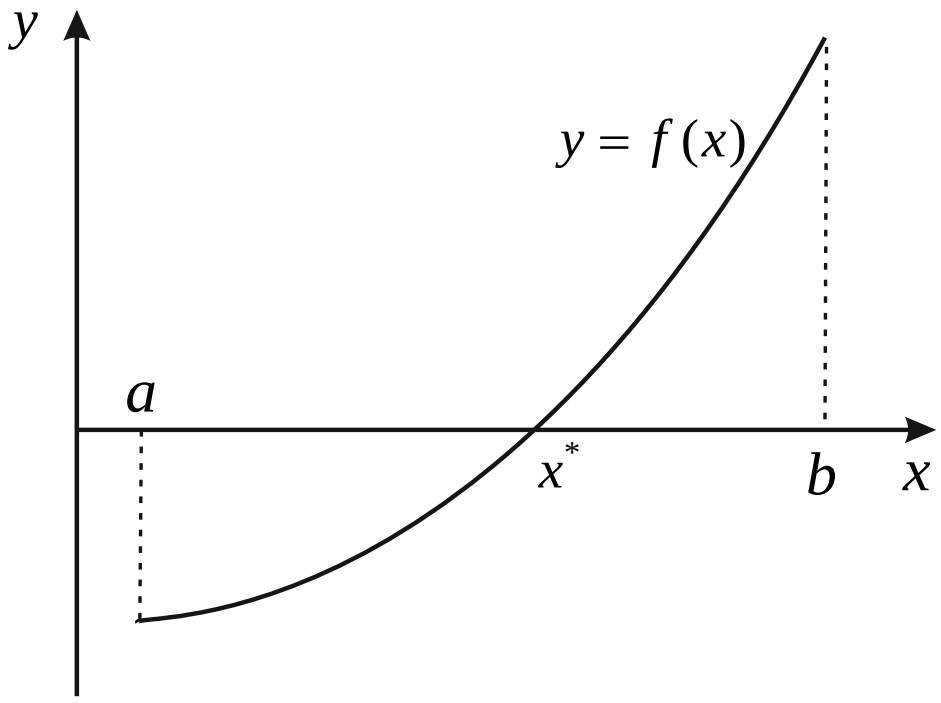

Рис. 1. Функция $y = f(x)$, имеющая на отрезке $[a, b]$ корень x^*

Как уже отмечалось, нелинейные уравнения можно решить аналитически лишь в некоторых, немногих, случаях. С помощью численных же методов можно решить практически любые уравнения, причём существует довольно много этих методов и всегда можно подобрать наиболее оптимальный или перепроверить решение несколькими методами. При этом решения в отличие от аналитических методов получаются приближёнными, однако всегда можно задать требуемую точность решения, что сводит на нет этот недостаток.

Решение уравнений численными методами проходит в **два этапа**. **Первый – отделение корней.** На числовой оси выделяются достаточно узкие интервалы, содержащие корни. **Второй – уточнение корней.** В найденных интервалах конкретными методами отыскиваются корни с заданной точностью.

§ 3. Этап отделения корней

Абсолютно универсальных методов отделения корней, особенно для сложных функций, не существует. Всегда имеет место опасность пропустить тот или иной корень. Можно быть уверенным только в отношении элементарных функций, поведение которых хорошо известно: например, тригонометрические, показательные, логарифмические и другие функции.

Самый простой, но и самый трудоёмкий – графический метод, т. е. построение графика функции $y = f(x)$. В настоящее время при наличии компьютерных программ построения графиков это не представляет собой сложной задачи. Однако масштаб построения должен быть достаточно мелким, а интервал по оси x – достаточно большим, а это – два противоречивых требования. Лучше всего сначала построить график в широком интервале, а затем – в «подозрительных» местах, в более мелком масштабе.

Часто эту задачу можно упростить, разбив заданную функцию $f(x)$ на две более простые (или даже элементарные) функции ($f_1(x)$ и $f_2(x)$), т. е. сделав преобразование

$$f(x) = 0 \quad \Rightarrow \quad f_1(x) = f_2(x),$$

а затем построив графики функций

$y_1 = f_1(x)$ и $y_2 = f_2(x)$.

Если функция $y = f(x)$ имеет корень, то функции y_1 и y_2 будут пересекаться в точке, абсцисса которой будет совпадать с искомым корнем.

Пример

Если задано уравнение

$$0.5\exp(0.6x) + 1 - 3/x = 0,$$

то функцию слева можно разбить на две функции

$$y_1 = 0.5\exp(0.6x) + 1 \text{ и } y_2 = 3/x$$

и построить их графики (что довольно легко сделать по характеристическим точкам с пересчётом). Эти графики будут пересекаться в точке, абсцисса которой даст корень x^* (рис. 2).

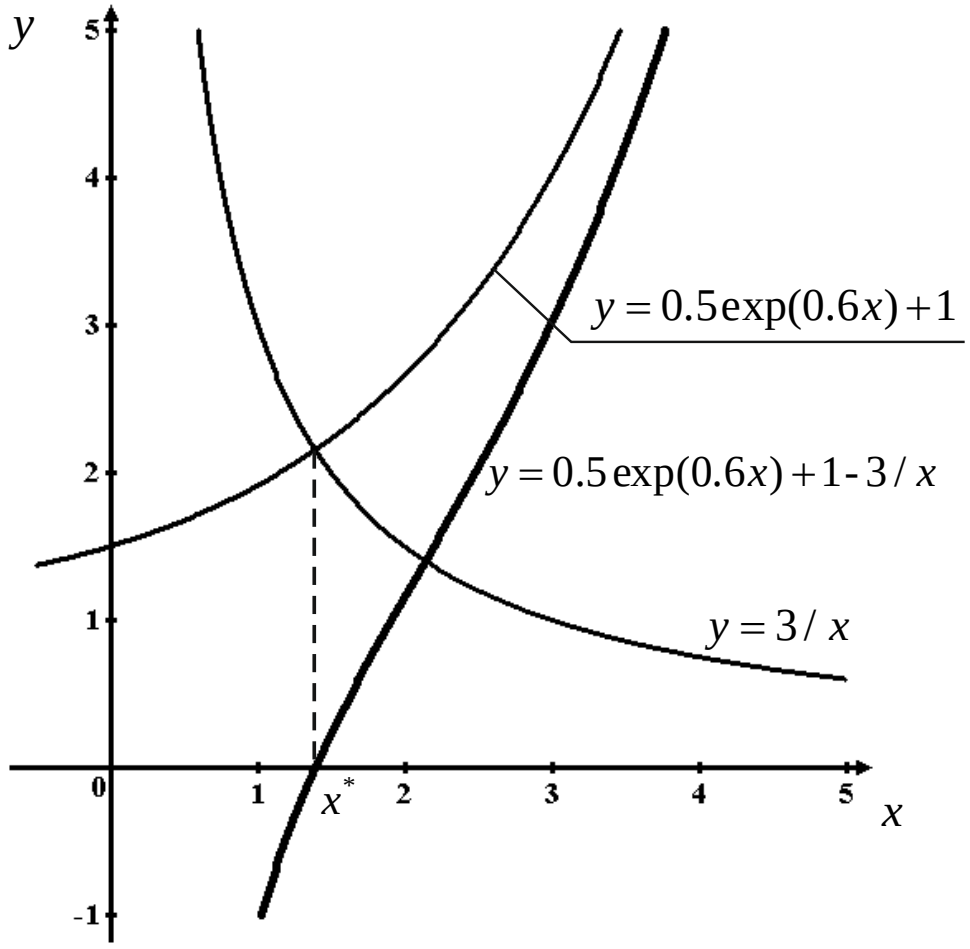

Рис. 2. Функция $y = 0.5\exp(0.6x) + 1 - 3/x$ может быть разбита на функции $y = 0.5\exp(0.6x) + 1$ и $y = 3/x$, которые будут иметь точку пересечения, если x^* – корень уравнения $0.5\exp(0.6x) + 1 - 3/x = 0$

В результате использования того или иного графического метода можно отделить один или несколько корней с большой погрешностью: графики имеют точность максимум 2 – 3 значащие цифры. В качестве рабочих интервалов для уточнения корней выбирают интервалы с некоторым запасом, чтобы точно быть уверенным, что искомые корни будут в них входить. За концы интервалов принимают чаще всего узлы масштабной сетки или какие-то целые числа. Например, по рис. 2 можно выбрать интервал [1, 2].

Существуют также некоторые аналитические способы отделения корней.

Результатом отделения корней должны быть чётко обозначенные интервалы; если уравнение имеет несколько корней, соответствующие интервалы не должны пересекаться или быть смежными, иначе методы уточнения корней могут работать некорректно.

§ 4. Уточнение корней. Итерационный процесс. Теорема Вейерштрасса

Практически все методы уточнения корней являются **итерационными методами**, т. е. принадлежат к широкому классу численных методов вообще. **Итерация** – это **последовательное приближение** к чему-либо, в данном случае – к решению уравнения. В дальнейшем мы часто будем встречать итерационные методы в других областях вычислительной математики.

Теоретически итерационный процесс бесконечен, поэтому всегда должна быть задана **точность**, с которой необходимо получить решение и по достижении которой нужно прекращать итерации.

Чтобы реализовать итерационный процесс, необходимы четыре фактора: **итерационная формула**, **критерий сходимости**, **начальное приближение** и **критерий окончания**.

1. Итерационная формула. Она выводится из теоретических соображений; её наличие делает возможным сам итерационный процесс.

2. Критерий сходимости. Строго доказанное условие, гарантирующее **сходимость** итерационного процесса, т. е. его последовательное приближение к решению. Итерации могут и расходиться (удаляться от решения).

3. Начальное приближение. Позволяет **начать** итерационный процесс, так как **последующее приближение к решению** строится только на основе **предыдущего приближения** и необходимо с чего-то начинать.

4. Критерий окончания. Условие, позволяющее **завершить** итерационный процесс по достижении требуемой точности.

Практическая реализация итерационных методов в конкретных языках программирования осуществляется с помощью итерационных циклов. В языке Pascal и современных языках на его основе – Delphi и Lasarus – таковыми являются операторы циклов **repeat** и **while**.

Во многих методах решения нелинейных уравнений для построения итерационного процесса, т. е. в качестве итерационной формулы и критерия сходимости используется теорема Вейерштрасса.

Теорема Вейерштрасса. *Если на концах некоторого отрезка непрерывная функция* $f(x)$ *принимает значения разных знаков, то на этом отрезке уравнение* $f(x)=0$ *имеет хотя бы один корень. Если* $f(x)$ *имеет первую производную* $f'(x)$*, не меняющую на этом отрезке знак, то существует только один корень уравнения.*

Для дальнейшего изложения необходимо напомнить, что функция $f'(x)$, производная от функции $f(x)$, или просто **производная**, определена (если она вообще существует) в конкретной точке x и численно равна тангенсу угла наклона касательной, проведённой к функции $f(x)$ в точке x (рис. 3, 4, 5). Знак производной также определяется знаком тангенса угла наклона α (этот угол откладывается от горизонтали против часовой стрелки). Сама функция $f'(x)$ представляет собой совокупность значений производной по всем точкам x (на рисунках показаны производные в одной произвольной точке x_1).

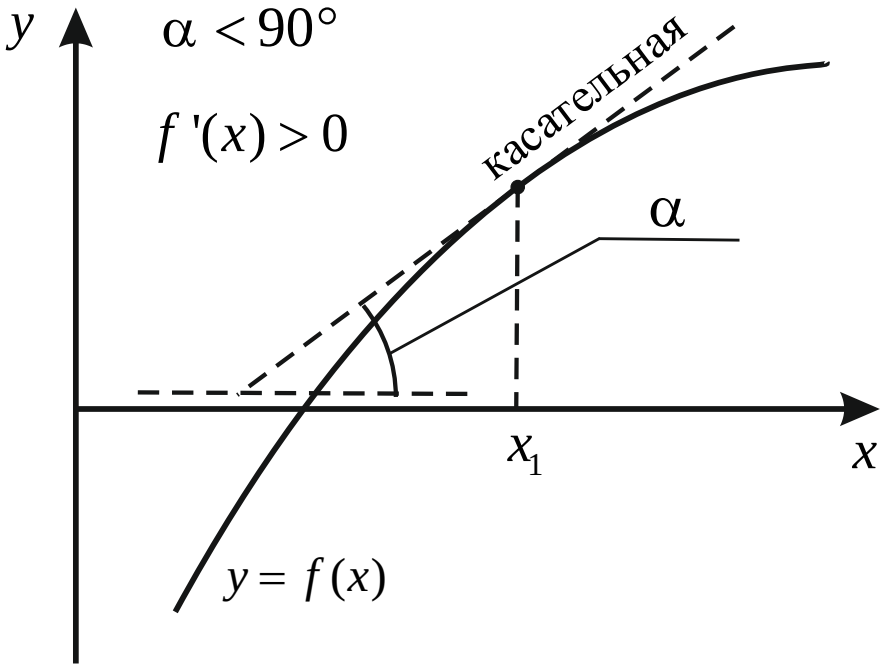

Рис. 3. Производная $f'(x)$ функции $f(x)$ в точке x_1 определяется тангенсом угла наклона α касательной в этой точке. Производная положительна, если $\alpha < 90°$

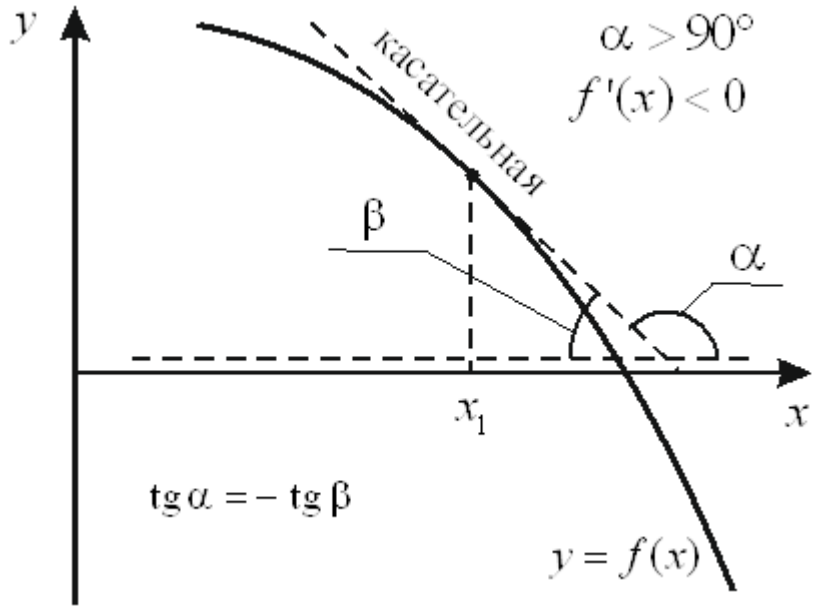

Рис. 4. Производная $f'(x)$ функции $f(x)$ в точке x_1 определяется тангенсом угла наклона α касательной в этой точке. Производная отрицательна, если $\alpha > 90°$; при этом она определяется через тангенс угла β, смежного с углом α, по формуле приведения $\operatorname{tg}\alpha = -\operatorname{tg}\beta$

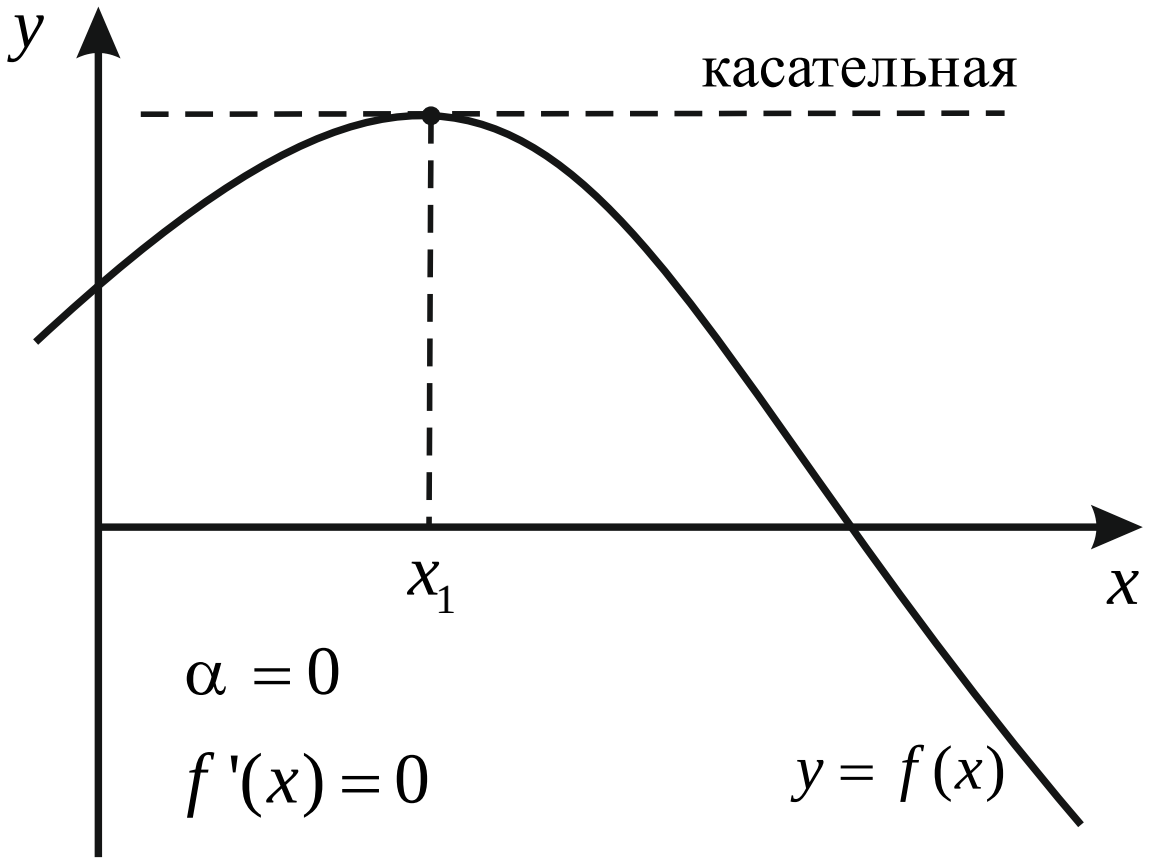

Рис. 5. Производная $f'(x)$ функции $f(x)$ в точке x_1 определяется тангенсом угла наклона α касательной в этой точке.
Производная равна 0, если $\alpha = 0$

Исходя из вышесказанного можно проиллюстрировать теорему Вейерштрасса следующим образом. Если функция $f(x)$ монотонно убывает на отрезке [*a*, *b*] (или, наоборот – монотонно возрастает) и при этом пересекает ось *x*, (имеет корень x^*), то функция от точки *a* будет лежать выше оси *x*, а функция от точки *b* – ниже этой оси, т. е. функция будет менять знак на концах отрезка. При этом знак производной не меняется, т. е. наклон касательной $\alpha > 90^o$ (рис. 6). Если же знак производной на отрезке [*a*, *b*] будет меняться, то возможно пересечение оси *x* большее количество раз (рис. 7).

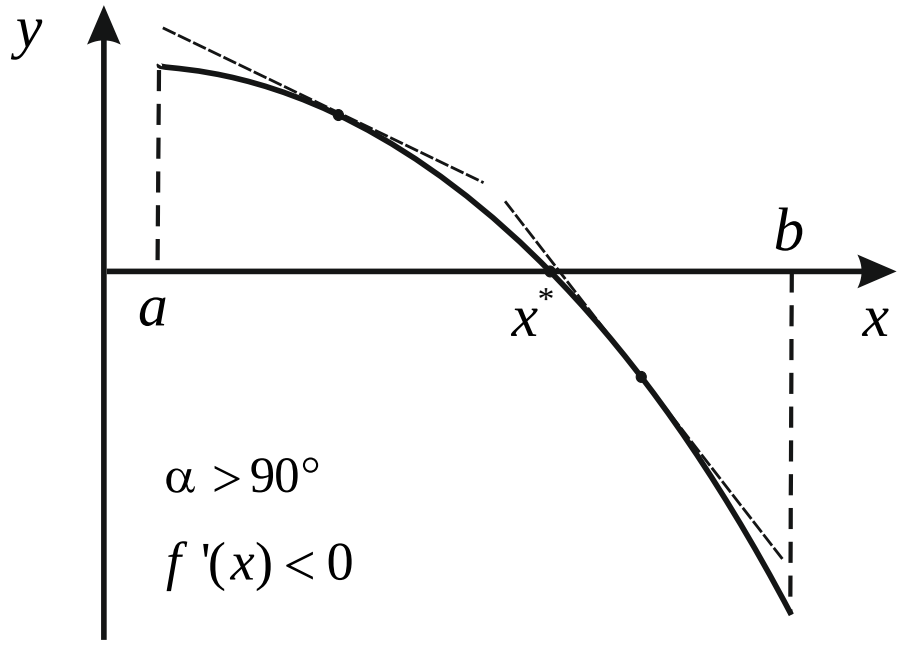

Рис. 6. Иллюстрация теоремы Вейерштрасса. Производная на отрезке $[a, b]$ знак не меняет; имеется только один корень

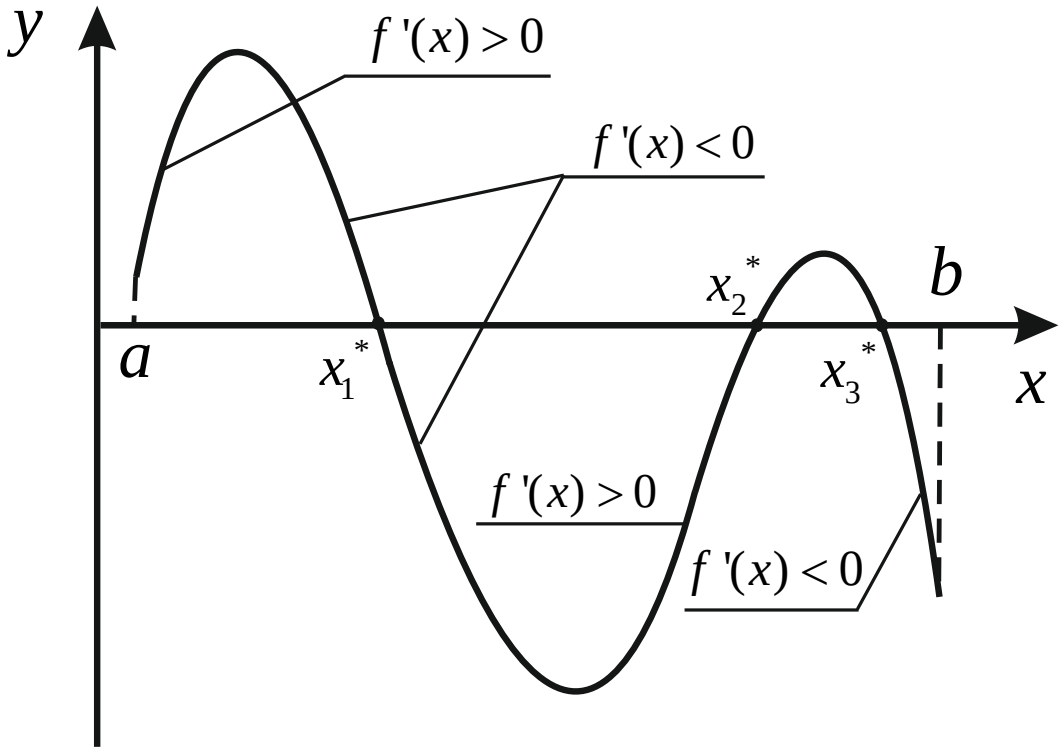

Рис. 7. Иллюстрация теоремы Вейерштрасса. Производная на отрезке $[a, b]$ несколько раз меняет свой знак; имеется несколько корней

Таким образом, **первый этап – отделение корней –** представляет собой предварительное математическое исследование для применения конкретных численных методов решения нелинейных уравнений. Рассмотрим четыре метода: 1) **метод деления отрезка пополам**, 2) **метод хорд**, 3) **метод простых итераций Якоби** и 4) **метод Ньютона**.

§ 5. Метод деления отрезка пополам

Другие названия метода: **метод дихотомии**, **метод бисекций**. Идея метода состоит в делении отрезка, содержащего корень, пополам; определении, в какой из двух половин останется корень, и отбрасывании ненужной половины. Затем процесс повторяется, т. е. в каждой следующей итерации уменьшается длина отрезка, содержащего корень. Итерации можно прекратить, когда эта длина станет ничтожной. Вместо **итерационной формулы** используется **критерий наличия корня** на отрезке, основанный на теореме Вейерштрасса. Если функция имеет на концах отрезка разные знаки (вне зависимости от того, возрастающая она или убывающая), это свидетельствует о наличии корня и произведение этих функций всегда будет отрицательным (см. рис. 1, 4): $f(a)\cdot f(b)<0$. В методе исходный отрезок $[a, b]$ делится пополам, берётся средняя на отрезке точка $(a+b)/2$. Критерий можно применить к любому из двух отрезков, например – к левому:

$$\boxed{f(a)f\left(\frac{a+b}{2}\right)<0} \text{ – } \textbf{критерий наличия корня.} \qquad (3)$$

Программно можно поставить такое условие (условный оператор) и в зависимости от его выполнения или невыполнения можно отбросить левый или правый отрезок. Отбрасывание отрезка программно означает перенос точки a в точку $(a+b)/2$ для левого отрезка (это делается с помощью оператора присваивания) или перенос точки b в точку $(a+b)/2$ для правого отрезка:

a:=(a+b)/2 или b:=(a+b)/2.

Таким образом проводится одна итерация. **Начальным приближением** метода служит исходный отрезок $[a, b]$, полученный на этапе

отделения корней. После каждой итерации длина отрезка уменьшается вдвое.

В методе не требуется **критерий сходимости**, так как условие (3) обеспечивает нахождение корня. Как и в подавляющем большинстве итерационных методов, должна быть задана **точность** ε, с которой необходимо найти корень; по ней строится **критерий окончания** – итерации следует заканчивать, когда длина полученного отрезка $[a, b]$ (лучше – по модулю) станет меньше заданной точности:

$$\boxed{|b-a|<\varepsilon} \text{ – } \textbf{критерий окончания.} \qquad (4)$$

При составлении **программы** (в данном пособии программы представлены на языке Pascal; для простоты применены комментарии, начинающиеся с «//», которых нет в классическом Pascal, но которые используются в Delphi или Lazarus) необходимо описать переменные, которые фигурируют в алгоритме, – концы отрезка $[a, b]$, точность ε (заменяем на латинский эквивалент, например eps), искомый корень x, а также функцию $f(x)$ с помощью **function**. Затем нужно ввести значения a, b, eps (так как они заданы) и организовать итерационный цикл уменьшения отрезка вдвое по критерию (3) с помощью оператора **repeat** или **while**. Выход из итерационного цикла будет осуществляться по критерию (4). В качестве результата, т. е. значения корня, можно принять середину полученного отрезка. Таким образом, программа может выглядеть так:

```
program dihotomia;
//описание переменных a, b и eps, x можно не описывать, так как
//он, как мы увидим ниже, в алгоритме участвовать не будет
var a,b,eps: real;
//описание заданной функции (например f), конкретный
//вид функции не приведён
function f(x:real):real;begin <…>end;
//начало основной программы
begin
//ввод исходных данных
write(‘a=’);readln(a);write(‘b=’);readln(b); write(‘eps=’);readln(eps);
//итерационный цикл (например – repeat)
repeat if f(a)*f((a+b)/2)<0 then b:=(a+b)/2 else a:=(a+b)/2;
```

```
//выход из цикла по критерию (4)
until abs(b-a)<eps;
//вывод полученного результата, значение корня x можно
//вывести в виде арифметического выражения; таким образом,
//переменная x непосредственно в алгоритме участвовать
//не будет
writeln('x=',(a+b)/2);
end.
```

Большое **преимущество** данного метода – возможность его применения к **недифференцируемым функциям** (но функция должна быть **непрерывна**) и отсутствие необходимости **предварительного математического исследования**. Второе обстоятельство позволяет использовать этот метод как **вспомогательный** при решении более сложных задач вычислительной математики, когда на некоторых этапах приходится решать нелинейные уравнения без дополнительной информации по этим уравнениям, т. е. этот метод самый простой и универсальный.

Серьёзный **недостаток** метода – его **медленная сходимость**, число итераций N приближённо можно найти по формуле

$$N \approx \log_2 \frac{b-a}{\varepsilon}. \qquad (5)$$

§ 6. Метод хорд

Другие названия метода: **метод секущих**, **метод ложного положения**.

Метод принципиально мало отличается от метода деления отрезка пополам, но обладает несколько более **быстрой сходимостью**.

Хордой для произвольной кривой служит прямая, соединяющая две точки этой кривой. Идея метода состоит в следующем. Хорда проводится через точки кривой, соответствующие аргументам a и b, т. е. концам отрезка $[a, b]$, содержащего корень x^* (рис. 8). Она делит отрезок на две части (точка x), которые анализируются по теореме Вейерштрасса и пустой отрезок отбрасывается, как и в предыдущем методе. Затем итерация повторяется.

На рис. 8 показаны три итерации. Первая хорда пересекает ось x в точке x_1; в эту точку после соответствующего анализа переносится (в данном случае) точка a, и итерация повторяется. После трёх итераций получаем приближение x_3.

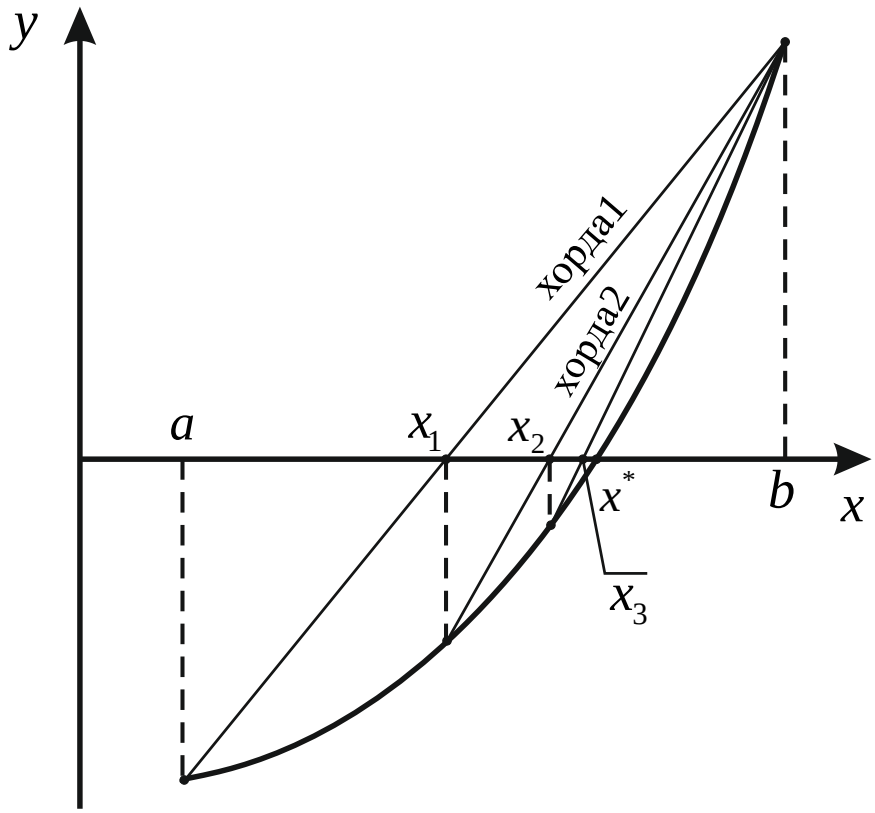

Рис. 8. Графическое представление метода хорд.
Показаны три итерации с последовательными
приближениями x_1, x_2 и x_3

Проиллюстрируем преимущество метода хорд перед дихотомией графически. Если функция $f(x)$ достаточно гладкая, в методе деления отрезка пополам чаще всего в каждой новой итерации попеременно отбрасывается то правый, то левый отрезок, а в методе хорд всегда отбрасываются либо только правые, либо только левые отрезки, т. е. точка x постепенно «подтягивается» к корню x^* (на рис. 8 «подтягивается» к корню конкретная точка a). Это можно учесть в алгоритме и после первой итерации не тратить машинные ресурсы на анализ по теореме Вейерштрасса. На рис. 9 представлена точно такая же функция и решение уравнения методом дихотомии. Из рисунка видно, что за те же три итерации получается менее точное приближение к корню.

Необходимо отметить, что подобная ситуация наблюдается не всегда; теоретически характер изменения функции может быть таков, что дихотомия будет более оптимальна. И всё же в подавляющем большинстве случаев метод хорд будет сходиться быстрее. Основной же выигрыш в быстроте будет давать отсутствие анализа по теореме Вейерштрасса в каждой итерации.

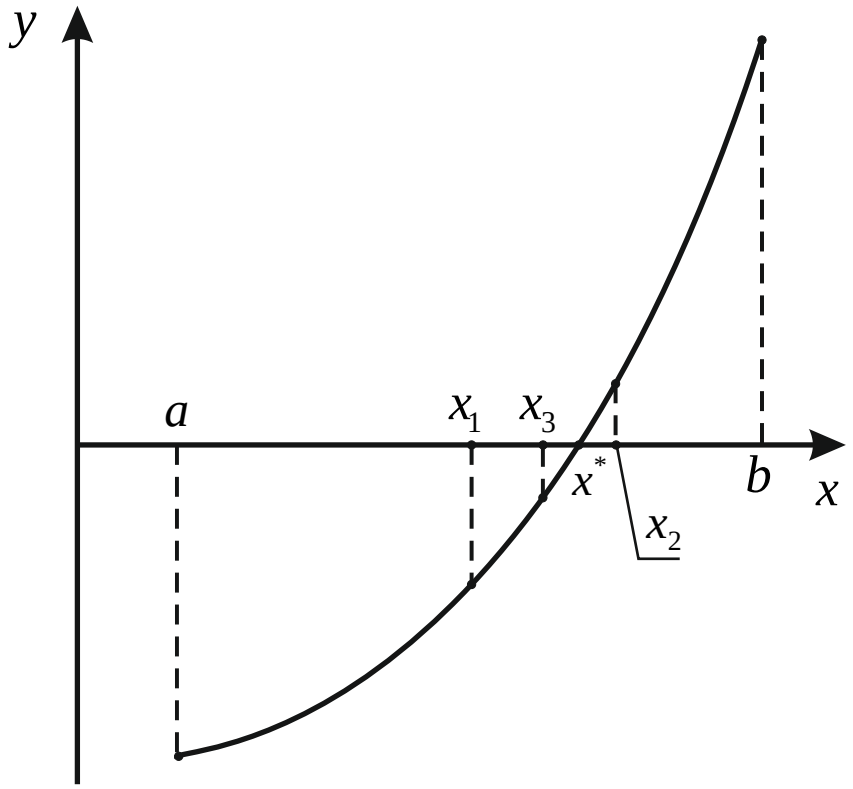

Рис. 9. Графическое представление метода деления отрезка пополам. Показаны три итерации с последовательными приближениями x_1, x_2 и x_3

Итерационную формулу проще всего получить из геометрических соображений. Она должна давать способ вычисления точки x. Дополним хорду до прямоугольного треугольника и определим точку x по известным параметрам. При этом возможны четыре варианта: убывающая или возрастающая функция, выпуклая вверх или выпуклая вниз; все эти варианты представлены на рис. 10 – 13.

Треугольник можно достраивать вниз от хорды или вверх, как будет видно ниже, это даёт те же результаты. Для определённости будем достраивать треугольник вниз.

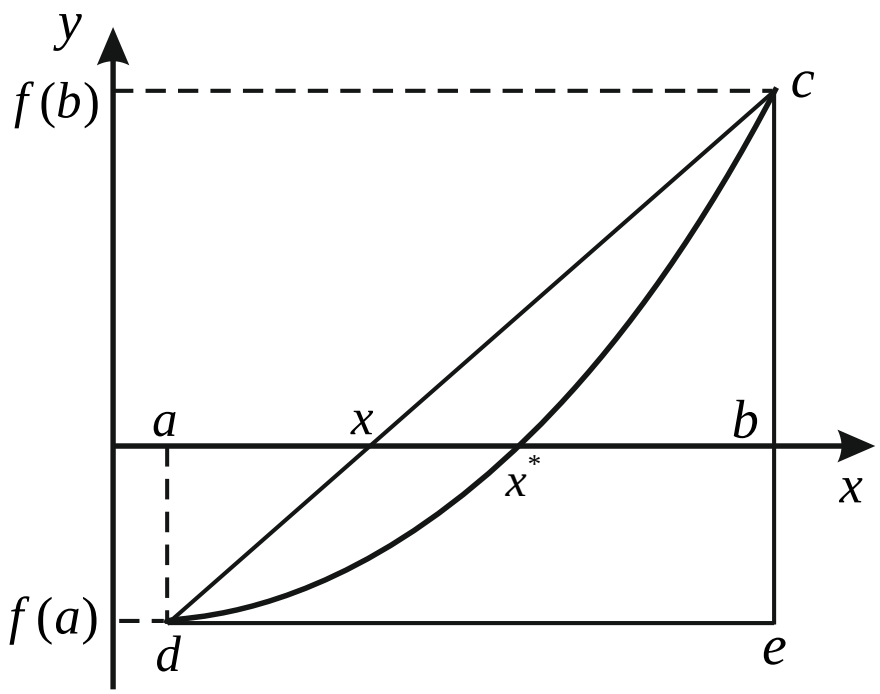

Рис. 10. К выводу итерационной формулы. Возрастающая выпуклая вниз функция

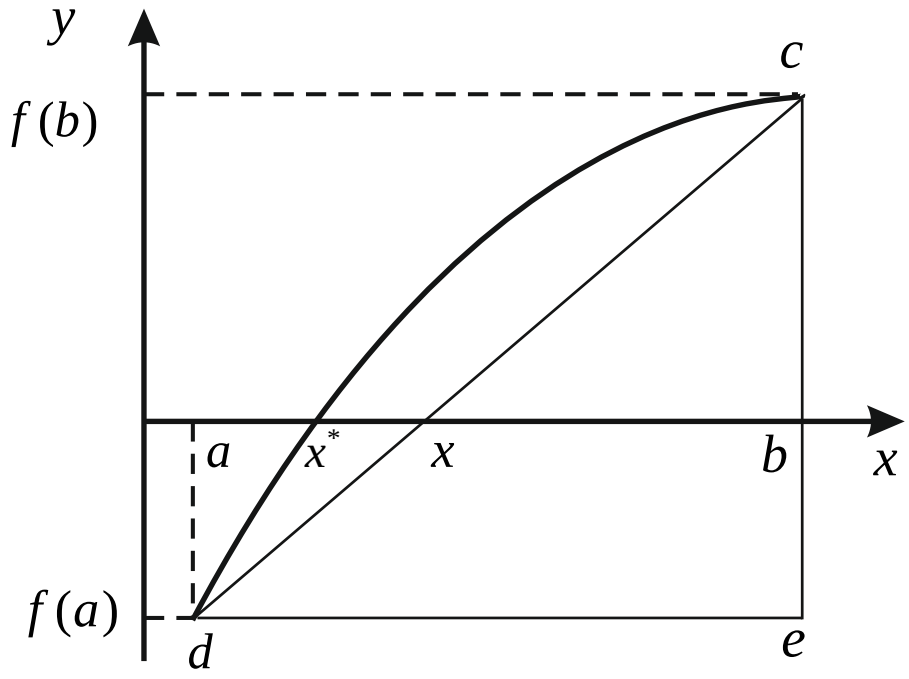

Рис. 11. К выводу итерационной формулы. Возрастающая выпуклая вверх функция

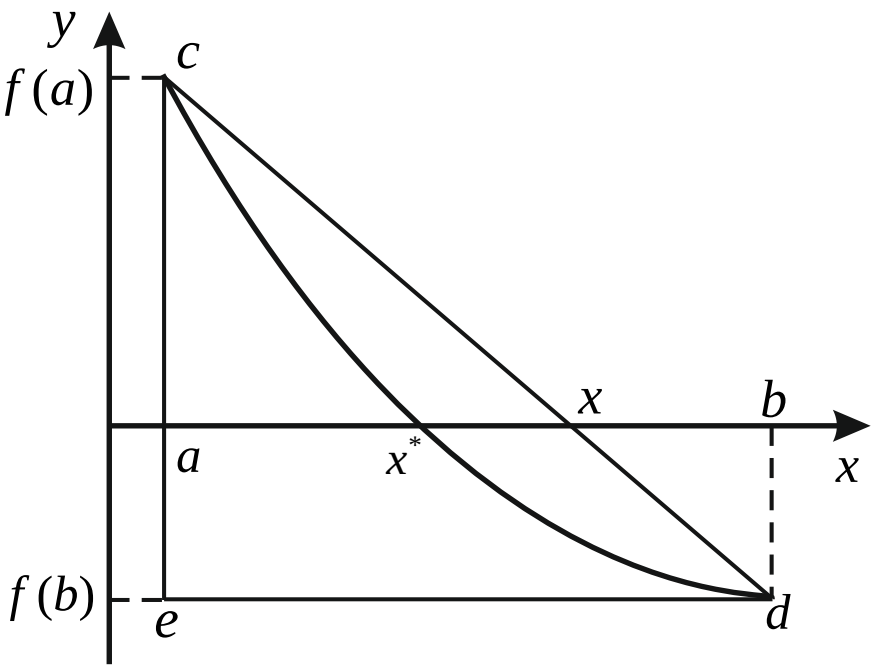

Рис. 12. К выводу итерационной формулы. Убывающая выпуклая вниз функция

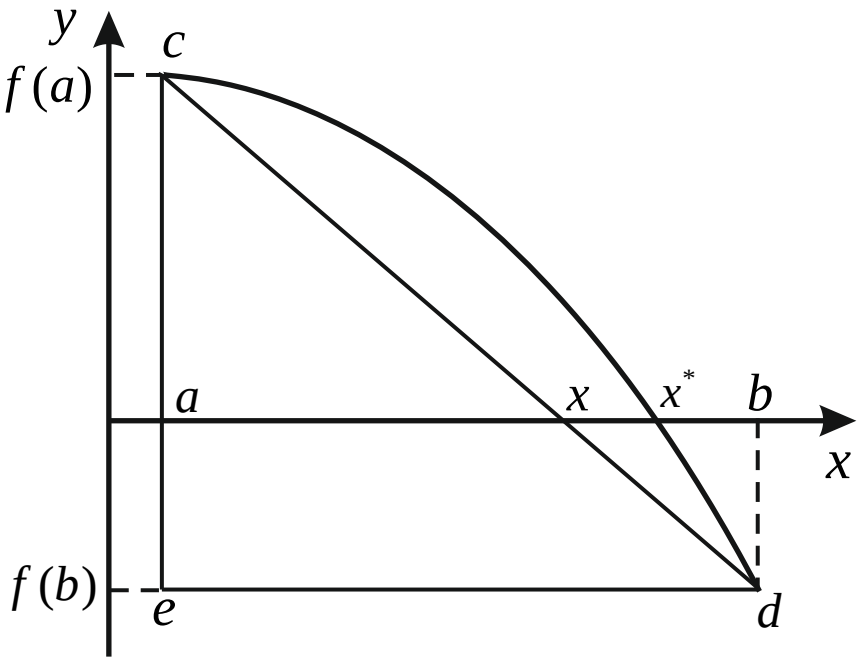

Рис. 13. К выводу итерационной формулы. Убывающая выпуклая вверх функция

На рис. 10 и 11 представлены возрастающие функции; в этих случаях мы получаем совершенно одинаковые треугольники: основной Δdce и смежный Δxcb. Из подобия этих треугольников имеем

$$\frac{b-x}{b-a}=\frac{f(b)-0}{f(b)-f(a)},$$

откуда можно найти искомую точку пересечения хорды с осью абсцисс:

$$\boxed{x=b-(b-a)\frac{f(b)}{f(b)-f(a)}}. \qquad (6)$$

Для убывающих функций (см. рис. 12 и 13) имеем подобные треугольники основной Δecd и смежный Δacx, для которых

$$\frac{x-a}{b-a}=\frac{f(a)-0}{f(a)-f(b)}$$

и

$$\boxed{x=a+(b-a)\frac{f(a)}{f(a)-f(b)}}. \qquad (7)$$

Полученные формулы, на первый взгляд, разные. Однако, если достраивать треугольники вверх от хорды, для возрастающих функций получим

$$\frac{x-a}{b-a}=\frac{0-f(a)}{f(b)-f(a)},$$

что приводит к формуле (7); и, наоборот, для убывающих функций

$$\frac{b-x}{b-a}=\frac{0-f(b)}{f(a)-f(b)}$$

ведёт к формуле (6). Таким образом, формулы (6) и (7) равнозначны, и каждая пригодна для всех случаев.

Как и в предыдущем методе, н**ачальным приближением** служит исходный отрезок [a, b], полученный на этапе отделения корней, и также не требуется **критерий сходимости**, так как условие (3) обеспечивает нахождение корня.

Критерием окончания итерационного процесса не может служить выражение (4), так как в методе хорд исходный отрезок не уменьшается до минимальной длины, меньшей заданной **точности** ε, один из концов отрезка фиксируется, а другой «подтягивается» к корню, т. е. длина отрезка остаётся довольно большой. Поэтому **критерий окончания** строится из сравнения двух последних итераций – процесс следует за-

канчивать, когда их разница по модулю станет меньше заданной точности. Если x_n и x_{n+1} – предыдущее и последующее приближения корня (n – номер итерации), то:

$$\boxed{|x_{n+1} - x_n| < \varepsilon} \text{ – } \textbf{критерий окончания.} \qquad (8)$$

Составление **программы.** Как и в предыдущем методе, необходимо описать концы отрезка $[a, b]$, точность ε, искомый корень x, а также функцию $f(x)$ с помощью **function**. Так как в критерии окончания фигурируют предыдущее и последующее приближение корня, нужно ещё описать предыдущий x0. Затем нужно ввести значения a, b, eps и провести первую итерацию по формуле, например (6), с анализом по теореме Вейерштрасса:

$$\boxed{f(a) \cdot f(x) < 0}. \qquad (9)$$

После этого станет ясно, какая точка, a или b, «подтягивается» к корню. Это следует зафиксировать во вспомогательной целочисленной переменной, например – m, пусть $m = 1$, если «подтягивается» точка a, и $m = 2$, если – точка b. Переменную m необходимо дополнительно описать. Далее нужно организовать итерационный цикл вычисления нового значения x по (6) с помощью оператора, например **repeat**. В начале каждого цикла (а первый цикл будет соответствовать второй итерации) необходимо сделать переприсваивание вычисленного перед этим значения x предыдущему значению x0: x0:=x. В конце цикла нужно уменьшить отрезок $[a, b]$ перенесением точки a или b во вновь полученную точку x. Для этого нужно проанализировать m (оператор **if**) и сделать соответствующее присваивание. (Можно заметить, что более изящный алгоритм, без m, можно было бы построить с использованием такой конструкции языка, как **указатель**). Выход из итерационного цикла будет осуществляться по критерию (8). В качестве результата принимается последнее приближение корня – x. Описанная программа может выглядеть так:

```
program horda;
//описание переменных a, b, eps, x, x0 и m
var a,b,eps,x,x0:real; m:integer;
//описание заданной функции (например f), конкретный
//вид функции не приведён
function f(x:real):real; begin <…>end;
//начало основной программы
```

```
begin
//ввод исходных данных
write('a=');readln(a);write('b=');readln(b); write('eps=');readln(eps);
//первая итерация по формуле (6) (можно использовать и (7))
x:=b-(b-a)*f(b)/(f(b)-f(a))
//анализ по теореме Вейерштрасса (9)
if f(a)*f(x)<0 then m:=2 else m:=1;
//уменьшение отрезка
if m=1 then a:=x else b:=x;
//итерационный цикл (например – repeat); в начале
//цикла – переприсваивание
repeat x0:=x;
//вычисление нового приближения
x:=b-(b-a)*f(b)/(f(b)-f(a))
//уменьшение отрезка
if m=1 then a:=x else b:=x;
//выход из цикла по критерию (8)
until abs(x-x0)<eps;
//вывод полученного результата – последнее приближение корня
writeln('x=',x);
end.
```

§ 7. Метод простых итераций (Якоби)

В основе метода лежит тождественное преобразование исходного уравнения (1) в уравнение вида

$$\boxed{f(x)=0 \quad \Rightarrow \quad x=\varphi(x)}, \tag{10}$$

т. е. из функции $f(x)$ каким-то образом (вообще говоря, произвольным) выражается x в «чистом» виде. Всё, что после этого остаётся, называется функцией $\varphi(x)$. Это преобразование, разумеется, не является решением уравнения (1), так как хотя слева получается x, он же остаётся в правой части в составе функции $\varphi(x)$. Таким образом, все дальнейшие действия производятся с эквивалентным уравнением

$$\boxed{x=\varphi(x)}. \tag{11}$$

Заметим также, что функция $\varphi(x)$ может быть получена разными способами; обычно сначала преобразование (9) делают наиболее простым способом, а если далее возникают сложности применения метода простых итераций (например, нарушение критерия сходимости), $\varphi(x)$ можно выразить другим способом.

Пример 1. Например, дано исходное уравнение $f(x)=0$:

$$0.8x^2 - 2x - \ln x = 0. \qquad (12)$$

График этой функции представлен на рис. 14. Для уравнения (12) имеем два корня x_1^* и x_2^*. В результате графического отделения корней можно выбрать два интервала, содержащие корни: первый корень

$$[0.0001, 1] \qquad (13)$$

(для этого случая нельзя брать левую границу интервала 0, так как функция содержит логарифм), второй корень

$$[2, 4]. \qquad (14)$$

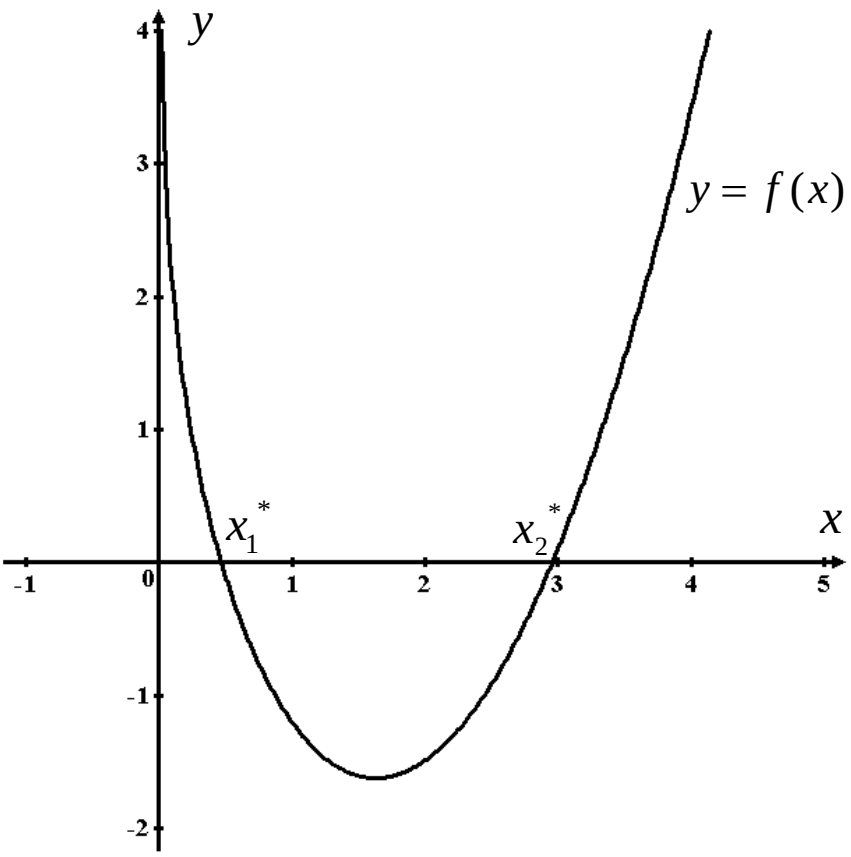

Рис. 14. График функции $y = 0.8x^2 - 2x - \ln x$;
x_1^* и x_2^* – корни уравнения $0.8x^2 - 2x - \ln x = 0$

Функцию $\varphi(x)$ можно выразить тремя способами:

$$x = \sqrt{\frac{\ln x + 2x}{0.8}}, \text{ т. е. } \varphi(x) = \sqrt{\frac{\ln x + 2x}{0.8}}; \tag{15}$$

$$x = \frac{0.8x^2 - \ln x}{2}, \text{ т. е. } \varphi(x) = \frac{0.8x^2 - \ln x}{2}; \tag{16}$$

$$x = e^{0.8x^2 - 2x}, \text{ т. е. } \varphi(x) = e^{0.8x^2 - 2x}. \tag{17}$$

Видимо, наиболее простой способ – второй, с него можно и начать. Причём все три способа, если они применимы, должны приводить к одному и тому же результату.

Метод простых итераций основан на теореме (доказательство которой здесь не приводится):

Теорема. *Пусть* $\varphi(x)$ *определена и дифференцируема на отрезке* $[a, b]$ *и все её значения* $\varphi(x) \in [a, b] \; \forall \; x \in [a, b]$. *Тогда, если существует число* q, *такое, что* $|\varphi'(x)| \le q < 1$ *на отрезке* $[a, b]$, *то последовательность* $x_{n+1} = \varphi(x_n), \quad n = 0, 1, 2, \ldots$ *сходится к единственному на* $[a, b]$ *решению уравнения* $x = \varphi(x)$ *при любом начальном значении* $x_0 \in [a, b]$.

Из теоремы следует, что **итерационная формула метода простых итераций**

$$\boxed{x_{n+1} = \varphi(x_n), \quad n = 0, 1, 2, \ldots} \tag{18}$$

(n – номер итерации), т. е. последующее приближение корня x_{n+1} находится на основе предыдущего приближения x_n.

Начальным приближением x_0 может служить любая точка отрезка $[a, b]$; чаще всего для простоты принимают один из концов отрезка. Но это начальное приближение должно удовлетворять **критерию сходимости**

$$\boxed{|\varphi'(x_0)| < 1}. \tag{19}$$

На практике после преобразования (9) в критерий (19) подставляют концы отрезка $[a, b]$. Возможны три варианта: а) подходят обе точки – выбирают любую, б) подходит одна из точек – выбор однозначен, в) ни одна из точек не подходит. В последнем случае можно попробовать немного расширить или сузить интервал, например, если $a = 0$, а в критерий сходимости входит логарифм, то эта точка заведомо

не подходит. Однако, если взять не 0, а какое-то малое число, например 0,0001, то критерий может пройти. Если всё это не помогает, можно попробовать выразить $\varphi(x)$ другим способом, т. е. сделать преобразование (9) по-другому. Если и это не приведёт к успеху, можно сделать вывод, что данный метод не пригоден для нахождения корня и требуется применение другого метода.

Как и в предыдущем методе, **критерий окончания** строится из сравнения двух последних итераций:

$\boxed{|x_{n+1} - x_n| < \varepsilon}$ **– критерий окончания метода простых итераций.** (20)

Пример 1 (продолжение, см. (12)).

Функция $\varphi(x)$ – (15), рис. 15. Производная

$$\varphi'(x) = \frac{1/x + 2}{2\sqrt{0.8(\ln x + 2x)}}.$$

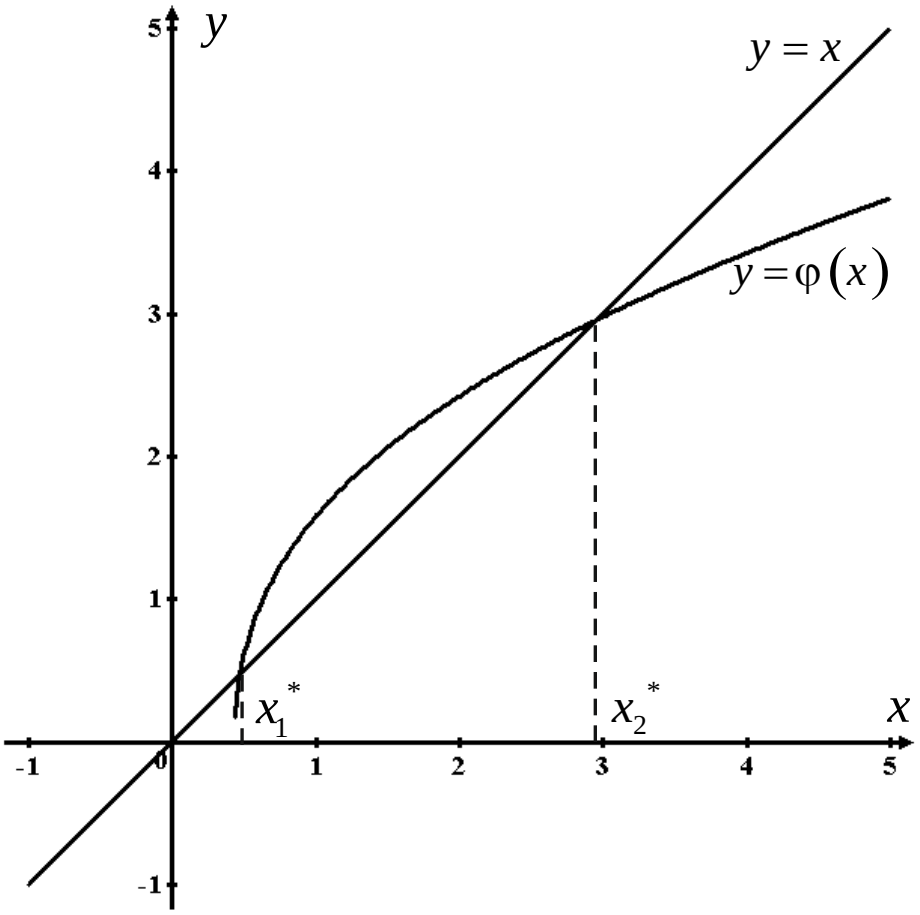

Рис. 15. График функции $\varphi(x)$ (15)

Подставляем концы интервалов. Первый корень – интервал (13), для $x_0 = 0.0001$ производная не существует, для $x_0 = 1$ $\varphi'(x_0) = 0.3953 < 1$ критерий выполняется. Второй корень – интервал (14), для $x_0 = 2$ $\varphi'(x_0) = 0.6451 < 1$, для $x_0 = 4$ $\varphi'(x_0) = 0.4105 < 1$; обе точки подходят. Выбираем

$$x_0 = 1 \text{ и } x_0 = 4. \tag{21}$$

Функция $\varphi(x)$ – (16). Производная

$\varphi'(x) = 0.8x - \dfrac{1}{2x}$ (рис. 16).

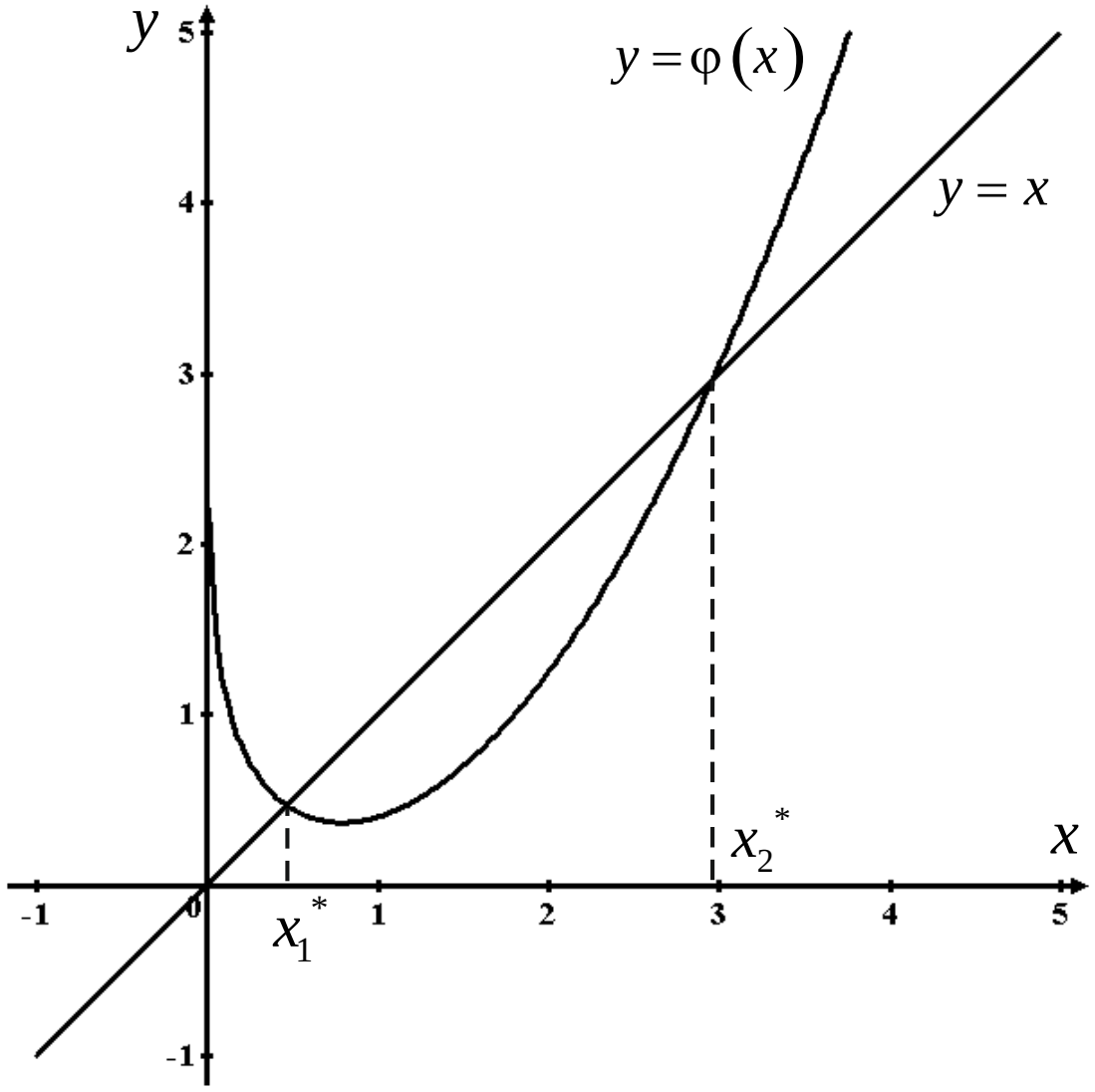

Рис. 16. График функции $\varphi(x)$ (16)

Первый корень, для $x_0 = 0.0001$ $|\varphi'(x_0)| = |-5000| > 1$ – критерий не выполняется, для $x_0 = 1$ $\varphi'(x_0) = 0.3 < 1$ – выполняется. Второй корень – интервал (14), для $x_0 = 2$ $\varphi'(x_0) = 1.35 > 1$, для $x_0 = 4$ $\varphi'(x_0) = 3.075 > 1$; обе точки не подходят. Выбираем

$x_0 = 1$ – только для первого корня

Функция $\varphi(x)$ – (17), рис. 17. Производная

$$\varphi'(x) = (1.6x - 2)e^{0.8x^2 - 2x}. \qquad (22)$$

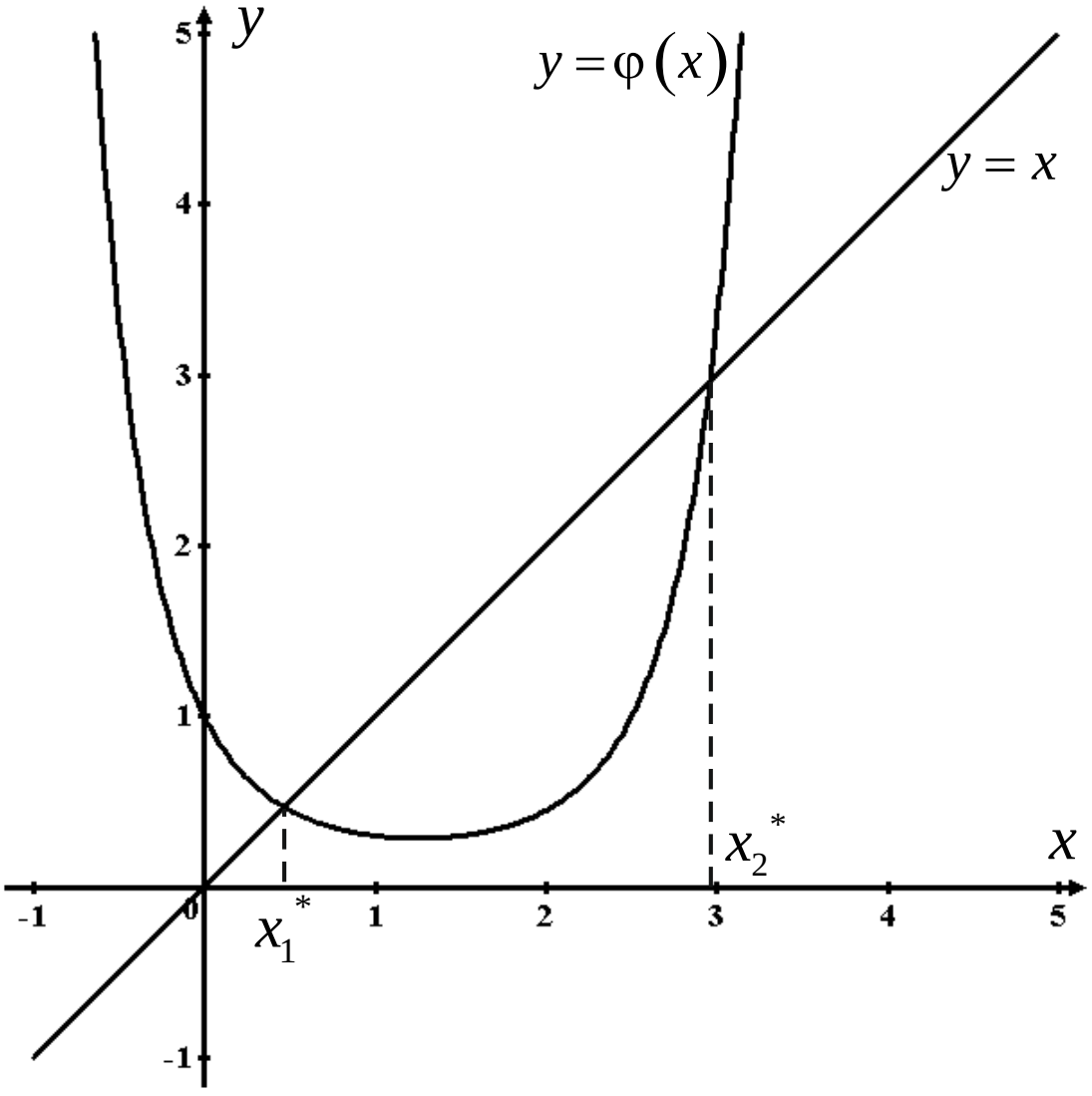

Рис. 17. График функции $\varphi(x)$ (17)

Первый корень, для $x_0 = 0.0001$ $|\varphi'(x_0)| = |-1.9996| > 1$, для $x_0 = 1$ $|\varphi'(x_0)| = |-0.1205| < 1$. Второй корень – интервал (14), для $x_0 = 2$ $\varphi'(x_0) = 0.5392 < 1$, для $x_0 = 4$ $\varphi'(x_0) = 777.7 > 1$. Выбираем

$$x_0 = 1 \text{ и } x_0 = 2. \tag{23}$$

(Конец продолжения примера 1).

Приведём упрощённый, графический, способ вывода итерационной формулы метода и критерия сходимости. Преобразование (9) фактически представляет собой разбиение функции $y = f(x)$ на две функции

$$y_1 = x \tag{24}$$

и

$$y_2 = \varphi(x), \tag{25}$$

т. е. то же, что мы делали на этапе отделения корней (см. рис. 2). Таким образом, первая функция всегда будет иметь вид (24) и на графике изображаться как биссектриса первой четверти. Вторая функция, $\varphi(x)$, может располагаться по отношению к первой четырьмя различными способами, что приводит к четырём разным ситуациям.

1) $\varphi'(x) > 0$ и $|\varphi'(x)| < 1$,

т. е. $\varphi(x)$ функция **возрастающая** и **наклон касательной** в любой точке на [*a, b*] **меньше** 45º (рис. 18; нужно мысленно провести касательные к кривой $y = \varphi(x)$ в каждой точке). Так как tg 45º = 1, последнее означает, что $|\varphi'(x)| < 1$. Выберем начальное приближение x_0 и сделаем следующие построения. Из точки x_0 восстановим перпендикуляр до пересечения с кривой $y = \varphi(x)$ (т. е. находим значение $\varphi(x_0)$) и из полученной точки проведем горизонтальную линию до пересечения с $y = x$ (это делается на основании уравнения (11), т. е. мы приравниваем значения двух функций). Из точки, полученной на $y = x$, мы опускаем перпендикуляр на ось x и получаем следующее приближение корня x_1.

Таким образом реализуется итерационная формула (18). Далее, на следующей итерации, мы повторяем действия: из x_1 восстанавливаем перпендикуляр до $y=\varphi(x)$, приравниваем значения двух функций и опускаем перпендикуляр от $y=x$ до пересечения с осью *x*, т. е. получается приближение x_2; и т. д. Из рисунка видно, что каждая следующая итерация приближает получаемое значение x_i к корню, что соотносится с критерием сходимости (19).

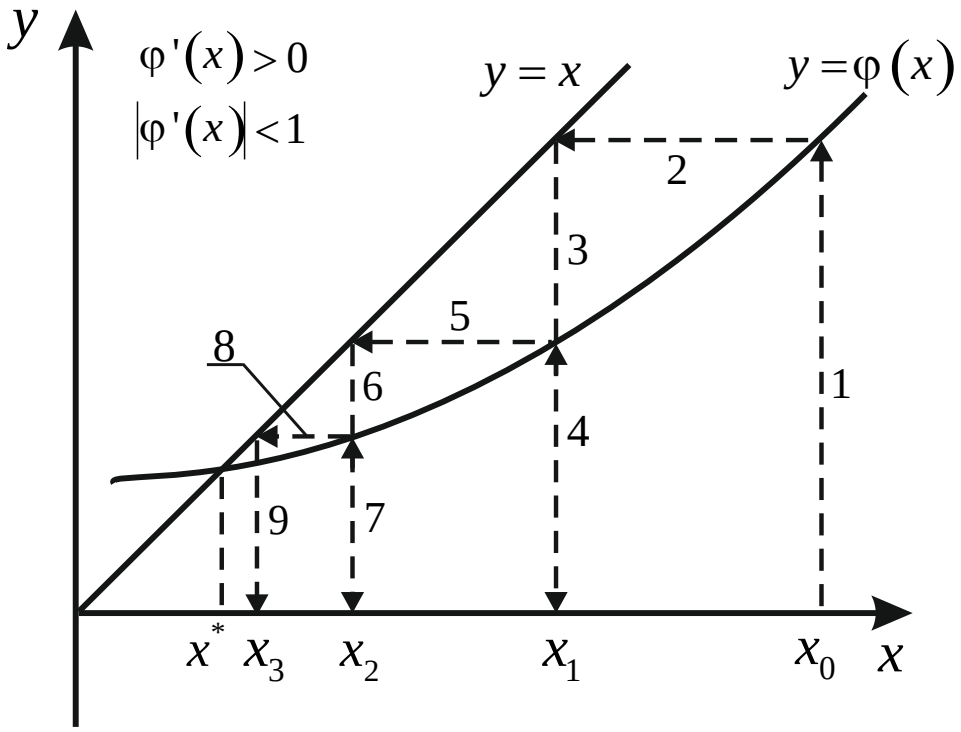

Рис. 18. Функция $y=\varphi(x)$ возрастающая и $|\varphi'(x)|<1$. x^* – корень уравнения $f(x)=0$, x_0 – начальное приближение, x_i – последовательные итерации. Цифры на штриховых линиях соответствуют порядку построения итераций. **Итерационный процесс сходится**

2) $\varphi'(x)<0$ и $|\varphi'(x)|<1$.

Функция **убывающая** и **наклон касательной меньше** 45º для сопряжённого угла (рис. 19), т. е. так же $|\varphi'(x)|<1$. Выберем начальное приближение x_0 и сделаем построения, аналогичные предыдущему пункту. Как видно из рисунка, итерации также сходятся, что отвечает

критерию (19). Что интересно, для этого случая последующие приближения x_i получаются попеременно слева и справа от корня.

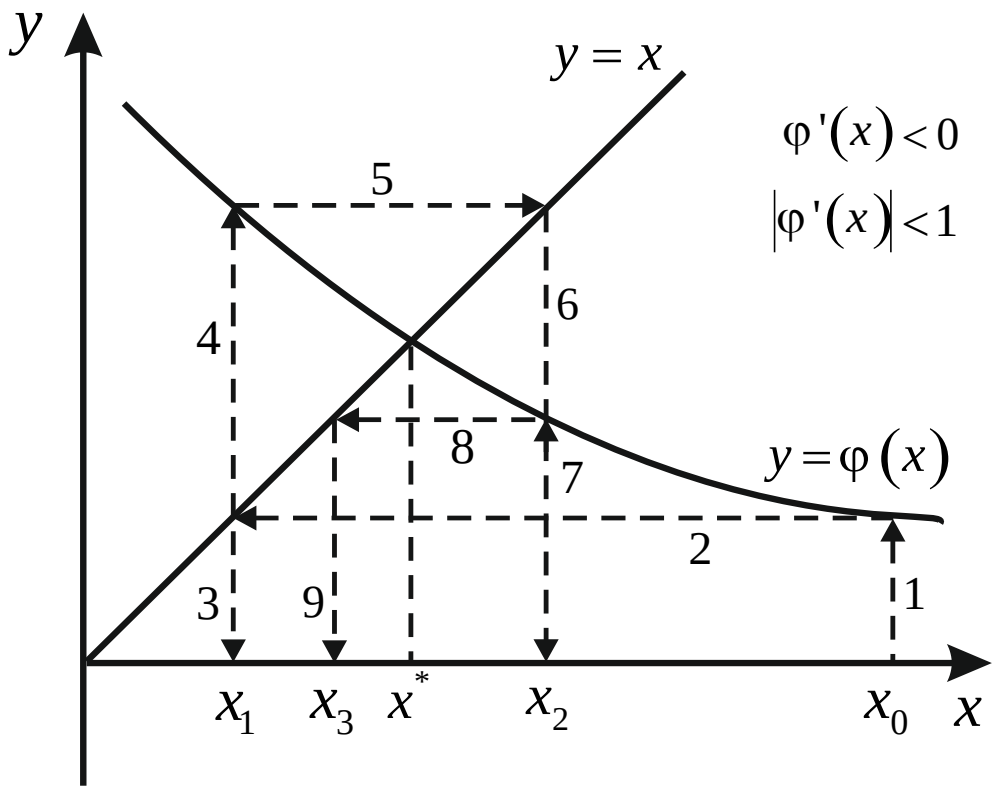

Рис. 19. Функция $y=\varphi(x)$ убывающая и $|\varphi'(x)|<1$. x^* – корень уравнения $f(x)=0$, x_0 – начальное приближение, x_i – последовательные итерации. Цифры на штриховых линиях соответствуют порядку построения итераций. **Итерационный процесс сходится**

3) $\varphi'(x)>0$ и $|\varphi'(x)|>1$.

Функция **возрастающая** и **наклон касательной больше** 45º (рис. 20), т. е. $|\varphi'(x)|>1$. Если сделать те же самые построения для этого случая, то мы обнаружим, что каждое следующее приближение x_i будет находиться ещё дальше от корня, т. е. итерации расходятся (и, соответственно, критерий сходимости не выполняется).

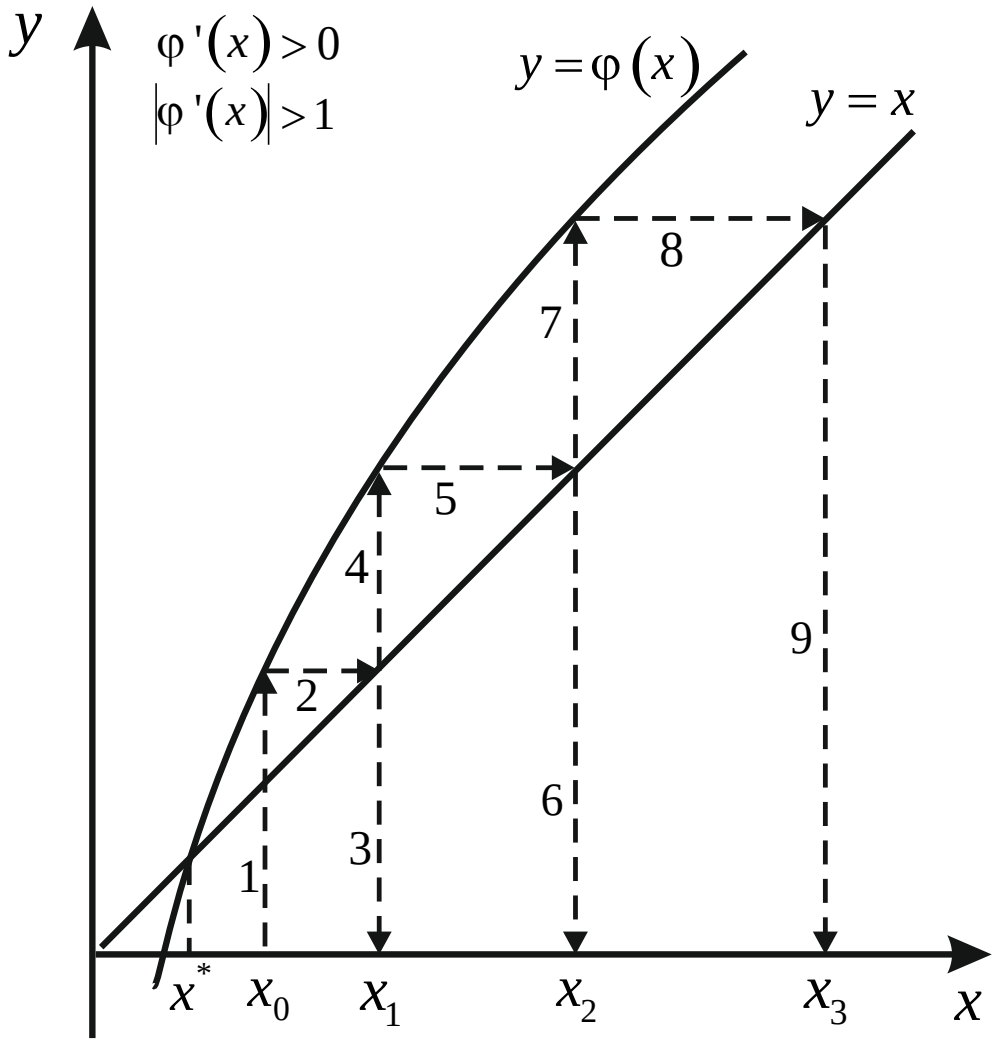

Рис. 20. Функция $y=\varphi(x)$ возрастающая и $|\varphi'(x)|>1$. x^* – корень уравнения $f(x)=0$, x_0 – начальное приближение, x_i – последовательные итерации. Цифры на штриховых линиях соответствуют порядку построения итераций. **Итерационный процесс расходится**

4) $\varphi'(x)<0$ и $|\varphi'(x)|>1$.

Функция **убывающая** и **наклон касательной больше** 45° для сопряжённого угла (рис. 21), т. е. так же $|\varphi'(x)|>1$. Критерий сходимости не выполняется, итерационный процесс расходится. Приближения x_i получаются попеременно слева и справа от корня.

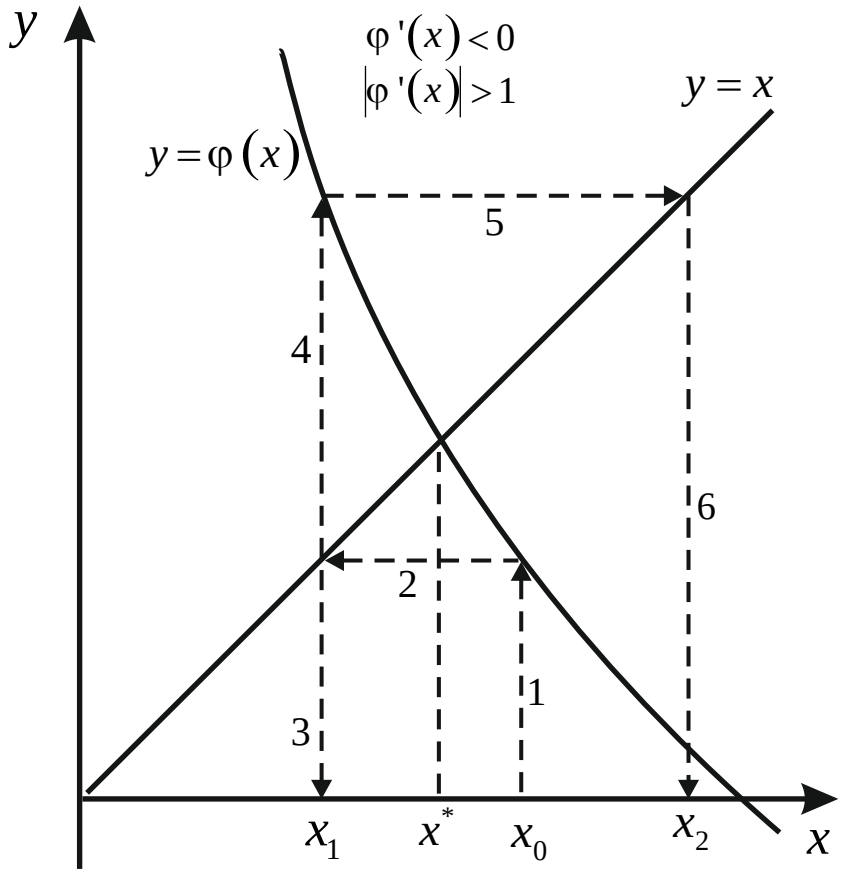

Рис. 21. Функция $y = \varphi(x)$ убывающая и $|\varphi'(x)| > 1$. x^* – корень уравнения $f(x) = 0$, x_0 – начальное приближение, x_i – последовательные итерации. Цифры на штриховых линиях соответствуют порядку построения итераций. **Итерационный процесс расходится**

Рекомендации к составлению программы. Поскольку в этом методе начальным приближением является x_0, проверяемый по критерию сходимости, сам отрезок [*a, b*] вводить не нужно. Из итерационной формулы (18) видно, что для проведения итераций необходимо иметь предыдущий корень x_n (обозначим его x0) и последующий x_{n+1} (обозначим x). Тогда описать следует три переменные – x, x0 и точность eps. Последние две переменные нужно ввести (x0 – как начальное приближение). Ввод может выглядеть так:

```
…
//ввод исходных данных
write(‘x0=’);readln(x0); //строка содержит ошибку!
write(‘eps=’);readln(eps);
…
```

Затем должен следовать итерационный цикл (например – **repeat**), где будут проводиться итерации по (18). Выход из итерационного цикла будет осуществляться по критерию (20), который нужно оформить как условный оператор после **until**. Чтобы итерационный цикл работал правильно, в конце каждого цикла необходимо сделать переприсваивание переменных x и x0 (оператор присваивания x0:=x;). Но если этот оператор будет стоять после итерационной формулы, итерации будут правильными, однако критерий окончания срабатывать не будет, так как x и x0 будут иметь одинаковое значение. Выходом из этого может быть помещение переприсваивания в начало итерационного цикла, перед итерационной формулой. При этом правильно будут работать все итерации, кроме первой. В самом деле, перед входом в цикл x0 будет иметь значение начального приближения (после ввода), а x – не будет иметь никакого определённого значения (реально – 0 или «мусор», т. е. любое случайное число). После входа в цикл x0 также потеряет свою определённость. Избежать этого можно с помощью следующего приёма. В показанном выше фрагменте программы, – вводе данных, значение начального приближения можно поместить не в ячейку памяти x0, а в ячейку памяти x (т. е. readln(x);), тогда, в самом первом цикле, это значение попадёт в переменную x0 (по переприсваиванию) и всё будет работать корректно. Текстовую подсказку ввода начального приближения (write(‘x0=’);) можно оставить без изменений.

Описанная программа может выглядеть так:

```
program jacobi;
//описание переменных eps, x и x0
var eps,x,x0: real;
//описание функции φ(x) (например f), конкретный
//вид функции не приведён
function f(x:real):real;begin <…>end;
//начало основной программы
begin
//ввод исходных данных
write(‘x0=’);readln(x); //ввод начального приближения в
//переменную x
write(‘eps=’);readln(eps);
//итерационный цикл (например – repeat); в начале
//цикла – переприсваивание
repeat x0:=x;
//вычисление нового приближения по (18)
x:=f(x0);
```

```
//выход из цикла по критерию (20)
until abs(x-x0)<eps;
//вывод полученного результата – последнее приближение корня
writeln('x=',x);
end.
```

Проиллюстрируем работу метода на примере.

Пример 2. Пусть дано уравнение

$$0.2x^2 - 1.4x + 0.7 = 0 \qquad (26)$$

Нужно найти корень этого уравнения, ближайший к началу координат. График представлен на рис. 22. Функцию $\varphi(x)$ можно выразить двумя способами:

$$\varphi(x) = x^2 / 7 + 0.5; \qquad (27)$$

$$\varphi(x) = \sqrt{7x - 3.5}. \qquad (28)$$

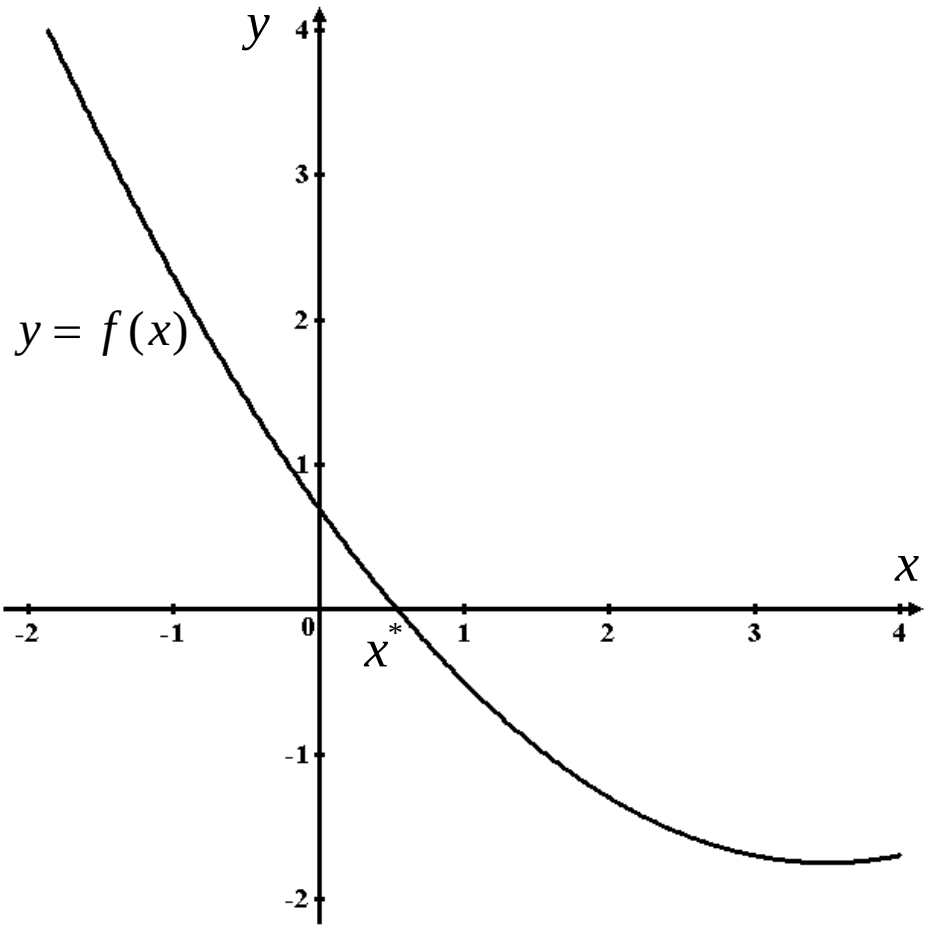

Рис. 22. График функции $y = 0.2x^2 - 1.4x + 0.7$; x^* – корень уравнения (26), ближайший к началу координат

Могут быть и другие способы выражения, например

$$\varphi(x) = 0.2x^2 - 0.4x + 0.7. \qquad (29)$$

Выберем интервал $[-1, 2]$.

Для (27) имеем $\varphi'(x) = 2x/7$. При $x_0 = -1$ $|\varphi'(x)| = |-0.2857| < 1$; при $x_0 = 2$ $|\varphi'(x)| = |0.5114| < 1$ (рис. 23). Обе точки подходят. Расчёт по программе с точностью, например, ε = 0.000001, даёт одинаковый результат $5.41960 \cdot 10^{-1}$.

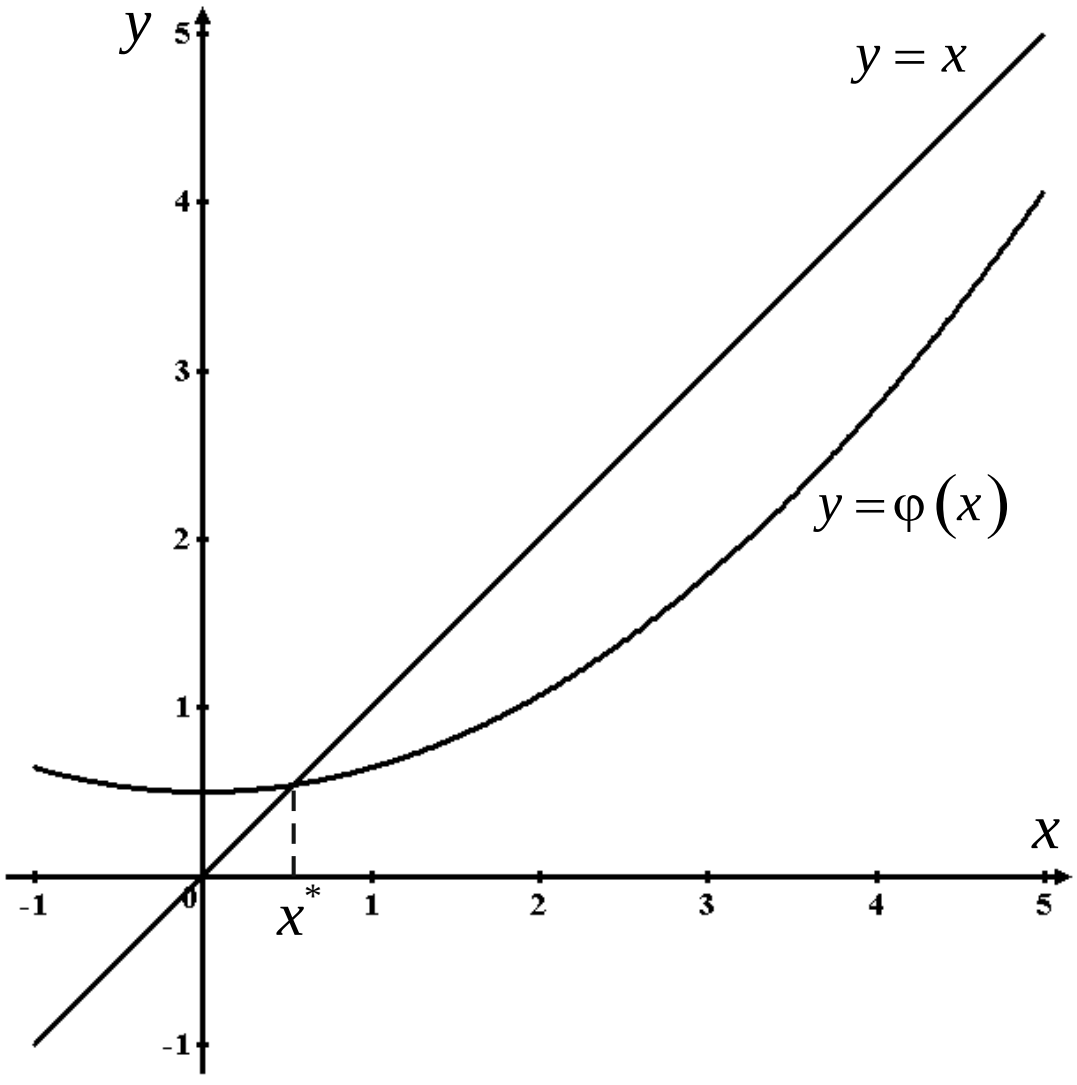

Рис. 23. График функции (27), x^* – корень уравнения (26), ближайший к началу координат

Для (28): $\varphi'(x) = 7 \Big/ \left(2\sqrt{7x - 3.5}\right)$ (рис. 24). При $x_0 = -1$ значение $\varphi'(x)$ не существует (действительное), при $x_0 = 2$ $\left|\varphi'(x)\right| = \left|1.0801\right| > 1$ – критерий не удовлетворяется. При таком выражении $\varphi(x)$ искомый корень найти нельзя.

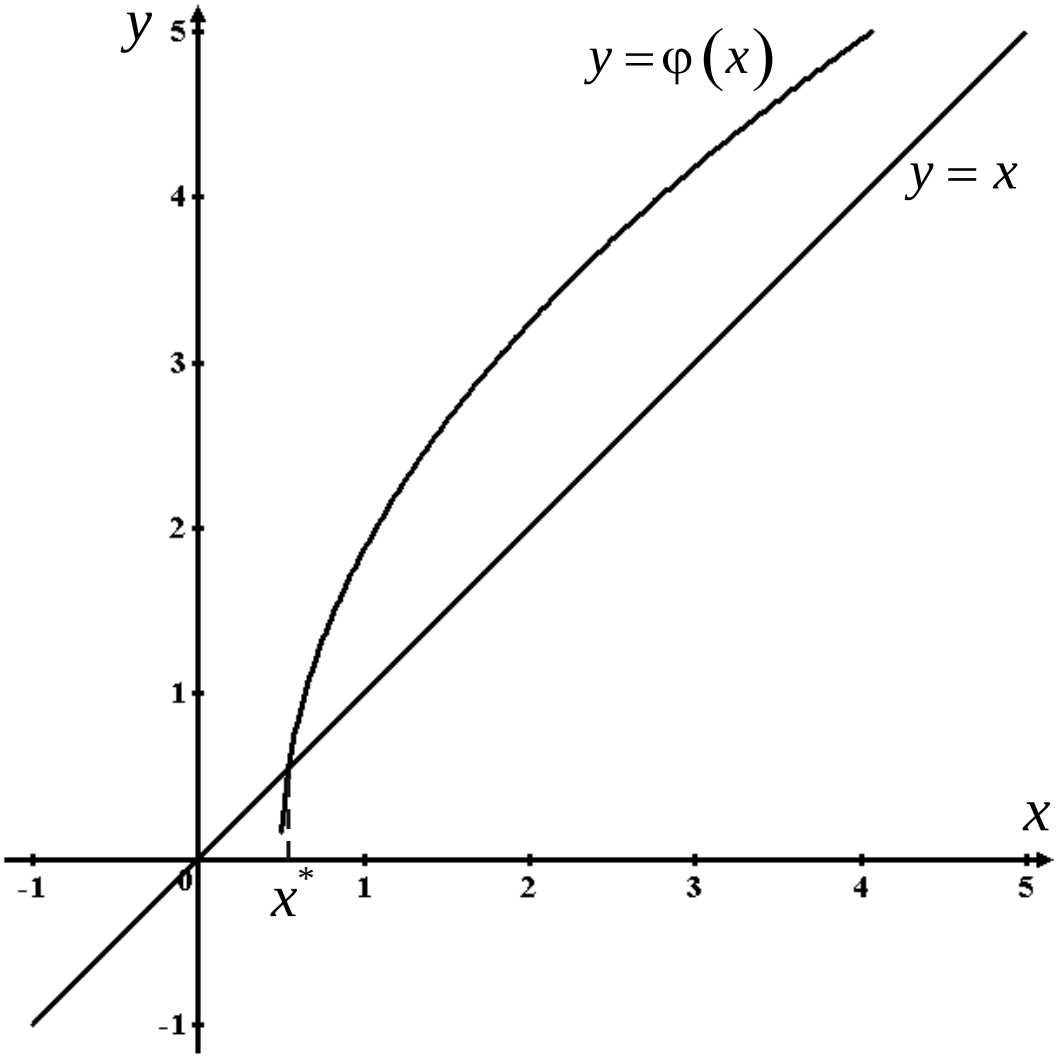

Рис. 24. График функции (28), x^* – корень уравнения (26), ближайший к началу координат

Для (29) имеем $\varphi'(x) = 0.4x - 0.4$ (рис. 25). При $x_0 = -1$ $\left|\varphi'(x)\right| = \left|-0.8\right| < 1$; при $x_0 = 2$ $\left|\varphi'(x)\right| = \left|0.4\right| < 1$. Обе точки подходят. Расчёт по программе с такой же точностью даёт тот же результат $5.41960 \cdot 10^{-1}$.

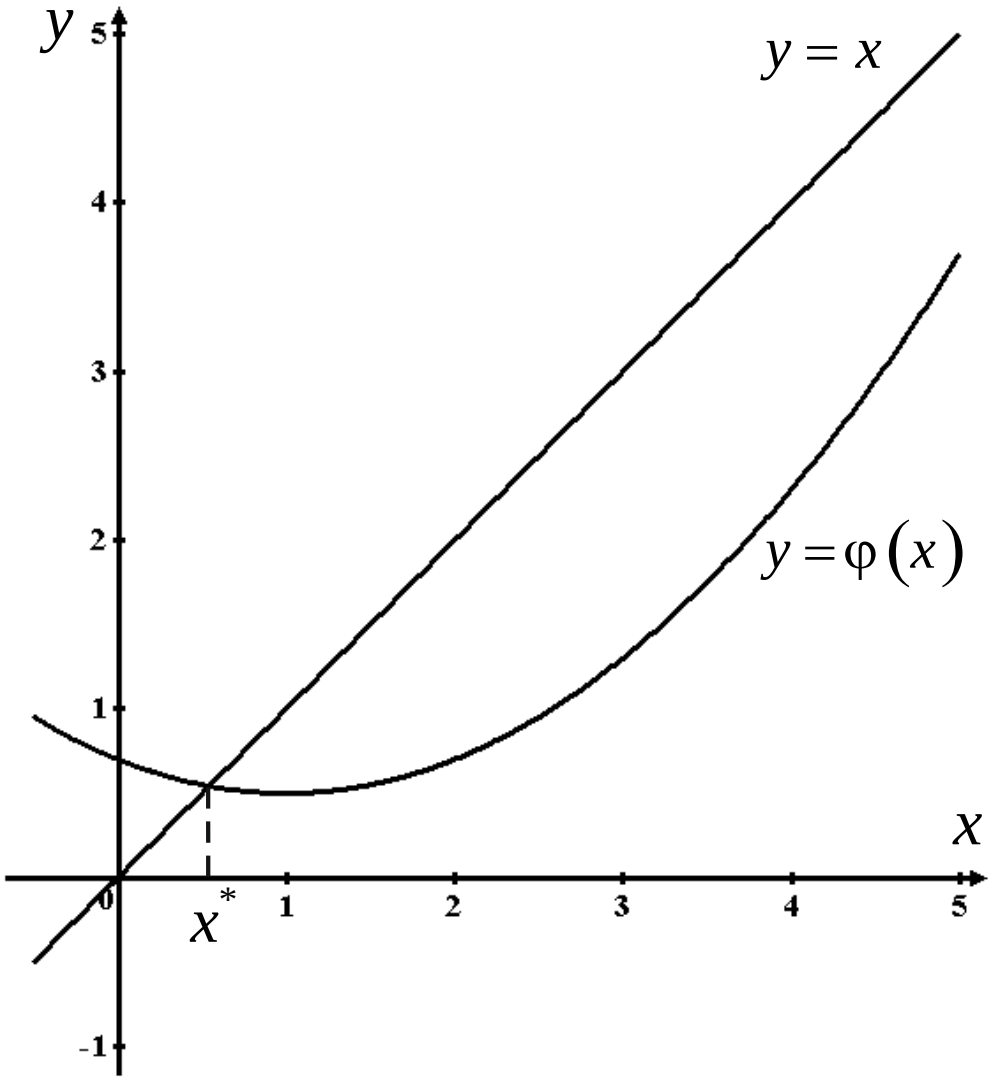

Рис. 25. График функции (29), x^* – корень уравнения (26), ближайший к началу координат

Рассмотрим более сложный пример с нахождением двух корней – уравнение (12), см. рис. 14.

Пример 1 (окончание). Для $\varphi(x)$ – (15), см. рис. 15, расчёт по программе для найденных начальных приближений (21) даёт: при $x_0 = 4$ $x^* = 2.958292$, при $x_0 = 1$ (для первого корня) получается то же самое значение, т. е. решение «скатывается» ко второму корню. Это весьма распространённая ситуация при нахождении нескольких корней. Таким образом для (15) можно найти только второй корень.

Для $\varphi(x)$ – (16), см. рис. 16, по (22) находится только первый корень при $x_0 = 1$ $x^* = 4.67560 \cdot 10^{-1}$.

Для $\varphi(x)$ – (17), см. рис. 16, по (23) при $x_0 = 1$ имеем $x^* = 4.67560 \cdot 10^{-1}$, а при $x_0 = 4$ (для второго корня) получается точно такое же значение, т. е. при данном выражении $\varphi(x)$ второй корень найти нельзя.

По сравнению с предыдущими двумя методами метод простых итераций значительно быстрее сходится. Недостатком является необходимость предварительного математического исследования. Осложняющий фактор – наличие нескольких, близко расположенных корней, когда даже соответствие критерию сходимости не гарантирует нахождение того или иного корня.

§ 8. Метод Ньютона (касательных)

В отличие от предыдущего в данном методе работа происходит с исходным уравнением (1). Итерационная формула выводится из формулы Тэйлора для функции одной переменной:

$$\boxed{f(x) = f(x_0) + (x - x_0) f'(x_0) + \frac{(x - x_0)^2}{2!} f''(x_0) + \ldots}. \quad (30)$$

Если ограничиться первыми двумя членами этого бесконечного ряда, то можно получить выражение

$$x = x_0 - \frac{f(x_0)}{f'(x_0)}, \quad (31)$$

из которого можно построить итерационную формулу, положив x_0 предыдущим, а x – последующим приближением корня. Также эту формулу можно получить из уравнения касательной, поэтому метод и имеет такое второе название (см. геометрическую интерпретацию) (рис. 26).

Теорема. *Пусть* $f(x)$ *определена и дважды дифференцируема на отрезке* $[a, b]$, *причём* $f(a) f(b) < 0$, *а производные* $f'(x)$ *и* $f''(x)$ *сохраняют знак на* $[a, b]$. *Тогда, исходя из начального приближения* $x_0 \in [a, b]$, *удовлетворяющего неравенству* $f(x_0) f''(x_0) > 0$, *можно построить последовательность* $x_{n+1} = x_n - f(x_n) / f'(x_n);\ n = 0, 1, 2, 3, \ldots$, *сходящуюся к единственному на* $[a, b]$ *решению уравнения* $f(x) = 0$.

Таким образом, **итерационная формула метода Ньютона**

$$\boxed{x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}; \quad n = 0, 1, 2, ...}, \tag{32}$$

где n – номер итерации.

Критерий сходимости:

$$\boxed{f(x_0) \cdot f''(x_0) > 0}, \tag{33}$$

где x_0 – начальное приближение.

Критерий окончания будет такой же, как в предыдущем методе:

$$\boxed{|x_{n+1} - x_n| < \varepsilon}, \tag{34}$$

где ε – заданная точность.

Метод Ньютона схож с методом простых итераций, но имеет свою итерационную формулу и критерий сходимости. Последний – весьма жёсткий и зачастую не выполняется. Однако из всех четырёх рассмотренных методов **метод Ньютона – самый оптимальный**, он обладает квадратичной сходимостью (если начальное приближение хорошее, т. е. на участке, где функция достаточно гладкая). По структуре программа метода будет точно такой, как в предыдущем методе, только требуется описание двух функций: $f(x)$, назовём её f(x), и $f'(x)$, назовём её f1(x). В предварительном математическом исследовании используется свой критерий сходимости (33), т. е. необходимо, чтобы функция $f(x)$ имела вторую производную. **Программа**:

```
program kasat;
//описание переменных eps, x и x0
var eps,x,x0: real;
//описание функций f(x) (например f) и f'(x) (например f),
//конкретный вид функций не приведён
function f(x:real):real;begin <…>end;
function f1(x:real):real;begin <…>end;
//начало основной программы
begin
//ввод исходных данных
write('x0=');readln(x); //ввод начального приближения в
```

```
//переменную x
write('eps=');readln(eps);
//итерационный цикл (например – repeat); в начале
//цикла – переприсваивание
repeat x0:=x;
//вычисление нового приближения по (32)
x:=x0–f(x0)/f1(x0);
//выход из цикла по критерию (34)
until abs(x-x0)<eps;
//вывод полученного результата – последнее приближение корня
writeln('x=',x);
end.
```

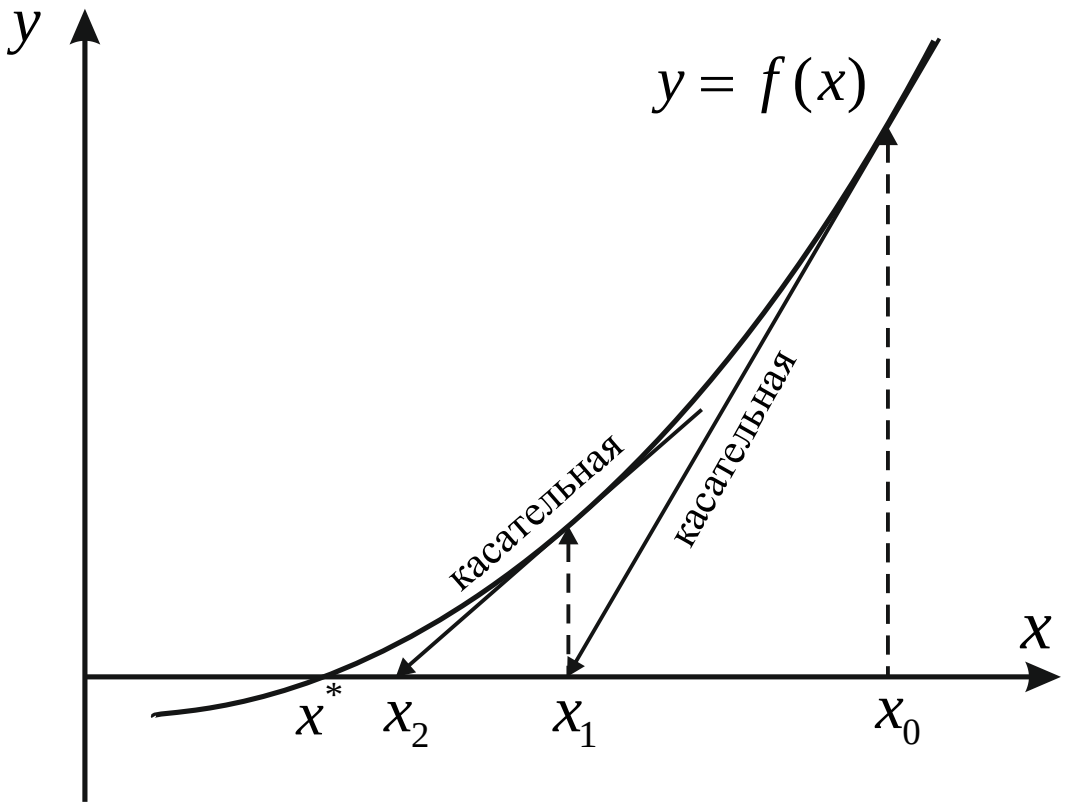

Рис. 26. Графическая интерпретация метода Ньютона. x^* – корень уравнения, x_0 – начальное приближение, x_1 и x_2 – первая и вторая итерация

Графическая интерпретация метода Ньютона (см. рис. 26). Если имеется начальное приближение x_0, то порядок построений таков: из x_0 восстанавливается перпендикуляр до пересечения с графиком функции и через полученную точку проводится касательная до пересечения с осью x и получается следующее приближение x_1.

Затем во второй итерации опять восстанавливается перпендикуляр и проводится касательная – приближение x_2, и т. д. Из рисунка видно, что уже за три итерации в данном случае можно получить корень с большой точностью.

Приведём примеры расчётов по методу Ньютона.

Пример 1 (уравнение (12), рис. 14). Первая и вторая производные:

$f'(x) = 1.6x - 2 - 1/x$ и $f''(x) = 1.6 + 1/x^2$.

Предварительное математическое исследование (для (13) и (14)):

при $x_0 = 0.0001$ $f(x_0) f''(x_0) = 9.2101 \cdot 10^8 > 0$ – выполняется;

при $x_0 = 1$ $f(x_0) f''(x_0) = -3.12 < 0$ – не выполняется;

при $x_0 = 2$ $f(x_0) f''(x_0) = -2.7622 < 0$ – не выполняется;

при $x_0 = 4$ $f(x_0) f''(x_0) = 3.4137 > 0$ – выполняется.

При точности $\varepsilon = 0.000001$:

первый корень: $4.6756 \cdot 10^{-1}$;

второй корень: 2.858293.

Пример 2 (уравнение (26), см. рис. 22). Первая и вторая производные:

$f'(x) = 0.4x - 1.4$ и $f''(x) = 0.4$.

Предварительное математическое исследование (для интервала $[-1, 2]$):

при $x_0 = -1$ $f(x_0) \cdot f''(x_0) = 0.92 > 0$ – выполняется;

при $x_0 = 2$ $f(x_0) \cdot f''(x_0) = -0.52 < 0$ – не выполняется.

При точности $\varepsilon = 0.000001$ корень: $5.4196 \cdot 10^{-1}$.

§ 9. Сравнительная характеристика методов

При выборе того или иного метода решения нелинейных уравнений требуется учёт всех их преимуществ и недостатков. Для дифференцируемых функций при возможности проведения предварительного математического исследования и когда нужна быстрая сходи-

мость решений (особенно при высокой задаваемой точности) предпочтение следует отдавать методам простых итераций или Ньютона. Когда предварительное математическое исследование невозможно или затруднительно, в частности, при использовании решения нелинейных уравнений как вспомогательной операции внутри других алгоритмов, а также для недифференцируемых функций, нужно использовать метод дихотомии или хорд.

Отдельную проблему представляет необходимость нахождения сразу нескольких, близко расположенных корней. Здесь может помочь комбинация различных методов.

Кроме того, при использовании метода простых итераций в решении конкретных задач химии и химической технологии следует помнить, что в математике при выводе расчётных формул и критериев оси абсцисс и ординат не имели никакой размерности. В реальных задачах эти оси всегда имеют разную размерность и масштаб, причём, очень часто – со степенью. Это необходимо учитывать и вносить коррективы в критерий сходимости (наклон касательной 45º).

Глава 2. АППРОКСИМАЦИЯ ФУНКЦИЙ. МЕТОД НАИМЕНЬШИХ КВАДРАТОВ

§ 1. Введение

Подавляющее большинство измерений в науке сводится к двум видам.

1. Измеряется **одна и та же величина** несколько раз (*n* измерений), поскольку каждое измерение обладает погрешностью и лишь из множества измерений можно восстановить объективную картину (рис. 27). В этом случае применимы известные формулы среднего значения и среднеквадратической погрешности, а также корректен статистический анализ по критерию Фишера и другим критериям.

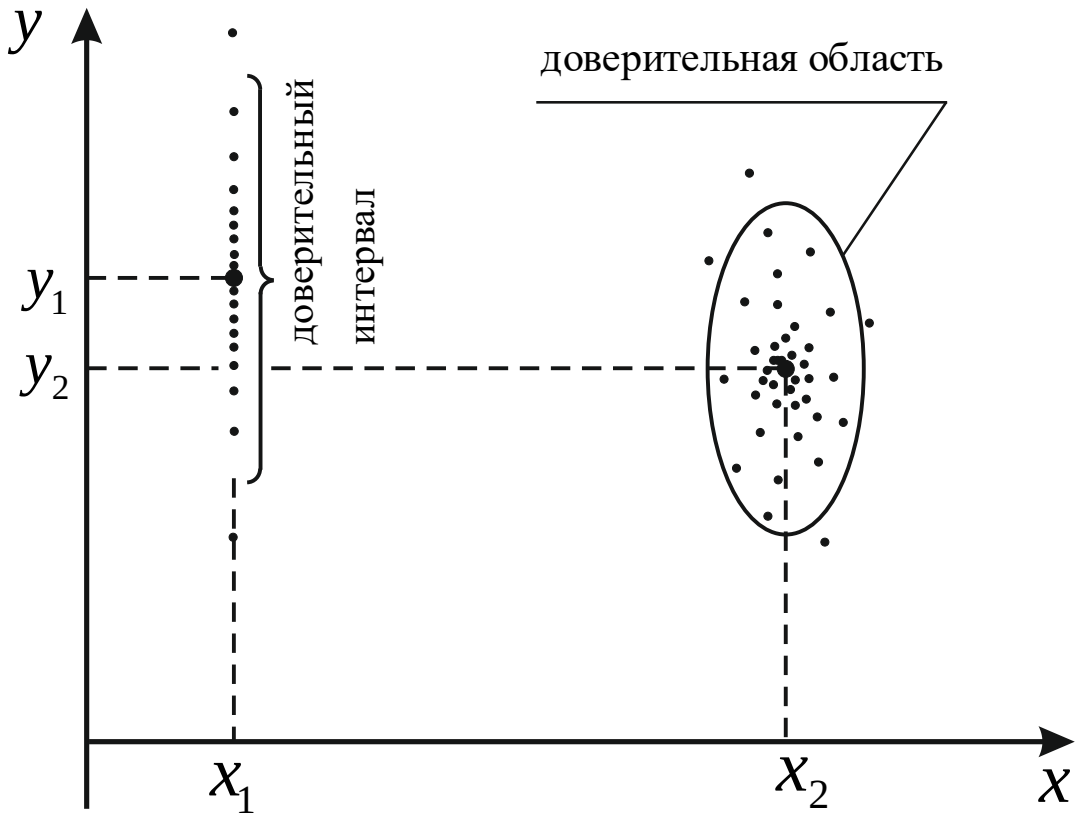

Рис. 27. Измерение одной величины несколько раз, y_1 и y_2 – измеряемые величины, x_1 – неизменный объект или фиксированный точный аргумент, x_2 – фиксированный аргумент, задаваемый с погрешностью, жирные точки – точные значения измеряемой величины, простые точки – отдельные измерения, доверительный интервал и доверительная область (например, – 90%-ные) – области, в которые попадают 90 % всех измерений

2. Снимается **функциональная зависимость** какой-то величины y от аргумента x, т. е. $y = f(x)$ (возможен также случай зависимости от нескольких аргументов). При этом, как правило, каждое значение аргумента в данном диапазоне измерений задаётся только один раз и снимается соответствующее ему значение функции, причём с определённой погрешностью. Ситуация усложняется тем, что значения аргумента могут быть заданы тоже лишь с какой-то точностью. Таким образом, мы имеем совершенно другую картину; здесь понятие среднего значения лишено смысла, нельзя также использовать обычные критерии статистической обработки, такие, как критерий Фишера. И если в первом случае целью является получение одной конкретной величины (т. е. – значения функции), то в случае снятия функциональной зависимости необходимо получить эту зависимость $y = f(x)$, как правило, в аналитическом виде. Разумеется, и там и там это можно сделать лишь с какой-то точностью. Восстановление вида функции на основе экспериментальных (или – других, имеющих погрешность) данных, называется **аппроксимацией** (приближением) **функции**. Искомая истинная (восстанавливаемая) зависимость обозначена на рис. 28 и 29 жирной штриховой линией, потому что совершенно точно её получить невозможно, а можно лишь построить математическими методами кривую, приближенную к ней, т. е. с погрешностью, которая будет зависеть от качества эксперимента, точности измерения x и y, дисперсии и других факторов.

Воспроизводимость эксперимента для первого вида измерений определяется сравнением нескольких независимых серий измерений по полученным значениям средних величин и **дисперсий** («размытостей» набора экспериментальных точек). Для второго вида измерений, как правило, весьма затруднительно проводить повторные опыты для каждого из значений аргументов, ведь они не могут быть заданы совершенно точно. На заре компьютерной эпохи в науке получили распространение **методы планирования эксперимента**, когда в опытном диапазоне аргументов выбиралось несколько точек и для них проводились параллельные опыты. Были разработаны соответствующие расчётные методики. Однако, помимо указанной уже трудности точного задания аргумента несколько раз, такой подход подразумевал, что в экспериментальном интервале поведение функции хорошо известно, и

она не меняет свой характер, что не всегда соответствовало действительности. Со временем в науке возобладал другой подход: **автоматизация измерений и компьютеризация** их обработки позволила «набирать статистику» за счёт многократного увеличения количества опытов; например, вместо десятка трудоёмких измерений внутри экспериментального диапазона появилась возможность снять сотни и тысячи точек. При этом не было никакой необходимости задавать один и тот же аргумент несколько раз. **Воспроизводимость** эксперимента при таком подходе может быть проверена по результатам обработки нескольких независимых серий внутри одного и того же экспериментального интервала аргументов.

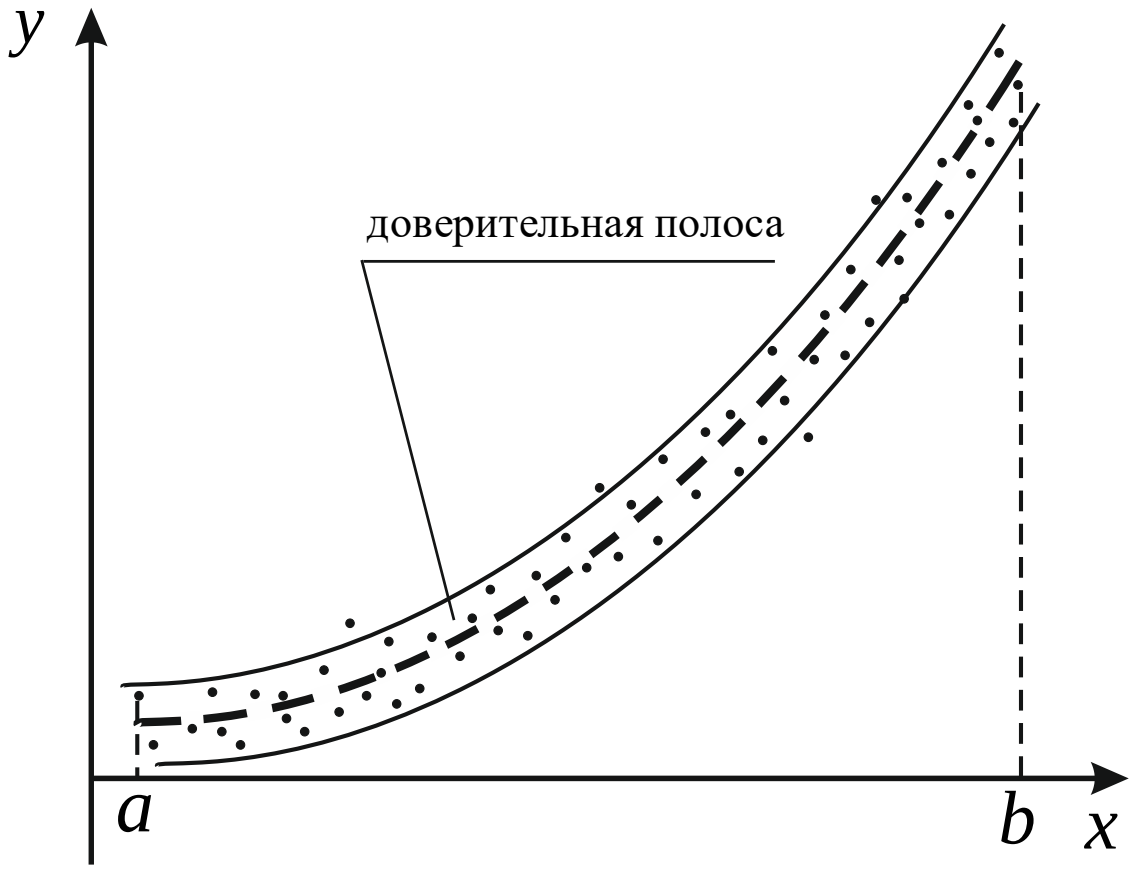

Рис. 28. Снятие экспериментальной зависимости, $[a, b]$ – экспериментальный диапазон измерений по аргументу x. Жирная штриховая линия – искомая истинная зависимость. Простые точки – отдельные измерения. Тонкими линиями выделена доверительная полоса (например, – 95%-ная) при постоянной абсолютной погрешности измерения функции y по всему диапазону

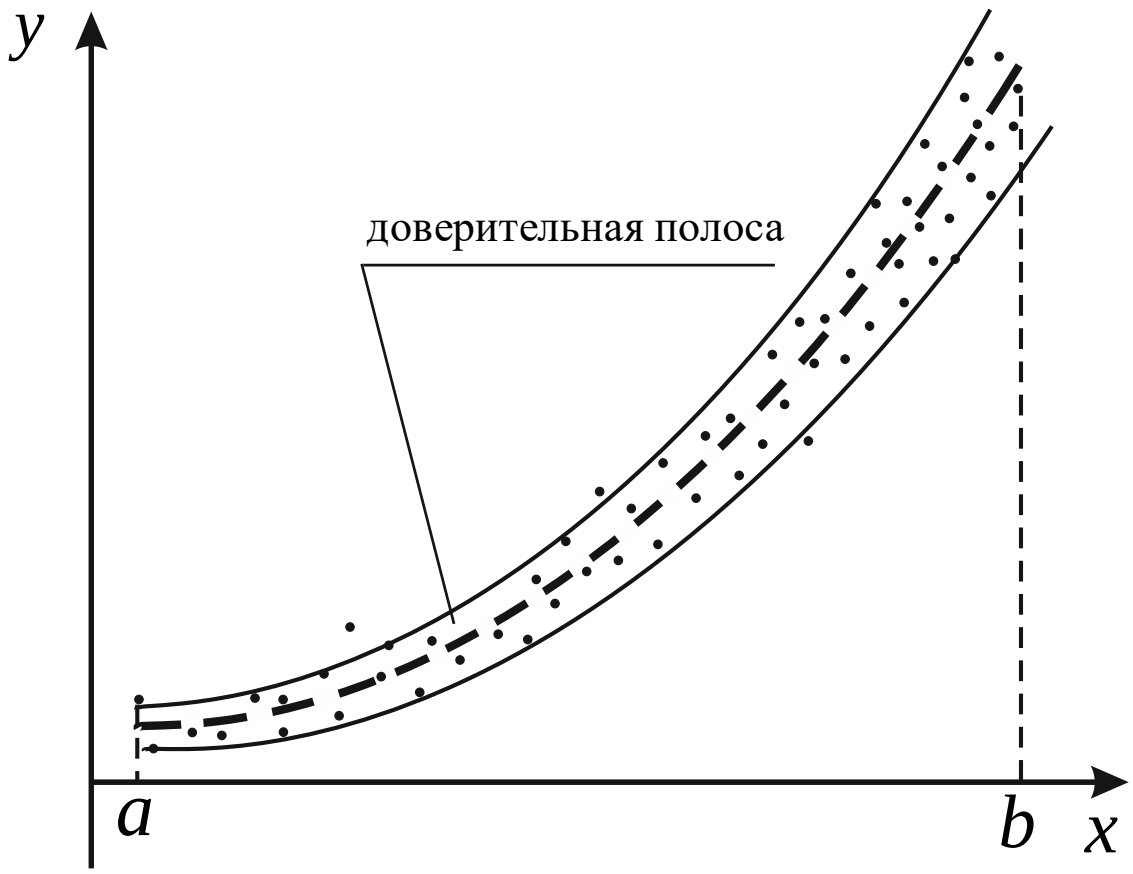

Рис. 29. Снятие экспериментальной зависимости, $[a, b]$ – экспериментальный диапазон измерений по аргументу x. Жирная штриховая линия – искомая истинная зависимость. Простые точки – отдельные измерения. Тонкими линиями выделена доверительная полоса (например, – 95%-ная) при постоянной относительной погрешности измерения функции y по всему диапазону (абсолютная погрешность при этом увеличивается)

В методах восстановления функциональной зависимости (аппроксимации функций), как правило, считается, что аргумент задаётся точно, а функция измеряется с какой-то погрешностью. Учёт неточности задания аргумента в этих методах возможен, при этом повышается громоздкость формул, но никаких принципиальных сложностей не возникает.

Одним из самых распространённых методов аппроксимации является **метод наименьших квадратов**. Для простоты будем считать, что x в экспериментах задаётся точно.

§ 2. Метод наименьших квадратов

Итак, в экспериментальном диапазоне [a, b] мы имеем экспериментальные точки (n опытов, табл. 1), которые отражают искомую истинную зависимость, обозначенную жирной штриховой линией (рис. 30). Необходимо построить математическими методами кривую $y = f(x)$, приближенную к этой истинной (покажем её сплошной тонкой линией). Полностью эти кривые никогда не будут совпадать. Для вывода теоретических формул предположим, что тонкая кривая уже имеется. На рисунке она заметно удалена от истинной, это сделано для наглядности; в реальных ситуациях эти две кривые могут проходить очень близко.

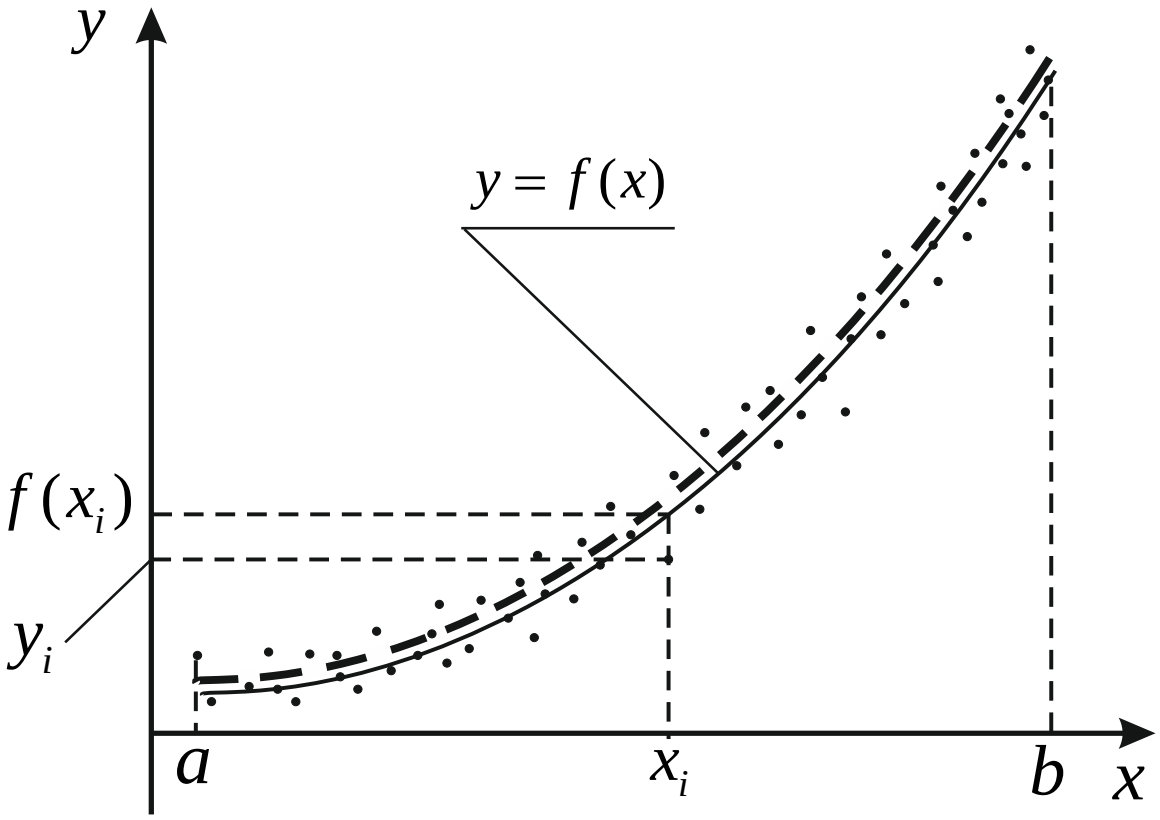

Рис. 30. Экспериментальные точки в диапазоне [a, b] отражают истинную зависимость (жирная штриховая линия). Тонкая кривая – зависимость $y = f(x)$, которая будет построена. Для i-ой экспериментальной точки x_i имеем два значения функции: y_i и $f(x_i)$

Таблица 1. Таблица экспериментальных данных снятия зависимости $y = f(x)$; всего n экспериментальных точек

x	x_1	x_2	x_3	…	x_{n-1}	x_n
y	y_1	y_2	y_3	…	y_{n-1}	y_n

Если рассмотреть какую-то i-ую экспериментальную точку x_i (см. рис. 30), то ей на рисунке будут соответствовать два значения функции – экспериментальный y_i (из таблицы) и $f(x_i)$ на кривой, которую предстоит построить. **Невязкой** называется разность

$$\boxed{f(x_i) - y_i - \text{невязка}}. \qquad (35)$$

Она может быть определена для любой экспериментальной точки и будет разной. Как видно из рис. 30, невязка имеет знак: если экспериментальная точка лежит выше кривой $y = f(x)$ (которая будет построена), то невязка отрицательная, если – ниже, то положительная.

Совершенно очевидно, что по длинам невязок всех экспериментальных точек можно судить о том, насколько оптимально мы построили кривую $y = f(x)$. На рис. 31 показаны примеры неудачных аппроксимаций. Кривая 1 вообще лежит вне области экспериментальных точек и все невязки будут очень большой величины (тем не менее, это – аппроксимация, ведь возможен ещё более худший вариант). Для кривой 2 некоторые экспериментальные точки подходят близко к ней и имеют малые длины невязок, но большинство точек – с большими невязками. Таким образом, необходимо суммировать длины невязок всех экспериментальных точек. При этом возникает проблема знака невязки; в математике избавиться от знака можно двумя способами – взять модуль или возвести в квадрат. Как мы потом увидим, чисто математически выгоден второй вариант, при дальнейших действиях потребуется дифференцирование, а квадрат дифференцируется хорошо.

Итак, нужно возвести в квадрат все невязки и просуммировать их по всем точкам. Поскольку любой эксперимент имеет погрешность, эта сумма никогда не будет нулевой или близкой к нулю, но для наиболее оптимальной аппроксимации она должна быть минимальной:

$$\boxed{\sum_i \left(f(x_i) - y_i \right)^2 \to \min}. \qquad (36)$$

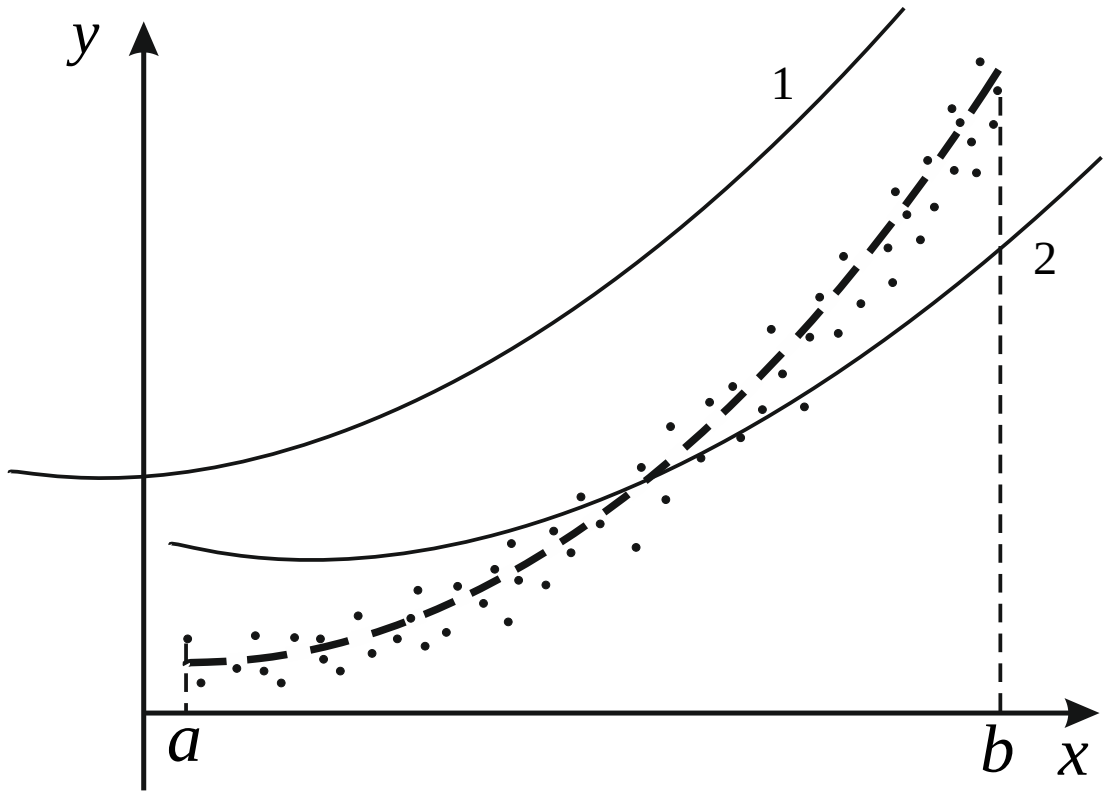

Рис. 31. Примеры неудачных аппроксимаций. Кривая 1 лежит вне экспериментального диапазона, кривая 2 имеет большие невязки для большинства точек

Это **принцип наименьших квадратов**, который лежит в основе **метода наименьших квадратов**.

Для дальнейшего анализа необходимо каким-то образом определить функцию $y = f(x)$ в **общем виде**. (Определение конкретного вида этой функции означало бы решение задачи аппроксимации). Общий вид функции предполагает собой определённый аналитический вид с какими-то неизвестными параметрами и коэффициентами. Например, общий экспоненциальный вид

$$y = Ae^{Bx} \tag{37}$$

имеет параметры (константы) A – предэкспоненциальный множитель, и B – множитель в показателе степени. Если для A и B известны значения, то функция (37) становится конкретной, т. е. если бы мы могли какую-то функцию в общем виде (например – (37)) подставить в (36) вместо $f(x_i)$, то задача аппроксимации свелась бы к нахождению ко-

эффициентов *A* и *B*. А это, в свою очередь, является тривиальной задачей математики. Таким образом, решением задачи аппроксимации в данном случае был бы конкретный вид функции (37).

Однако проблема состоит в том, что в подавляющем большинстве случаев общий вид предполагаемой зависимости $y = f(x)$ неизвестен. На помощь приходит тот факт, что любую функцию можно разложить в бесконечный ряд Тэйлора (30). Если разложение проводится в окрестности нуля ($x_0 = 0$), то ряд превращается в ряд Маклорена, который можно свести к **полиному степени** m, если заменить производные и факториалы коэффициентами a_i и ограничить количество членов ряда числом m. Тогда функция $y = f(x)$ примет вид

$$\boxed{y = a_0 + a_1 x + a_2 x^2 + ... + a_m x^m}. \qquad (38)$$

Если теперь вместо $f(x_i)$ в (37) подставить (38), причём *x* получат индексы i, то мы будем иметь **полиномиальную аппроксимацию**, **принцип наименьших квадратов** для которой запишется

$$\boxed{\sum_i \left(a_0 + a_1 x_i + a_2 x_i^2 + ... + a_m x_i^m - y_i\right)^2 \to \min}. \qquad (39)$$

Таким образом, в случае полиномиальной аппроксимации задача сводится к нахождению коэффициентов a_i.

Но при этом возникает ещё одна проблема: заранее неизвестно, сколько членов полиномиальной функции (38) нужно брать, чтобы получить хорошую аппроксимацию. Поэтому последовательно проводят **несколько аппроксимаций**, с разным количеством членов ряда, а потом сравнивают полученные результаты. Например, можно сделать аппроксимации (см. (38)) для линейной функции $(m = 1)$ – **линейная аппроксимация**, квадратного трёхчлена $(m = 2)$ – **квадратичная аппроксимация**, для кубической параболы $(m = 3)$, и так далее. Наглядно это продемонстрировано на рис. 32 и 33, где показаны линейная и квадратичная аппроксимации для разного набора данных. Из рисунков даже визуально видно, какая аппроксимация лучше. Разумеется, на практике применяют гораздо более строгие, аналитические, критерии. О них будет сказано позднее.

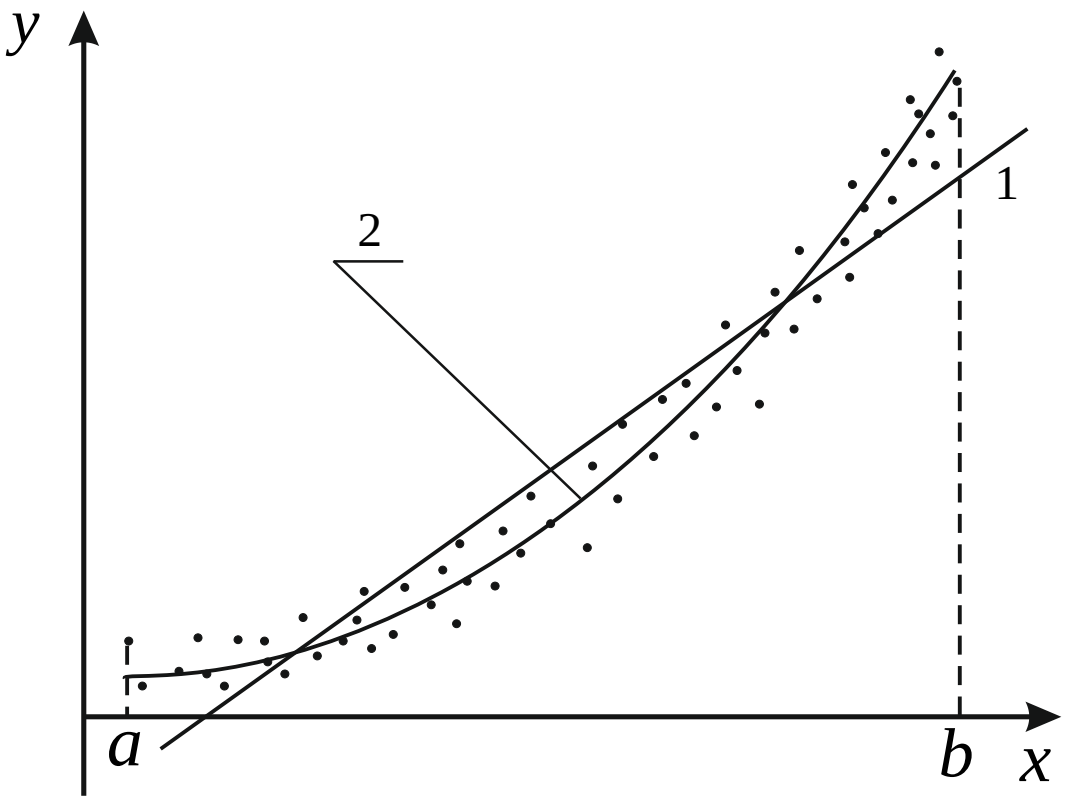

Рис. 32. Аппроксимация набора экспериментальных данных линейной функцией 1 и квадратным трёхчленом 2. Более оптимальна аппроксимация 2

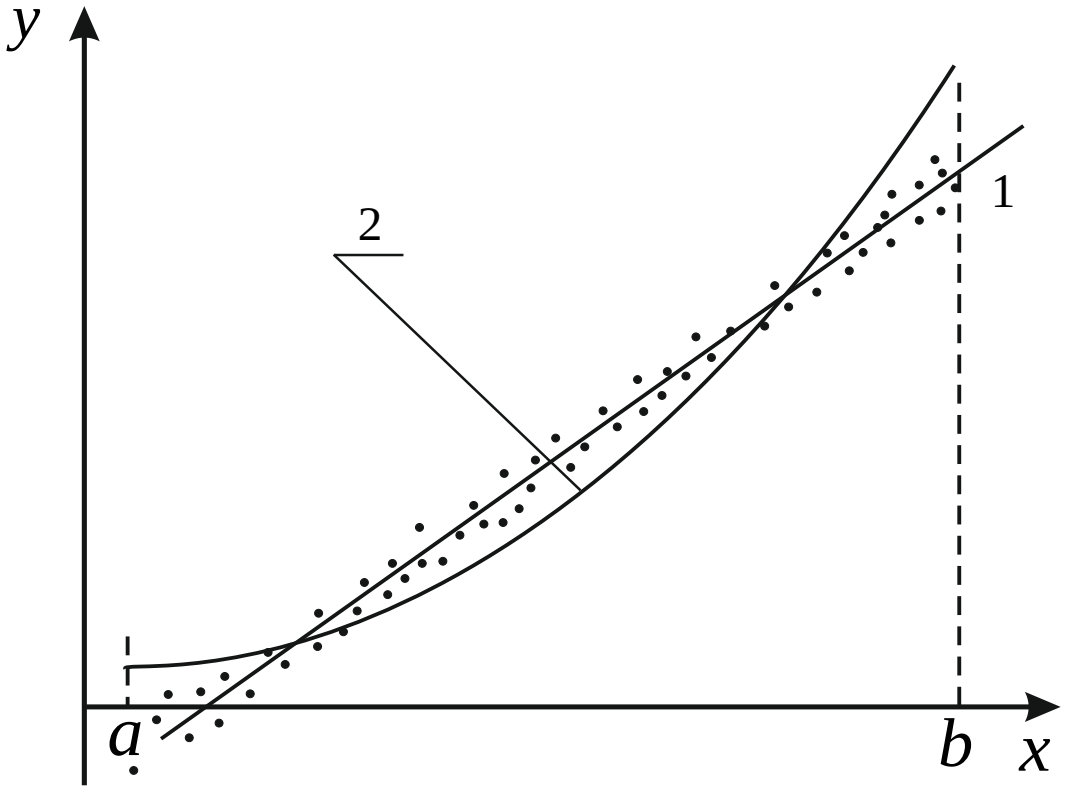

Рис. 33. Аппроксимация набора экспериментальных данных линейной функцией 1 и квадратным трёхчленом 2. Более оптимальна аппроксимация 1

§ 3. Решение задачи полиномиальной аппроксимации

С точки зрения математики принцип наименьших квадратов в левой своей части представляет собой функцию нескольких переменных a_i, которые являются **неизвестными** и искомыми (в нашем случае x_i и y_i **известны** из эксперимента). Обозначим эту функцию Ф, тогда

$$\Phi(a_0, a_1, a_2, ..., a_m) \to \min, \tag{40}$$

где m – степень полинома.

Как известно, функция нескольких переменных стремится к экстремуму (в данном случае – к минимуму), когда частные производные по всем аргументам будут равны нулю:

$$\begin{cases} \left(\dfrac{\partial \Phi}{\partial a_0}\right) = 0 \\ \left(\dfrac{\partial \Phi}{\partial a_1}\right) = 0 \\ ... \\ \left(\dfrac{\partial \Phi}{\partial a_m}\right) = 0. \end{cases} \tag{41}$$

Таким образом, мы получаем **систему уравнений**, в которой количество уравнений (m + 1) будет равно количеству неизвестных a_i. Такую систему можно решить, т. е. нужно продифференцировать (39), составить систему уравнений и решить её.

$$\begin{cases} \dfrac{\partial}{\partial a_0}\left(\sum_i \left(a_0 + a_1 x_i + a_2 x_i^{\,2} + ... + a_m x_i^{\,m} - y_i\right)^2\right) = 0, \\ \dfrac{\partial}{\partial a_1}\left(\sum_i \left(a_0 + a_1 x_i + a_2 x_i^{\,2} + ... + a_m x_i^{\,m} - y_i\right)^2\right) = 0, \\ ... \\ \dfrac{\partial}{\partial a_m}\left(\sum_i \left(a_0 + a_1 x_i + a_2 x_i^{\,2} + ... + a_m x_i^{\,m} - y_i\right)^2\right) = 0. \end{cases} \tag{42}$$

Учитывая то, что производная суммы равна сумме производных и используя правило дифференцирования сложных функций, из (42) имеем

$$\begin{cases} \sum_i \left(2\left(a_0 + a_1 x_i + a_2 x_i^{\,2} + \ldots + a_m x_i^{\,m} - y_i\right)1\right) = 0, \\ \sum_i \left(2\left(a_0 + a_1 x_i + a_2 x_i^{\,2} + \ldots + a_m x_i^{\,m} - y_i\right)x_i\right) = 0, \\ \ldots \\ \sum_i \left(2\left(a_0 + a_1 x_i + a_2 x_i^{\,2} + \ldots + a_m x_i^{\,m} - y_i\right)x_i^{\,m}\right) = 0. \end{cases} \quad (43)$$

Двойки выносятся перед суммами и сокращаются. Скобки умножаются на множители и знаки суммы вносятся в скобки:

$$\begin{cases} \sum_i (a_0) + \sum_i (a_1 x_i) + \sum_i \left(a_2 x_i^{\,2}\right) + \ldots + \sum_i \left(a_m x_i^{\,m}\right) - \sum_i (y_i) = 0, \\ \sum_i (a_0 x_i) + \sum_i \left(a_1 x_i^{\,2}\right) + \sum_i \left(a_2 x_i^{\,3}\right) + \ldots + \sum_i \left(a_m x_i^{\,m+1}\right) - \sum_i (y_i x_i) = 0, \\ \ldots \\ \sum_i \left(a_0 x_i^{\,m}\right) + \sum_i \left(a_1 x_i^{\,m+1}\right) + \sum_i \left(a_2 x_i^{\,m+2}\right) + \ldots + \sum_i \left(a_m x_i^{\,m+m}\right) - \sum_i \left(y_i x_i^{\,m}\right) = 0. \end{cases}$$

Находящиеся под суммами a_j (j – индекс коэффициентов полинома (38)) от i не зависят, поэтому их можно вынести перед знаками суммы; отрицательные члены перенесём направо.

$$\begin{cases} a_0 \sum_i (1) + a_1 \sum_i x_i + a_2 \sum_i x_i^{\,2} + \ldots + a_m \sum_i x_i^{\,m} = \sum_i y_i, \\ a_0 \sum_i x_i + a_1 \sum_i x_i^{\,2} + a_2 \sum_i x_i^{\,3} + \ldots + a_m \sum_i x_i^{\,m+1} = \sum_i (y_i x_i), \\ \ldots \\ a_0 \sum_i x_i^{\,m} + a_1 \sum_i x_i^{\,m+1} + a_2 \sum_i x_i^{\,m+2} + \ldots + a_m \sum_i x_i^{\,m+m} = \sum_i \left(y_i x_i^{\,m}\right). \end{cases} \quad (44)$$

Получается так называемая **нормальная система уравнений** (44). В случае полиномиальной аппроксимации она представляет собой **систему линейных уравнений**. Неизвестными этой системы являются a_j, а соответствующие суммы – известными коэффициентами (x_i и y_i – экспериментальные данные). Для дальнейшего составления программы определим, что a_j – вектор (одномерный массив), обозначим его **a**, элементы массива с индексом j меняются от 0 до m (это степень полинома; $m+1$ – число уравнений в (44) и число неизвестных a_j);

коэффициенты в левой части (44), т. е. суммы, представляют собой матрицу (двумерный массив) $(m+1)\times(m+1)$, обозначим её **b**; свободные члены в правой части – вектор такой же размерности, как и **a**, обозначим его **c**.

Необходимо отметить, что сумма единиц от 1 до n в системе (44) равна просто n.

Если обозначить k – номер строки в (44), $k=0,1,...,m$, а j – номер столбца, $j=0,1,...,m$, то в общем виде нормальную систему можно записать

$$\boxed{\sum_{j=0}^{m}\left(a_j\sum_{i=1}^{n}x_i^{\,j+k}\right)=\sum_{i=1}^{n}\left(y_i x_i^{\,k}\right)}. \qquad (45)$$

Для наглядности представим систему (44) для $m=3$ (кубическая парабола)

$$\begin{cases} a_0\cdot n+a_1\sum_i x_i+a_2\sum_i x_i^{\,2}+a_3\sum_i x_i^{\,3}=\sum_i y_i, \\ a_0\sum_i x_i+a_1\sum_i x_i^{\,2}+a_2\sum_i x_i^{\,3}+a_m\sum_i x_i^{\,4}=\sum_i\left(y_i x_i\right), \\ a_0\sum_i x_i^{\,2}+a_1\sum_i x_i^{\,3}+a_2\sum_i x_i^{\,4}+a_m\sum_i x_i^{\,5}=\sum_i\left(y_i x_i^{\,2}\right), \\ a_0\sum_i x_i^{\,3}+a_1\sum_i x_i^{\,4}+a_2\sum_i x_i^{\,5}+a_m\sum_i x_i^{\,6}=\sum_i\left(y_i x_i^{\,3}\right). \end{cases} \qquad (46)$$

Анализ (46) показывает, что в левой части на диагоналях, перпендикулярных диагоналям матрицы **b**, располагаются **одинаковые элементы**; систему для $m=2$ можно получить из имеющейся вычёркиванием последней строки и последнего столбца левой части (и так далее). Кроме того, необходимо отметить, что левая часть не содержит y_i, т. е. определяется только набором экспериментальных значений x_i.

Методы решения систем уравнений (линейных и нелинейных) представляют собой отдельный раздел вычислительной математики; в соответствии с этим решение системы (44) будет рассмотрено в следующей главе данного пособия.

Если предположить, что эта система уже решена, – относительно a_j, то можно довести аппроксимацию до конца. Это будет означать, что мы получили **аппроксимирующее уравнение** вида (38). Поскольку

мы проводили последовательно несколько аппроксимаций (например, для $m=1,\ m=2$ и $m=3$), получится несколько аппроксимирующих уравнений, для нашего примера

$$\begin{gathered} y = a_0 + a_1 x; \\ y = a_0 + a_1 x + a_2 x^2; \\ y = a_0 + a_1 x + a_2 x^2 + a_3 x^3, \end{gathered} \tag{47}$$

в которых все a_j будут иметь конкретные числовые значения. Далее нужно провести анализ соответствия полученных уравнений экспериментальным данным и выбрать наилучшую аппроксимацию. Анализ заключается в расчёте значений функций $f(x_i)$ по аппроксимирующему уравнению для каждого экспериментального x_i и вычислении среднеквадратической погрешности по формуле

$$\boxed{S = \alpha\sqrt{\frac{\sum_i \left(y_i - f(x_i)\right)^2}{n(n-1)}}}, \tag{48}$$

где α – коэффициент Стьюдента для числа точек n. Это делается для каждого уравнения (например, (47)); лучшая аппроксимация будет иметь наименьшую S.

На практике считается, что проводить аппроксимации со степенью полинома выше $m=5$ нецелесообразно. Теоретически, если задать степень полинома, равную числу экспериментальных точек, то аппроксимирующая кривая (неправильного периодического вида) пройдёт точно через все эти точки, что не будет соответствовать действительности, так как невязки должны иметь место, т. е. количество экспериментальных точек должно быть много больше степени полинома.

Здесь необходимо сделать **важное замечание**, которое касается **всех видов аппроксимации**. В результате расчётов коэффициенты в уравнениях (47) получаются с очень большим количеством значащих цифр (более 20, что обеспечивает компьютер). Разумеется, это не соответствует сравнительно малой (как правило, не выше 2 – 3 значащих цифр) точности определения экспериментальных данных. Тем не менее, для вычисления значения функции нужно использовать полученные коэффициенты именно в таком виде, а округлять лишь **конечный**

результат (значение y) в соответствии с погрешностями экспериментальных x_i и y_i, основываясь на расчёте среднеквадратического отклонения по (48). В противном случае, если округлить значения коэффициентов a_j, может сильно возрасти погрешность определения значений y. В полученном среднеквадратическом отклонении оставляют 1 – 2 значащие цифры и округляют рассчитанное значение y в соответствии с этим.

§ 4. Программа расчёта коэффициентов системы и свободных членов

Поскольку решение систем линейных уравнений представляет собой отдельную задачу и будет рассмотрено в следующей главе, составим программу, в которой рассчитываются коэффициенты и свободные члены нормальной системы уравнений. А программа решения этой системы будет дана в следующей главе. Результаты работы первой программы выведем в бинарный файл (а также – на экран и в текстовый файл, для контроля), из которого вторая программа будет считывать информацию.

Таким образом, в программе подготовки нормальной системы (первая программа) будут фигурировать двумерный массив **b**, одномерный – **c**, а также одномерные массивы экспериментальных данных **x** и **y**. Массива коэффициентов аппроксимирующего уравнения **a** не будет (хотя он входит в нормальную систему). Расчёт **b** и **c** лучше проводить по строкам, т. е. во внешнем цикле задаётся номер строки (индекс k), затем во внутреннем цикле (по j) перебираются все столбцы, далее здесь же, не закрывая цикл по k, рассчитывается свободный член этой строки, и только после этого закрывается цикл по строкам (k). Вывод **b** производится после внутреннего цикла, а вывод **c** – перед завершением внешнего. В программе должен быть предусмотрен механизм работы с переменной размерностью массивов, проще всего – заданием их размерности в подразделе **const** (более рациональный алгоритм можно сделать с использованием динамической памяти). Размерность массивов можно изменить, задав другое значение констант n и m в **одном месте** программы (при этом потребуется перекомпиляция). (В настоящем пособии такой способ будет использоваться везде).

```
program normsyst;
//описание числа экспериментальных точек n и
//степени полинома m:
const n=50;m=3;
//описание массивов: b – коэффициенты системы,
//c – свободные члены, x и y – экспериментальные точки:
var b:array[0..m,0..m] of real;c:array[0..m] of real;
x,y: array[1..n] of real;
i,j,k:integer; //описание счётчиков цикла i, j, k
//описание файловых переменных для текстового – fpt,
//и бинарного – fpb, файлов, id – исходные данные:
id:text;fpt:text;fpb:file of real;
//начало основной программы
begin   //ввод исходных данных из текстового файла 'id1.txt':
assign(id,'id1.txt');reset(id);
for i:=1 to n do begin readln(id,x[i]);readln(id,y[i]);end;close(id);
//вариант ввода исходных данных с клавиатуры:
//for i:=1 to n do begin
//write('x[',i,']=');readln(x[i]); write('y[',i,']=');readln(y[i]);end;
//инициализация работы с файлами ft.txt и fb.bin:
assign(fpt,'ft.txt');rewrite(fpt);assign(fpb,'fb.bin'); rewrite (fpb);
//расчёт и вывод коэффициентов b[k,j] и свободных членов c[k];
//внешний цикл по строкам, индекс k:
for k:=0 to m do begin
//внутренний цикл по столбцам, индекс j:
for j:=0 to m do begin
//накопление суммы b[k,j] в цикле по i, см. (45):
b[k,j]:=0; for i:=1 to n do b[k,j]:=b[k,j]+exp((j+k)*ln(x[i]));
//вывод b[k,j] на экран, в текстовый и бинарный файл:
writeln('b[',k,',',j,']=',b[k,j]); writeln(fpt,'b[',k,',',j,']=',b[k,j]);
write(fpb,b[k,j]);end;
//расчёт c[k] в этой же строке (накопление суммы), см. (45):
c[k]:=0; for i:=1 to n do c[k]:=c[k]+y[i]*exp(k*ln(x[i]));
//вывод c[k] на экран, в текстовый и бинарный файл:
writeln('c[',k,']=',c[k]); writeln(fpt,'c[',k,']=',c[k]);write(fpb,c[k]);
//закрытие цикла по строкам (индекс k):
end;    //закрытие файлов и конец программы:
close(fpt); close(fpb);end.
```

Примечание. 1. Размерность массивов **b**, **c**, **x** и **y** можно изменить, задав другие значения констант n и m (с перекомпиляцией). 2. Коэффициенты **b** и свободные члены **c** выводятся в бинарный файл по строкам. Считывание информации из этого файла (другой программой) должно быть в том же порядке. 3. Так как в программе не использован математический модуль, при расчёте **b** и **c** по формуле (45) для взятия степени **x** используется формула с логарифмом, поэтому значения **x** должны быть положительными. При использования программы для отрицательных значений аргумента требуется пересчёт значений **x** (смещение в положительную область), а затем, после проведения аппроксимации, – обратный пересчёт.

§ 5. Неполиномиальная аппроксимация

Как уже отмечалось (§ 3), в некоторых случаях заранее известен общий вид аппроксимирующей функции, например – экспоненциальный (37). Тогда можно провести неполиномиальную аппроксимацию, которая, как правило, проще и даёт более удобный результат. В общий вид таких функций входят коэффициенты и параметры (в нашем примере – A и B), к определению которых и будет сводиться задача аппроксимации.

Рассмотрим этот пример до конца. Подставим (37) в принцип наименьших квадратов (36):

$$\sum_i \left(A\exp(Bx_i) - y_i \right)^2 \to \min \tag{49}$$

– принцип наименьших квадратов для экспоненциальной функции. Используя (41), получаем систему уравнений

$$\begin{cases} \dfrac{\partial}{\partial A}\left(\sum\limits_i \left(A\exp(Bx_i) - y_i \right)^2 \right) = 0, \\ \dfrac{\partial}{\partial B}\left(\sum\limits_i \left(A\exp(Bx_i) - y_i \right)^2 \right) = 0. \end{cases} \tag{50}$$

По аналогии с (43) имеем

$$\begin{cases} \sum\limits_i \left(2\left(A\exp(Bx_i) - y_i \right)\exp(Bx_i) \right) = 0, \\ \sum\limits_i \left(2\left(A\exp(Bx_i) - y_i \right) Ax_i \exp(Bx_i) \right) = 0. \end{cases} \tag{51}$$

После преобразований получаем

$$\begin{cases} \sum_i \left(A\exp^2(Bx_i) \right) = \sum_i \left(y_i \exp(Bx_i) \right), \\ \sum_i \left(A^2 x_i \exp^2(Bx_i) \right) = \sum_i \left(y_i A x_i \exp(Bx_i) \right). \end{cases}$$

$$\begin{cases} A\sum_i \left(\exp^2(Bx_i) \right) = \sum_i \left(y_i \exp(Bx_i) \right), \\ A^2 \sum_i \left(x_i \exp^2(Bx_i) \right) = A\sum_i \left(y_i x_i \exp(Bx_i) \right). \end{cases}$$

Окончательно

$$\begin{cases} A\sum_i \left(\exp(2Bx_i) \right) = \sum_i \left(y_i \exp(Bx_i) \right), \\ A\sum_i \left(x_i \exp(2Bx_i) \right) = \sum_i \left(y_i x_i \exp(Bx_i) \right). \end{cases} \quad (52)$$

Это система из двух нелинейных уравнений с двумя неизвестными (аналог нормальной системы уравнений). Почленно делим уравнения друг на друга:

$$\frac{\sum_i \left(\exp(2Bx_i) \right)}{\sum_i \left(x_i \exp(2Bx_i) \right)} - \frac{\sum_i \left(y_i \exp(Bx_i) \right)}{\sum_i \left(y_i x_i \exp(Bx_i) \right)} = 0. \quad (53)$$

Уравнение (53) – нелинейное с одним неизвестным (B), оно может быть решено численными методами. После определения B можно вычислить A напрямую по формуле

$$A = \frac{\sum_i \left(y_i \exp(Bx_i) \right)}{\sum_i \left(\exp(2Bx_i) \right)} \quad (54)$$

или

$$A = \frac{\sum_i \left(y_i x_i \exp(Bx_i) \right)}{\sum_i \left(x_i \exp(2Bx_i) \right)}.$$

Эти формулы получены из (52). Таким образом, здесь нет необходимости решать систему уравнений.

В качестве **примера**, зададим значения констант в уравнении (37), например A = 1.5 и B = 0.25. Для нескольких точек (например, n = 20) в диапазоне $[-1, 5]$ рассчитаем значения функции (37) и изменим их с помощью рандомизации (т. е. зададим случайные отклонения). Таким образом будет смоделирован экспериментальный разброс точек. Полученные точки представлены в табл. 2 (а также на рис. 34).

Таблица 2. «Экспериментальные точки» зависимости $y = 1.5e^{0.25x}$, полученные рандомизацией точных значений

Номер п/п	x	y	Номер п/п	x	y
1	-1.0030	1.3	11	2.0010	2.7
2	-0.7010	1.1	12	2.3007	2.6
3	-0.4002	1.35	13	2.6030	2.9
4	-0.1030	1.6	14	2.9002	3.0
5	0.2007	1.53	15	3.2010	3.6
6	0.5002	1.6	16	3.5001	3.2
7	0.8010	1.9	17	3.8010	3.9
8	1.1006	1.9	18	4.1100	4.0
9	1.4030	2.3	19	4.4030	4.5
10	1.7020	2.1	20	4.7007	5.0

Будем считать эти данные результатом эксперимента по (53) и (54) определим коэффициенты A и B (37), а затем сравним их с заданными значениями.

Программа должна содержать функцию, рассчитывающую значение левой части уравнения (53) при передаваемом ей B; это требуется для решения нелинейного уравнения (53). Решать лучше всего методом деления отрезка пополам, так как при этом не требуется предварительное математическое исследование, а диапазон изменения B, как правило, легко задать. Обычно априорно известно грубое приближение этого коэффициента, поэтому легко задать и требуемую точность.

В программу вводятся исходные данные из файла. остальная информация – с клавиатуры. Для нашего примера можно взять Bn=0.01 и Bk=0.9, а eps=10^{-8}. Результаты выводятся в другой файл.

```
program NepolApp;
//количество экспериментальных точек:
const n=20;
```

```
//описание вспомогательных сумм S1, S2, коэффициентов
//экспоненциальной зависимости (37) – A и B, пределов
//изменения коэффициента B в (37) – Bn и Bk, точности
//определения B – eps, счётчика цикла i, файловой переменной tf
//и массивов исходных данных x[i] и y[i]:
var S1,S2,A,B,Bn,Bk,eps:real;i:integer; tf:text;
x,y:array[1..n]of real;
//функция расчёта левой части нелинейного уравнения (53)
//при заданном значении B:
function f(B:real):real;
//описание вспомогательных сумм S1, S2, S3, S4
//и счётчика цикла i:
var S1,S2,S3,S4:real;i:integer;
begin S1:=0;S2:=0;S3:=0;S4:=0; //обнуление сумм
for i:=1 to n do begin //расчёт сумм в цикле:
S1:=S1+exp(2*B*x[i]);S2:=S2+y[i]* exp(B*x[i]);
S3:=S3+x[i]*exp(2*B*x[i]);S4:=S4+x[i]*y[i]*exp(B*x[i]);end;
//возвращаемое значение:
f:=S1/S3-S2/S4;end;
//начало основной программы
begin
//открытие файла исходных данных 'file1a.txt':
assign(tf,'file1a.txt');reset(tf);
//считывание исходных данных из файла 'file1a.txt':
for i:=1 to n do readln(tf,x[i],y[i]);
//закрытие файла 'file1a.txt':
close(tf);
//ввод исходных данных
write('Bn=');readln(Bn);write('Bk=');readln(Bk);
write('eps=');readln(eps);
//расчёт коэффициента B уравнения (37) по (53),
//то есть – решения нелинейного уравнения (53) относительно B
//методом деления отрезка пополам (§ 5, глава 1); заданы пределы
//изменения коэффициента B в (37) – Bn и Bk и точность
//определения B – eps;
//итерационный цикл (например – repeat):
```

```
repeat
if f(Bn)*f((Bn+Bk)/2)<0 then Bk:=(Bn+Bk)/2 else Bn:=(Bn+Bk)/2;
until abs(Bk-Bn)<eps;
B:=(Bn+Bk)/2;
//расчёт коэффициента A уравнения (37) по (54):
S1:=0;S2:=0; //обнуление сумм
for i:=1 to n do begin //расчёт сумм в цикле:
S1:=S1+exp(2*B*x[i]);S2:=S2+y[i]* exp(B*x[i]);end;
A:=S2/S1;
// открытие файла 'file1b.txt' для вывода данных
assign(tf,'file1b.txt');rewrite(tf);
//вывод на экран и в файл значений A и B:
writeln(tf,'A=',A,'  B=',B);writeln('A=',A,'  B=',B);
//закрытие файла 'file1b.txt':
close(tf);
end.
```

В результате работы программы получено:

$A = 1.49947283347555$ (истинное значение 1.5);

$B = 0.248745609708130$ (истинное значение 0.25).

(Напомним, что при вычислениях по формуле (37) эти значения округлять нельзя, округляется только конечный результат).

§ 6. Линеаризация

При **линейной аппроксимации** нормальная система состоит всего из двух уравнений и нет необходимости применять специальные методы решения систем уравнений. Из (44) можно получить простые расчётные формулы для определения коэффициентов a_0 и a_1

$$a_1 = \frac{n\sum_i (y_i x_i) - \sum_i x_i \sum_i y_i}{n\sum_i x_i^2 - \left(\sum_i x_i\right)^2}; \qquad a_0 = \frac{\sum_i y_i - a_1 \sum_i x_i}{n}. \qquad (55)$$

В докомпьютерную эпоху эта методика широко применялась в тех случаях, когда зависимость была близка к линейной. Однако даже при явной нелинейной зависимости формулы (55) можно использовать для аппроксимации, если использовать линеаризацию.

Линеаризация – приведение нелинейной функциональной зависимости известного вида к линейной путём преобразования координат (т. е. аргумента и функции).

Рассмотрим пример зависимости (37). Прологарифмируем (кстати, при линеаризации логарифмирование применяется очень часто) её:

$$\ln y = \ln A + \ln\left(e^{Bx}\right),$$

$$\ln y = \ln A + Bx \ln e,$$

$$\ln y = \ln A + Bx. \qquad (56)$$

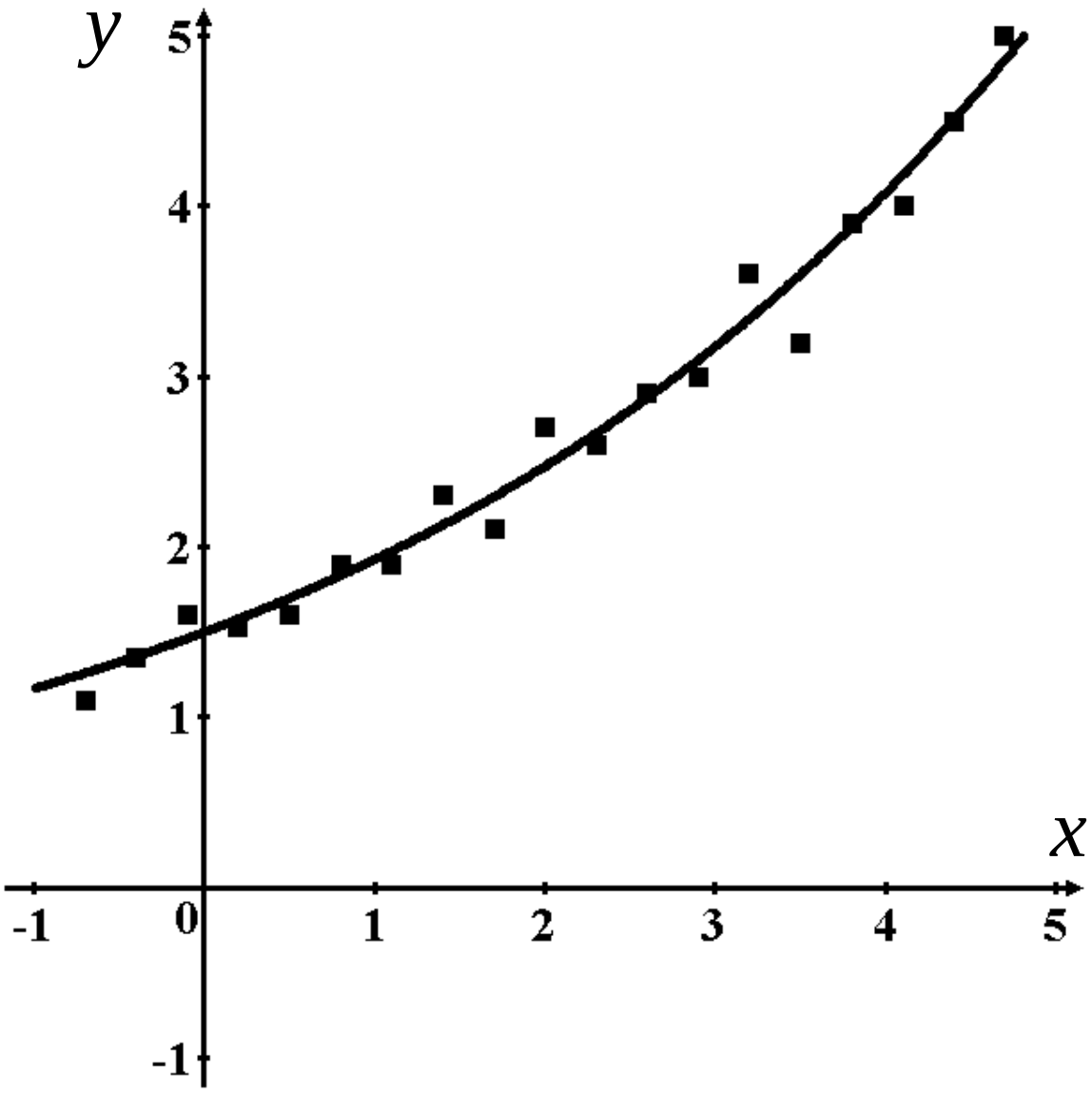

Рис. 34. График функции $y = 1.5e^{0.25x}$ («истинная» кривая).
«Экспериментальные точки» получены рандомизацией точных значений

Выражение (56) представляет собой уравнение прямой: $\ln y$ – новая функция, B – коэффициент при аргументе x, $\ln A$ – свободный член. Это выражение при аппроксимации может быть обработано с помощью (55). (В общем случае при линеаризации могут быть преобразованы и функции y и аргументы x).

Как и в предыдущем параграфе, зададим значения констант, например $A=1.5$ и $B=0.25$. График этой функции в диапазоне $[-1, 5]$ представлен на рис. 34. Для нескольких точек (например, $n=20$), как и выше, смоделируем экспериментальный разброс. Полученные точки представлены на рис. 34 и в табл. 2.

Для проведения линеаризации пересчитаем значения функции y на $\ln y$,; результат показан на рис. 35; здесь хорошо видно, что зависимость стала линейной.

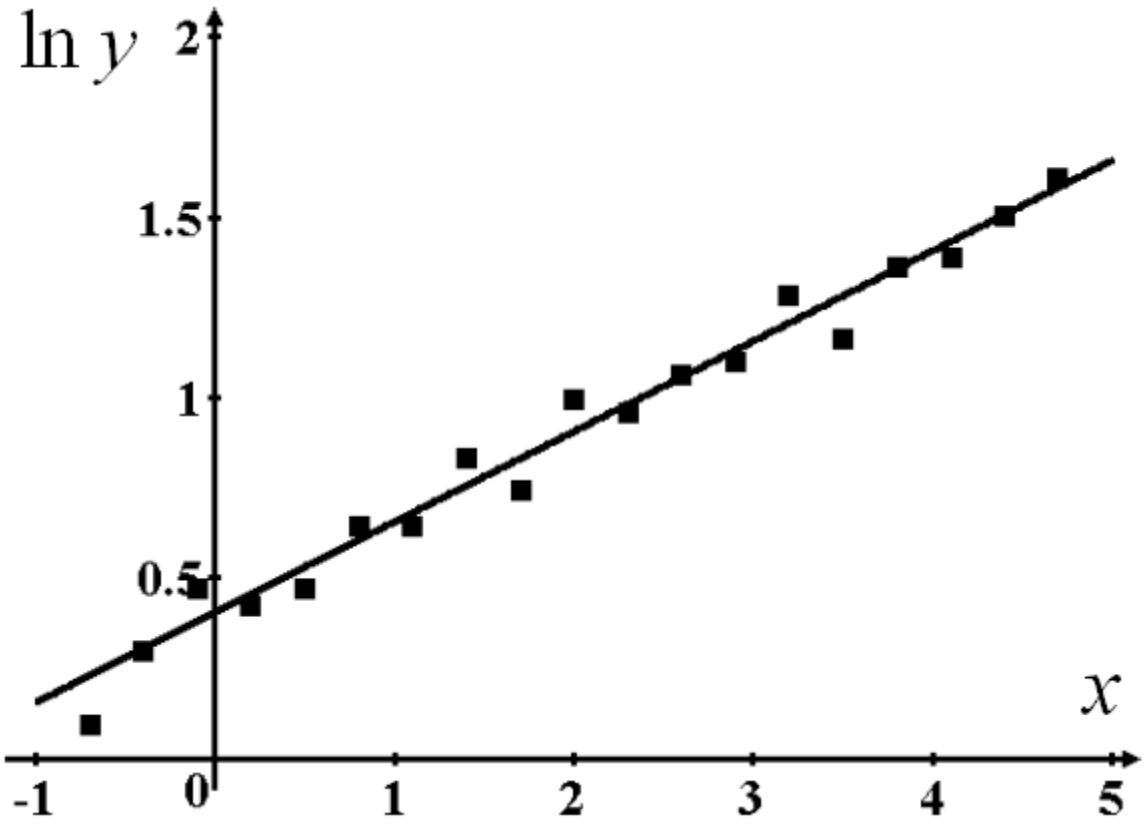

Рис. 35. Линеаризованный график функции $y=1.5e^{0.25x}$
(вида $\ln y=\ln A+Bx$) и линеаризованные «экспериментальные точки»

Составим **программу** линеаризации. Данные (см. табл. 2) будут считываться из файла (в данном примере – 'file1a.txt'). Затем нужно линеаризировать функцию y, т. е. $y_i := \ln y_i$. Для расчёта $a0$ и $a1$ по формулам (55) введём обозначения

$$S1 = \sum_i x_i\ ,\ S2 = \sum_i y_i\ ,\ S3 = \sum_i x_i^{\,2}\ ,\ S4 = \sum_i \left(y_i x_i\right)$$

и рассчитаем эти суммы. После определения $a0$ и $a1$ нужно сделать обратные преобразования («делинеаризацию») и получить значения A и B.

```
program linearisation; //для Паскаля – не более 8 символов
//количество экспериментальных точек:
const n=20;
//описание сумм S1, S2, S3 и S4, коэффициентов линейного
//уравнения (55) – a0, a1 и экспоненциальной зависимости
//(37) –, а также диапазона аргументов xn, xk
//и вспомогательной переменной x1
var S1,S2,S3,S4,a0,a1,A,B,x1,xn,xk:real;i:integer;
x,y:array[1..n]of real;tf:text;
//начало основной программы
begin
//открытие файла исходных данных
assign(tf,'file1a.txt');reset(tf);
//обнуление сумм
S1:=0;S2:=0;S3:=0;S4:=0;
for i:=1 to n do begin
//считывание исходных данных из файла 'file1a.txt'
readln(tf,x[i],y[i]);
//линеаризация функции:
y[i]:=ln(y[i]);
//расчёт сумм в цикле:
S1:=S1+x[i];S2:=S2+y[i];S3:=S3+sqr(x[i]);S4:=S4+x[i]*y[i];end;
//расчёт коэффициентов линейного уравнения по (55):
a1:=(n*S4-S1*S2)/(n*S3-sqr(S1));a0:=(S2-a1*S1)/n;
```

```
//закрытие файла 'file1a.txt' и открытие файла
//'file1b.txt' для вывода данных
close(tf);assign(tf,'file1b.txt');rewrite(tf);
//вывод коэффициентов a0 и a1 на экран и в файл
writeln(tf,'a0=',a0,'  a1=',a1);writeln('a0=',a0,'  a1=',a1);
//«делинеаризация», то есть пересчёт коэффициентов
//A и B из a0 и a1 и их вывод на экран и в файл:
A:=exp(a0);B:=a1;writeln(tf,'A=',A,'  B=',B);writeln('A=',A,'  B=',B);
//проверка полученной формулы в диапазоне от xn=-1 до xk=5;
//для n=20 точек в цикле; x1 – текущее значение аргумента;
//рассчитанные значения выводятся в файл 'file1b.txt'
write('xn=');readln(xn);write('xk=');readln(xk);
x1:=xn;for i:=1 to n do begin
writeln(tf,'x=',x1,' y=',A*exp(B*x1));x1:=x1+(xk-xn)/n;end;
//закрытие файла и конец программы
close(tf);end.
```

Результаты аппроксимации с использованием линеаризации представлены на рис. 36. Полученная аппроксимирующая кривая очень близко проходит около исходной кривой и лишь незначительно отклоняется в правом верхнем углу.

В результате работы программы получены значения:

$$\begin{aligned} a0 &= 4.07145356115343\cdot 10^{-1};\\ a1 &= 2.47309858479032\cdot 10^{-1};\\ A &= 1.50252249062222;\\ B &= 2.47309858479032\cdot 10^{-1}. \end{aligned} \qquad (57)$$

Таким образом, аппроксимирующее уравнение

$$y = 1.50252249062222\, e^{0.247309858479032x}. \qquad (58)$$

Как уже отмечалось (см. § 3) округлять значения коэффициентов A и B нельзя, так как может сильно возрасти погрешность расчёта значений y. Округляется только конечный результат. Среднеквадратическое отклонение подсчитывается по (48).

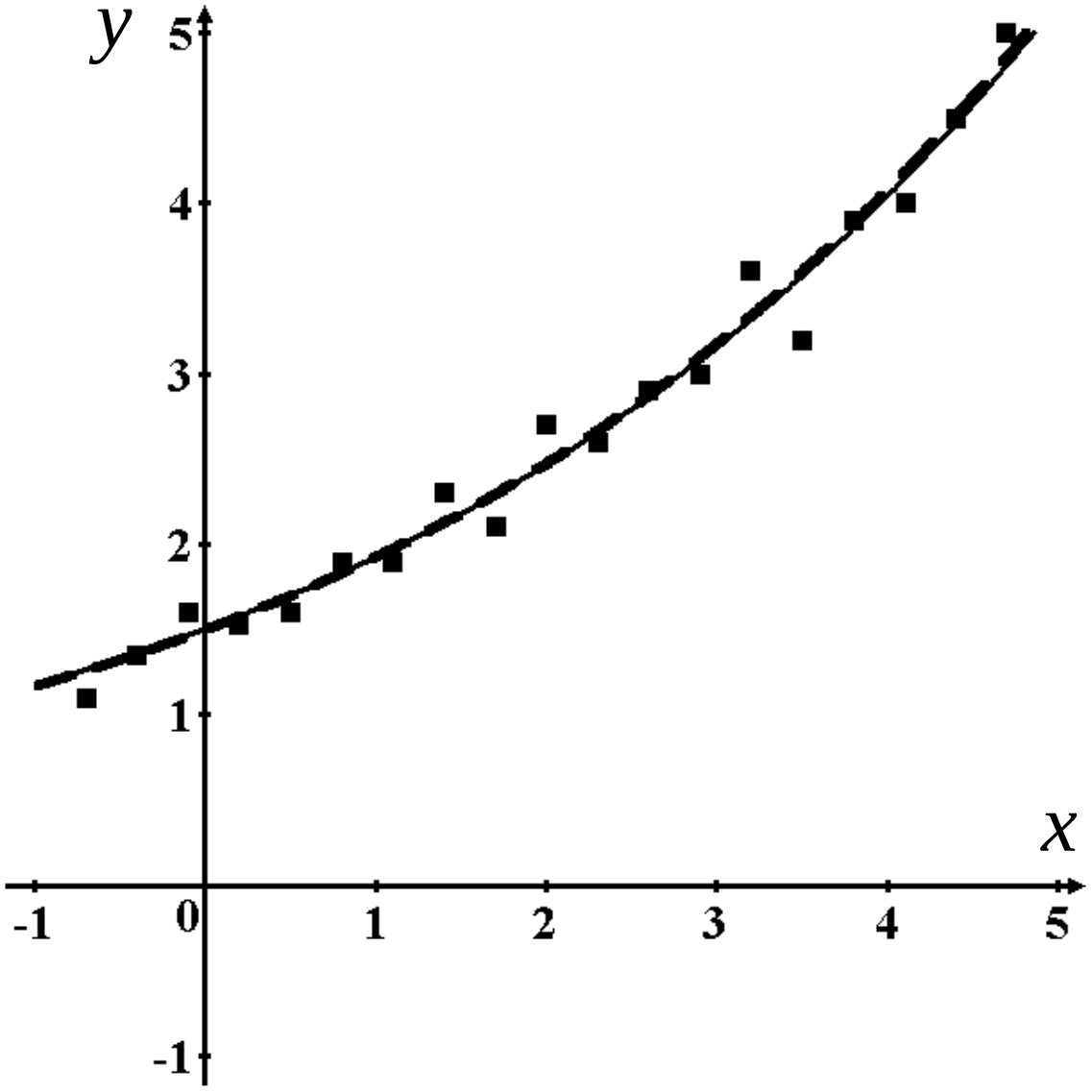

Рис. 36. График функции $y = 1.5e^{0.25x}$ («истинная» кривая – жирная штриховая линия). «Экспериментальные точки» получены рандомизацией точных значений (жирные точки). Тонкая сплошная линия – аппроксимирующая кривая

Как уже отмечалось, линеаризация очень широко использовалась в докомпьютерную эпоху, так как вычисления по (55) можно провести вручную. Однако возможность такой обработки требует априорного знания общего вида аппроксимирующей функции и лёгкости проведения самой линеаризации. Также очень часто проводилась графическая

линеаризация, когда строился график линеаризованной функции (см., например, рис. 34 и 35) и коэффициенты линейного уравнения определялись графически по тангенсу угла наклона прямой и по точке пересечения с осью ординат. Из-за своей наглядности это применяется даже и в настоящее время.

§ 7. Недостатки метода наименьших квадратов

Как уже отмечалось, при полиномиальной аппроксимации левая часть нормальной системы зависит только от экспериментальных значений аргумента x_i. Суммы, которые при этом рассчитываются, определяются только их набором. Как правило эти аргументы распределены по экспериментальному диапазону приблизительно равномерно. Если при этом их значения будут сильно различаться (например – на 2 – 3 порядка), то точки, расположенные в начале интервала, будут вносить в соответствующую сумму очень малую долю, и всё будет определяться большими аргументами x_i. Чтобы устранить этот недостаток, применяется **вариант метода наименьших квадратов с весами**. Обычно его применяют при разбросе значений аргументов более 2 – 3 порядков. Метод сводится к умножению аргументов на некоторые множители («веса»), что уменьшает их разброс по значениям.

Этот недостаток присущ и другим видам аппроксимации.

Формально любую функцию можно разложить в ряд Тэйлора, поэтому полином (38) представляется универсальным средством аппроксимации. Однако, на практике существует множество функций, которые плохо описываются полиномом вида (38), например:

$$y = A + \frac{B}{x}; \tag{59}$$

$$y = A\sqrt{B\,x}; \tag{60}$$

$$y = A\ln\left(B\,x\right), \tag{61}$$

а также – сложные периодические функции. В последнем случае надо строить метод наименьших квадратов на основе ряда Фурье. Во многих случаях помогают ряды с отрицательными степенями или дробными положительными степенями. Иногда, когда это возможно, лучший результат даёт неполиномиальная аппроксимация.

Практически для всех случаев, и в особенности для полиномиальной аппроксимации с высокими степенями полинома, нужно проверять результат аппроксимации графически, простого анализа по критерию (48) может оказаться недостаточно.

Следует также понимать, что в общем случае коэффициенты, определяемые в результате аппроксимации (в особенности – при полиномиальной), являются подгоночными коэффициентами и физического смысла не имеют. При расчётах их округлять нельзя, округляется только конечный результат. Однако в некоторых случаях, когда заранее известен вид аппроксимирующей функции (неполиномиальная аппроксимация), параметры и коэффициенты могут иметь физический смысл или даже являться целью исследования. В этом случае для оценки их погрешности применяют специальные методики.

Также существуют методы аппроксимации функций с использованием **полиномов Чебышёва** и другие методы.

Глава 3. РЕШЕНИЕ СИСТЕМ ЛИНЕЙНЫХ И НЕЛИНЕЙНЫХ УРАВНЕНИЙ

§ 1. Введение

В подавляющем большинстве случаев решение систем уравнений носит вспомогательный характер и редко представляет собой самостоятельную задачу. Однако трудно переоценить важность методов решения этих систем, поскольку очень большое количество задач из других областей вычислительной математики сводятся к системам линейных и нелинейных уравнений. В предыдущей главе мы видели, что решение задачи полиномиальной аппроксимации приводит к необходимости решения систем линейных уравнений. А ведь задача аппроксимации функций является одной из самых важных при обработке экспериментального материала. Другая важнейшая задача математической физики – решение дифференциальных уравнений, обыкновенных и в частных производных. Эти уравнения описывают большинство реальных физических, химических и физико-химических процессов и состояний системы. Применение соответствующих методов очень часто приводит к системам линейных и нелинейных уравнений, которые необходимо уметь решать. Во многих случаях получаются линейные уравнения специфического вида – с трёхдиагональной матрицей. Также с необходимостью решения систем линейных и нелинейных уравнений можно столкнуться при решении задач интерполяции.

В пособии будет рассмотрено четыре метода решения систем уравнений: 1. Метод исключения Гаусса. 2. Метод Гаусса – Зейделя. 3. Метод прогонки (для решения линейных систем с трёхдиагональной матрицей). 4. Метод Ньютона (для решения систем нелинейных уравнений).

§ 2. Решение систем линейных алгебраических уравнений (СЛАУ) методом исключения Гаусса с выбором главного элемента

Системой линейных алгебраических уравнений (СЛАУ) называется система вида:

$$\begin{cases} a_{0,0}x_0 + a_{0,1}x_1 + ... + a_{0,j}x_j + ... + a_{0,n}x_n = b_0 \\ a_{1,0}x_0 + a_{1,1}x_1 + ... + a_{1,j}x_j + ... + a_{1,n}x_n = b_1 \\ ... \\ a_{i,0}x_0 + a_{i,1}x_1 + ... + a_{i,j}x_j + ... + a_{i,n}x_n = b_i \\ ... \\ a_{n,0}x_0 + a_{n,1}x_1 + ... + a_{n,j}x_j + ... + a_{n,n}x_n = b_n \end{cases}. \qquad (62)$$

Здесь каждая строка представляет собой **уравнение** с **неизвестными** x_j, которые имеют степень, равную единице, и поэтому система является **линейной**. Каждое уравнение в отдельности решения не имеет; решить относительно x_j можно только систему, в которой **число неизвестных** $(n+1)$ и **число уравнений** $(n+1)$ **совпадает**. **Коэффициенты** $a_{i,j}$ представляют собой **квадратную матрицу** размерностью $(n+1)\times(n+1)$ и должны быть известны. Нумерация индексов (традиционно: i – номер строки, j – номер столбца) должна начинаться с 0 (т. е. i и j меняются от 0 до n). **Свободные члены** b_i составляют **вектор** (т. е. одномерный массив) размерностью $(n+1)$, и также должны быть известны. Неизвестные x_j тоже являются **вектором** размерностью $(n+1)$; относительно их и должна быть решена система (62). Столбцы (62) занимают одинаковые x_j.

Следует также напомнить, что по главной диагонали системы располагаются коэффициенты $a_{i,j}$ с одинаковыми индексами.

Метод исключения Гаусса основан на принципе, который очевиден при решении системы из двух уравнений:

$$\begin{cases} a_{0,0}x_0 + a_{0,1}x_1 = b_0, \\ a_{1,0}x_0 + a_{1,1}x_1 = b_1. \end{cases} \qquad (63)$$

Из первого уравнения нужно выразить, например, x_0 и подставить во второе уравнение:

$$x_0 = -\frac{a_{0,1}}{a_{0,0}}x_1 + \frac{b_0}{a_{0,0}}, \qquad (64)$$

$$a_{1,0}\left(-\frac{a_{0,1}}{a_{0,0}}x_1+\frac{b_0}{a_{0,0}}\right)+a_{1,1}x_1=b_1,$$

$$\left(-\frac{a_{0,1}}{a_{0,0}}a_{1,0}+a_{1,1}\right)x_1=b_1-\frac{a_{1,0}}{a_{0,0}}b_0. \qquad (65)$$

При этом в (65) останется только неизвестное x_1, т. е. мы получим одно уравнение с одним неизвестным. Таким образом, мы **исключили** неизвестное x_0. После определения x_1 можно вернуться к вычислению x_0 по формуле (64). Тот же алгоритм можно применить и к системе из трёх и более уравнений. **Общим принципом является то, что мы из первого по счёту уравнения выражаем первое по счёту неизвестное и подставляем его во все остальные уравнения, этим самым исключая его.** Затем это повторяется уже для меньшего количества уравнений.

Недостатком таких **алгебраических** операций является то, что на каждом следующем этапе появляются сложные «многоэтажные» дроби и вывести расчётные формулы для решения систем размерностью выше 3 практически невозможно. На помощь приходит **операция** из области программирования – **присваивание**. Можно заметить. что по виду уравнение (65) будет совпадать с уравнением

$$\left\{a_{1,1}x_1=b_1\right.$$

(формально – это система из одного уравнения с одним неизвестным), т. е. можно положить

$$a_{1,1}:=a_{1,1}-\frac{a_{0,1}}{a_{0,0}}a_{1,0};\quad b_1:=b_1-\frac{a_{1,0}}{a_{0,0}}b_0. \qquad (66)$$

При присваивании **новые** значения $a_{1,1}$ и b_1 рассчитываются на основе **старых**, после чего **старые значения пропадают**. (Нужно отметить, что при всех таких преобразованиях коэффициенты и свободные члены будут оставаться **известными**).

Таким образом, если при каждом **исключении** делать операции присваивания, результат **исключения** будет представлять собой СЛАУ того же вида, но с числом уравнений и неизвестных, на единицу меньшим. В конце концов мы придём к системе вида

$$\left\{ a_{\mathrm{n,n}} x_n = b_n , \right. \tag{67}$$

из которой напрямую может быть определён последний x_n по формуле

$$x_n = \frac{b_n}{a_{\mathrm{n,n}}}. \tag{68}$$

Формально выражение (68) также можно представить, как систему из одного уравнения, если положить $b_n := b_n / a_{\mathrm{n,n}}$ (т. е. рассчитать новое значение b_n на основе старого):

$$\left\{ x_n = b_n . \right. \tag{69}$$

Фактически уравнение (69) является решением для x_n. Можно заметить, что в ходе преобразований, подобных (64) – (66), на предпоследнем этапе получится система

$$\begin{cases} a_{n-1,n-1} x_{n-1} + a_{n-1,n} x_n = b_{n-1} \\ a_{n,n-1} x_{n-1} + a_{n,n} x_n = b_n . \end{cases} \tag{70}$$

На следующем этапе (последнем исключении) из первого уравнения после деления его на $a_{n-1,n-1}$ и присваиваний

$$a_{n-1,n} := \frac{a_{n-1,n}}{a_{n-1,n-1}}; \quad b_{n-1} := \frac{b_{n-1}}{a_{n-1,n-1}}$$

можно получить уравнение

$$x_{n-1} + a_{n-1,n} x_n = b_{n-1}. \tag{71}$$

В совокупности с уравнением (69) уравнение (71) представляет возможность прямого расчёта неизвестных x_n и x_{n-1}. Для общего случая из (69) и (71) можно составить систему:

$$\left\{ \begin{array}{l} x_0 + a_{0,1} x_1 + \ldots + a_{0,j} x_j + \ldots + a_{0,n} x_n = b_0 \\ \qquad x_1 + \ldots + a_{1,j} x_j + \ldots + a_{1,n} x_n = b_1 \\ \qquad\qquad \ldots \\ \qquad\qquad\quad x_j + \ldots + a_{i,n} x_n = b_i \\ \qquad\qquad\qquad\quad \ldots \\ \qquad\qquad\qquad\quad x_n = b_n \end{array} \right. . \tag{72}$$

Это система с **треугольной матрицей** (верхней треугольной), в ней заполнены только элементы, расположенные выше главной диагонали, а на самой главной диагонали находятся единицы. Система эта, с одной стороны, даёт возможность быстрого расчёта искомых x_j, с другой – сама она может быть получена в ходе гауссовых исключений. (Следует обратить внимание на то, что коэффициенты $a_{i,j}$ и свободные члены b_i системы (72) отличаются от таковых в исходной системе (62), так как последние изменяются при присваивании).

Первый этап метода исключений Гаусса (**прямой ход**) – приведение исходной системы (62) к треугольному виду (72). Так как при этом происходят тождественные преобразования, обе системы будут эквивалентны и решение системы (72) будет решением системы (62).

Второй этап метода исключений Гаусса (**обратный ход**) – решение системы с треугольной матрицей (72), начиная с последнего неизвестного x_n и заканчивая первым – x_0.

Первый этап

Для вывода расчётных формул рассмотрим преобразование системы с тремя уравнениями, а затем распространим результат на общий случай:

$$\begin{cases} a_{0,0}x_0 + a_{0,1}x_1 + a_{0,2}x_2 = b_0 \\ a_{1,0}x_0 + a_{1,1}x_1 + a_{1,2}x_2 = b_1 \\ a_{2,0}x_0 + a_{2,1}x_1 + a_{2,2}x_2 = b_2. \end{cases} \tag{73}$$

1. Первое исключение. Первое уравнение делим на $a_{0,0}$ и делаем переприсваивание:

$$\boxed{a_{0,1} := \frac{a_{0,1}}{a_{0,0}};\ a_{0,2} := \frac{a_{0,2}}{a_{0,0}};\ b_0 := \frac{b_0}{a_{0,0}}}. \tag{74}$$

При этом получается первое уравнение системы (72)

$$x_0 + a_{0,1}x_1 + a_{0,2}x_2 = b_0. \tag{75}$$

Из (75) выражаем x_0 и подставляем в остальные уравнения:

$$x_0 = -a_{0,1}x_1 - a_{0,2}x_2 + b_0,$$

$$\begin{cases} a_{1,0}(-a_{0,1}x_1 - a_{0,2}x_2 + b_0) + a_{1,1}x_1 + a_{1,2}x_2 = b_1, \\ a_{2,0}(-a_{0,1}x_1 - a_{0,2}x_2 + b_0) + a_{2,1}x_1 + a_{2,2}x_2 = b_2. \end{cases}$$

Раскрываем скобки

$$\begin{cases} -a_{0,1}a_{1,0}x_1 - a_{0,2}a_{1,0}x_2 + b_0a_{1,0} + a_{1,1}x_1 + a_{1,2}x_2 = b_1, \\ -a_{0,1}a_{2,0}x_1 - a_{0,2}a_{2,0}x_2 + b_0a_{2,0} + a_{2,1}x_1 + a_{2,2}x_2 = b_2 \end{cases}$$

и приводим подобные члены

$$\begin{cases} (a_{1,1} - a_{0,1}a_{1,0})x_1 + (a_{1,2} - a_{0,2}a_{1,0})x_2 = b_1 - b_0a_{1,0}, \\ (a_{2,1} - a_{0,1}a_{2,0})x_1 + (a_{2,2} - a_{0,2}a_{2,0})x_2 = b_2 - b_0a_{2,0}. \end{cases}$$

Для приведения последней системы к виду (73) (но на одно уравнение меньше) делаем присваивания

$$\boxed{\begin{array}{l} a_{1,1} := a_{1,1} - a_{0,1}a_{1,0};\ a_{1,2} := a_{1,2} - a_{0,2}a_{1,0};\ b_1 := b_1 - b_0a_{1,0}, \\ {}_{2,1}:= a_{2,1} - a_{0,1}a_{2,0};\ a_{2,2} := a_{2,2} - a_{0,2}a_{2,0};\ b_2 := b_2 - b_0a_{2,0} \end{array}}. \qquad (76)$$

Получаем систему

$$\begin{cases} a_{1,1}x_1 + a_{1,2}x_2 = b_1, \\ a_{2,1}x_1 + a_{2,2}x_2 = b_2 \end{cases} \qquad (77)$$

как результат первого исключения (коэффициенты и свободные члены здесь имеют другие значения по сравнению с (73)).

2. Второе исключение. Повторяем предыдущий алгоритм. Первое уравнение (77) делим на $a_{1,1}$.

$$\boxed{a_{1,2} := \frac{a_{1,2}}{a_{1,1}};\ b_1 := \frac{b_1}{a_{1,1}}}. \qquad (78)$$

Второе уравнение системы (72)

$$x_1 + a_{1,2}x_2 = b_1. \qquad (79)$$

Из (79) выражаем x_1 и подставляем в оставшееся уравнение:

$x_1 = -a_{1,2}x_2 + b_1,$

$\{a_{2,1}(-a_{1,2}x_2 + b_1) + a_{2,2}x_2 = b_2,$

$\{-a_{1,2}a_{2,1}x_2 + b_1a_{2,1} + a_{2,2}x_2 = b_2,$

$\{(a_{2,2} - a_{1,2}a_{2,1})x_2 = b_2 - b_1a_{2,1},$

$$\boxed{a_{2,2} := a_{2,2} - a_{1,2}a_{2,1};\ b_2 := b_2 - b_1a_{2,1}}. \qquad (80)$$

Получаем систему

$$\left\{ a_{2,2}x_2 = b_2 . \right. \tag{81}$$

Это результат второго исключения (система из одного уравнения).

3. Третье исключение. Опять повторяем тот же алгоритм. Первое (и оно же последнее) уравнение (79) делим на $a_{2,2}$:

$$\boxed{b_2 := \frac{b_2}{a_{2,2}}}. \tag{82}$$

Третье уравнение системы (72):

$x_2 = b_2$.

Дальнейшие действия по алгоритму невозможны.

Таким образом, если в программе последовательно выполнить операции (74), (76), (78), (80), (82), то мы получим систему с треугольной матрицей вида (72) соответствующей размерности. (Никакие другие действия не требуются!). Операции (74) и (76), а также (78) и (80), попарно имеют один и тот же алгоритм; в последнем исключении, проходящем по тому же самому алгоритму, вместо двух операций остаётся одна – (82). Т. е. все **исключения** должны быть объединены в один общий цикл (внешний).

Второй этап метода исключений Гаусса (**обратный ход**) – решение системы с треугольной матрицей (72) относительно искомых x_j. Начинается решение с последней строки (72), а заканчивается – первой; это может быть задано с помощью цикла **for downto**. В ходе решения в каждой i-той строке неизвестным является только x_j; $j = i$, остальные x_j; $j > i$ к этому времени уже определены. Таким образом, рекуррентная формула будет выглядеть так:

$$\boxed{x_i = b_i - \sum_{j=i+1}^{n} \left(a_{i,j} x_j \right)}. \tag{83}$$

Выбор главного элемента

Описанный алгоритм может быть улучшен в плане повышения точности вычислений на первом этапе метода. Проблема состоит в том, что если перед проведением исключения левый верхний коэффициент,

на который производится деление (операции (74) и (78)), окажется малым, то это приведёт к большим погрешностям при вычитаниях (операции (76) и (80)). Если же он окажется нулевым, то вычисления вообще невозможны.

Потеря точности при вычитаниях возможна, если уменьшаемое и вычитаемое по значению близки друг другу. Поясним это на следующем примере. Вычтем два близких числа, имеющих точность, например 6 значащих цифр: 425631 и 425612:

$$425631 - 425612 = 19.$$

В результате получилось число всего с 2 значащими цифрами (!).

Главным элементом в столбце (система (62)) называется наибольший по модулю коэффициент этого столбца. При проведении исключений нас интересует левый столбец. Для уменьшения описанного выше отрицательного эффекта перед каждым исключением нужно найти главный элемент в левой строке и переместить соответствующее уравнение на первое место. При этом главный элемент окажется на главной диагонали (выбор главного элемента). Таким образом можно избежать большой потери точности или деления на ноль. (Алгоритм поиска максимума и перестановок описан в первой части пособия).

Программа. Составим программу решения СЛАУ методом исключения Гаусса с выбором главного элемента для любой размерности системы, с вводом данных из бинарного файла. Такая программа будет сопряжена с программой вычисления коэффициентов и свободных членов нормальной системы при проведении полиномиальной аппроксимации (см. предыдущую главу), т. е. с помощью этой программы можно решить нормальную систему и завершить аппроксимацию. Для прикладных целей эти две программы можно объединить.

```
program gauss;
//описание размерности системы или числа неизвестных n+1:
const n=3;
//описание массивов: a – коэффициенты системы,
//b – свободные члены, a1, b1 – копии; x – неизвестные:
var a,a1:array[0..n,0..n] of real;b,b1,x:array[0..n] of real;
//описание счётчиков цикла i, j, k
//и вспомогательной целой переменной imax:
i,j,k,imax:integer;
```

```
//описание вспомогательной переменной max:
max:real;
//описание файловых переменных для текстового – fpt,
//и бинарного – fpb, файлов:
fpt:text;fpb:file of real;
//начало основной программы
begin
//инициализация работы с файлами ft.txt и fb.bin:
assign(fpt,'ft.txt');rewrite(fpt);assign(fpb,'fb.bin');reset(fpb);
//ввод исходных данных из бинарного файла
//и создание копий массивов a и b – a1 и b1:
//внешний цикл по строкам, индекс i:
for i:=0 to n do begin
//внутренний цикл по столбцам, индекс j:
for j:=0 to n do begin read(fpb,a[i,j]); a1[i,j]:= a[i,j]; end;
//ввод свободного члена в той же строке:
read(fpb,b[i]);b1[i]:= b[i];end;
//прямой ход метода Гаусса; внешний цикл
//по номеру исключения k:
for k:=0 to n do begin
//выбор главного элемента;
//первоначальное присваивание вспомогательным переменным:
max:=abs(a[k,k]);imax:=k;
//поиск в левом столбце максимума и его местоположения (imax):
for i:=k+1 to n do
if abs(a[i,k])>max then begin max:= abs(a[i,k]);imax:=i; end;
//перестановка уравнений (переменная max в своём качестве
//больше не нужна, используем её для перестановок):
for j:=k to n do begin max:=a[k,j];a[k,j]:=a[imax,j];
a[imax,j]:=max;end; max:=b[k];b[k]:=b[imax]; b[imax]:=max;
//конец выбора главного элемента
//получение уравнения треугольной системы (74), (78), (82):
for j:=k+1 to n do a[k,j]:=a[k,j]/a[k,k];
b[k]:=b[k]/a[k,k];
//получение системы, на одно уравнение меньшей, (76), (80):
for i:=k+1 to n do begin
```

```
for j:=k+1 to n do a[i,j]:=a[i,j]-a[k,j]*a[i,k];
b[i]:=b[i]-b[k]*a[i,k];end;
//конец цикла по номеру исключения k:
end;
//обратный ход метода Гаусса (83); убывающий цикл
for i:=n downto 0 do begin
x[i]:=b[i]; for j:=i+1 to n do x[i]:=x[i]-a[i,j]*x[j]; end;
//вывод корней x[i] на экран и в текстовый файл:
for i:=0 to n do begin
writeln('x[',i,']=',x[i]); writeln(fpt,'x[',i,']=',x[i]); end;
//проверка решения; первоначальные a и b изменены,
//но сохранены их копии a1 и b1; для каждого
//уравнения подсчитываем левые части и
//сравниваем их со свободными членами; для накопления
//суммы используем свободную переменную max:
for i:=0 to n do begin
max:=0; for j:=0 to n do max:=max+a1[i,j]*x[j];
writeln('left[',i,']=',max,'  b[',i,']=',b1[i]);
writeln(fpt,'left[',i,']=',max,'  b[',i,']=',b1[i]); end;
//конец проверки решения
//закрытие файлов:
close(fpt); close(fpb);
end.
```

Примечания. Размерность системы задаётся в одном месте программы при описании n. Чтобы изменить размерность, нужно задать этой константе другое значение и перекомпилировать программу (более оптимальный алгоритм – с использованием динамической памяти). Ввод данных можно производить из бинарного или текстового файла; приведён первый вариант. После ввода исходных массивов (коэффициентов и свободных членов), если предполагается проверка решения, нужно создать копии этих массивов, так как в алгоритме исключений они изменяются. Важно отметить, что в прямом ходе метода Гаусса, при проведении последнего исключения, когда k=n, во-первых, не должен работать цикл присваиваний коэффициентам **a** новых значений (при построении уравнения треугольной системы; см. (82)). Во-вторых,

вообще не должен работать цикл по получению системы на одно уравнение меньшей. Можно убедиться, что, действительно, всё это выполняется автоматически. После нахождения корней системы можно выполнить проверку решения.

Следует отметить, что для систем с **плохо обусловленными матрицами** применение метода может вести к большим погрешностям и просто неверным решениям. На практике обусловленность матриц необходимо контролировать.

§ 3. Решение СЛАУ методом простых итераций и Гаусса – Зейделя

Итерационная формула метода простых итераций для решения СЛАУ (не путать с таковой для решения нелинейных уравнений!) получается из исходной системы (62) путём тождественных преобразований и приведении её к эквивалентной системе. Из каждого уравнения (с номером i) системы одновременно выражаются неизвестные x_j, причём $j = i$:

$$\begin{cases} x_0 = -\dfrac{a_{0,1}}{a_{0,0}} x_1 - \dfrac{a_{0,2}}{a_{0,0}} x_2 - \ldots - \dfrac{a_{0,j}}{a_{0,0}} x_j - \ldots - \dfrac{a_{0,n}}{a_{0,0}} x_n + \dfrac{b_0}{a_{0,0}}, \\ x_1 = -\dfrac{a_{1,0}}{a_{1,1}} x_0 - \dfrac{a_{1,2}}{a_{1,1}} x_2 - \ldots - \dfrac{a_{1,j}}{a_{1,1}} x_j - \ldots - \dfrac{a_{1,n}}{a_{1,1}} x_n + \dfrac{b_1}{a_{1,1}}, \\ x_2 = -\dfrac{a_{2,0}}{a_{2,2}} x_0 - \dfrac{a_{2,1}}{a_{2,2}} x_1 - \ldots - \dfrac{a_{2,j}}{a_{2,2}} x_j - \ldots - \dfrac{a_{2,n}}{a_{2,2}} x_n + \dfrac{b_2}{a_{2,2}}, \\ \ldots \\ x_j = -\dfrac{a_{i,0}}{a_{i,j}} x_0 - \dfrac{a_{i,1}}{a_{i,j}} x_1 - \dfrac{a_{i,2}}{a_{i,j}} x_2 - \ldots - \dfrac{a_{i,n}}{a_{i,j}} x_n + \dfrac{b_i}{a_{i,j}}; i = j, \\ \ldots \\ x_n = -\dfrac{a_{n,0}}{a_{n,n}} x_0 - \dfrac{a_{n,1}}{a_{n,n}} x_1 - \dfrac{a_{n,2}}{a_{n,n}} x_2 - \ldots - \dfrac{a_{n,j}}{a_{n,n}} x_j - \ldots - \dfrac{a_{n,n-1}}{a_{n,n}} x_{n-1} + \dfrac{b_n}{a_{n,n}}. \end{cases} \quad (84)$$

В правой части системы (84) совокупность x_j, умноженных на соответствующие коэффициенты (обозначим эти коэффициенты $\alpha_{i,j}$), по своей структуре напоминает левую часть исходной системы (62), если дополнить полученную матрицу **α** недостающими диагональными элементами (они должны быть нулевыми, чтобы математически всё осталось без изменений). Обозначим вновь полученный вектор свободных членов **β**, тогда преобразованная система будет выглядеть так (для наглядности приведём систему при $n = 3$):

$$\begin{cases} x_0 = 0 \cdot x_0 + \alpha_{0,1} x_1 + \alpha_{0,2} x_2 + \alpha_{0,3} x_3 + \beta_0, \\ x_1 = \alpha_{1,0} x_0 + 0 \cdot x_1 + \alpha_{1,2} x_2 + \alpha_{1,3} x_3 + \beta_1, \\ x_2 = \alpha_{2,0} x_0 + \alpha_{2,1} x_1 + 0 \cdot x_2 + \alpha_{2,3} x_3 + \beta_2, \\ x_3 = \alpha_{3,0} x_0 + \alpha_{3,1} x_1 + \alpha_{3,2} x_2 + 0 \cdot x_3 + \beta_3 \end{cases} \quad (85)$$

или в векторном виде

x'=α x+β,

где **x** – вектор неизвестных x_j, которые можно отождествить с **предыдущими** значениями, **x'** – вектор неизвестных x_j, которые можно отождествить с **последующими** значениями. Матрицы **α** и **β** можно получить из исходной системы (62), выполнив действия (в программе) последовательно для каждой строки i:

1) в цикле по столбцам $j, (j = 0, 1, ..., n)$

$$\alpha_{i,j} = -\frac{a_{i,j}}{a_{i,i}}; \quad (86)$$

2) в этой же строке

$$\beta_i = \frac{b_i}{a_{i,i}}; \quad (87)$$

3) после прохождения всех строк диагональные элементы матрицы **α** нужно обнулить (после выполнения (86) они получат значение 1):

$$\alpha_{i,i} := 0, \; i = 0, 1, ..., n. \quad (88)$$

Таким образом, **итерационная формула метода простых итераций** будет выглядеть так:

$$\boxed{x_i^{(k+1)} = \sum_{j=0}^{n}\left(\alpha_{i,j}x_j^{(k)}\right) + \beta_i}, \tag{89}$$

где k – номер итерации (но не степень!).

Необходимо отметить, что после расчёта по (86) – (88), матрицы **α** и **β** не изменяются. **За одну** итерацию рассчитываются **все** неизвестные x_i^{k+1}.

Рассмотрим систему из четырёх уравнений (при n = 3); система (85) примет вид:

$$\begin{cases} x_0^{(k+1)} = 0\cdot x_0^{(k)} + \alpha_{0,1}x_1^{(k)} + \alpha_{0,2}x_2^{(k)} + \alpha_{0,3}x_3^{(k)} + \beta_0, \\ x_1^{(k+1)} = \alpha_{1,0}x_0^{(k)} + 0\cdot x_1^{(k)} + \alpha_{1,2}x_2^{(k)} + \alpha_{1,3}x_3^{(k)} + \beta_1, \\ x_2^{(k+1)} = \alpha_{2,0}x_0^{(k)} + \alpha_{2,1}x_1^{(k)} + 0\cdot x_2^{(k)} + \alpha_{2,3}x_3^{(k)} + \beta_2, \\ x_3^{(k+1)} = \alpha_{3,0}x_0^{(k)} + \alpha_{3,1}x_1^{(k)} + \alpha_{3,2}x_2^{(k)} + 0\cdot x_3^{(k)} + \beta_3. \end{cases} \tag{90}$$

При прохождении $(k+1)$-й итерации первая строка может быть рассчитана только однозначно, без вариантов. Однако можно заметить, что при расчёте второй строки (в той же самой итерации) вместо $x_0^{(k)}$ можно использовать только что полученное значение $x_0^{(k+1)}$. Распространим эту идею на всю систему:

$$\begin{cases} x_0^{(k+1)} = 0\cdot x_0^{(k)} + \alpha_{0,1}x_1^{(k)} + \alpha_{0,2}x_2^{(k)} + \alpha_{0,3}x_3^{(k)} + \beta_0, \\ x_1^{(k+1)} = \alpha_{1,0}x_0^{(k+1))} + 0\cdot x_1^{(k)} + \alpha_{1,2}x_2^{(k)} + \alpha_{1,3}x_3^{(k)} + \beta_1, \\ x_2^{(k+1)} = \alpha_{2,0}x_0^{(k+1)} + \alpha_{2,1}x_1^{(k+1)} + 0\cdot x_2^{(k)} + \alpha_{2,3}x_3^{(k)} + \beta_2, \\ x_3^{(k+1)} = \alpha_{3,0}x_0^{(k+1)} + \alpha_{3,1}x_1^{(k+1)} + \alpha_{3,2}x_2^{(k+1)} + 0\cdot x_3^{(k)} + \beta_3. \end{cases} \tag{91}$$

В этом и состоит **модификация Гаусса – Зейделя** метода простых итераций. Такой подход позволяет значительно сократить число итераций, необходимых для достижения заданной точности, т. е. – ускорить расчёты. Также, во многих случаях, эта модификация позволяет получить **сходящийся итерационный процесс** при расходящемся методе простых итераций.

Можно заметить, что для x_i, расположенных на главной диагонали, всё равно, какой номер итерации брать, так как соответствующие коэффициенты равны 0. **Итерационная формула метода Гаусса – Зейделя** будет выглядеть так:

$$x_i^{(k+1)} = \sum_{j=0}^{j=i-1} \left(\alpha_{i,j} x_j^{(k+1)} \right) + \sum_{j=i}^{n} \left(\alpha_{i,j} x_j^{(k)} \right) + \beta_i \qquad (92)$$

На первый взгляд эта формула выглядит сложнее, чем (89), но программно она реализуется проще. В методе (89) необходимы два массива неизвестных x_i – предыдущий и последующий, при реализации же (92) в каждой строке, после подсчёта правой части и присваивания полученного значения переменной справа, соответствующие неизвестные автоматически становятся **последующими** и без переприсваивания могут использоваться далее. (Правда, как мы увидим в дальнейшем, старое значение x_i придётся ещё какое-то время сохранять, чтобы проверить критерий окончания итераций. Поэтому правая часть (92) сначала записывается во вспомогательную переменную (например – vsp), а после проверки критерия – уже в $x_i^{(k+1)}$. Но vsp – неиндексированная переменная; это не то же самое, что содержать массив).

Чтобы использование метода (92) было успешным, необходимо выполнение **критерия сходимости**: **норма матрицы α должна быть меньше единицы**, т. е. должно выполняться одно из следующих условий:

$$\begin{aligned}
&1)\ \sqrt{\sum_{i=0}^{n}\sum_{j=0}^{n} \alpha_{i,j}{}^2} < 1; \\
&2)\ \max_i \sum_{j=0}^{n} \left|\alpha_{i,j}\right| < 1, \quad i = 0, 1, ..., n; \\
&3)\ \max_j \sum_{i=0}^{n} \left|\alpha_{i,j}\right| < 1, \quad j = 0, 1, ..., n.
\end{aligned} \qquad (93)$$

Практически это означает, что на главной диагонали исходной системы (62) должны располагаться наибольшие по абсолютному значению коэффициенты (без учёта свободного члена!). Это далеко не всегда выполняется; иногда исходную систему можно привести к нуж-

ному виду, совершив над ней элементарные преобразования (перестановка уравнений, умножение или деление уравнений на какое-то число и т. д.). Это неприемлемо для больших систем; во всех затруднительных случаях проще воспользоваться методом Гаусса. В реальности метод Гаусса – Зейделя всё же применяется довольно часто в случаях, когда коэффициенты $\left|\alpha_{i,j}\right|$ заведомо будут меньше 1. Это имеет место, например, при решении некоторых дифференциальных уравнений, когда получается система линейных уравнений. В этих случаях более рационально использовать метод Гаусса – Зейделя, как более простой и быстрый.

В качестве **начального приближения** $x_i^{(0)}$ **метода** можно использовать любые числа; если выбор будет неудачным, для получения результата просто потребуется больше итераций. На практике принимают или единицы, или вектор свободных членов **β**.

Критерием окончания метода Гаусса – Зейделя может служить выражение:

$$\boxed{\max_i \left| x_i^{(k+1)} - x_i^{(k)} \right| < \varepsilon}, \qquad (94)$$

где ε – заданная точность.

Программа. Система должна быть проверена на сходимость. Ввод данных осуществляется из текстового файла; он должен представлять собой колонку чисел в последовательности: первая строка коэффициентов ($n+1$ чисел), свободный член этой строки, вторая строка коэффициентов и так далее. В конце проводится проверка решения. В результирующий файл помещается решение и результаты проверки.

```
program zeidel;
//описание размерности системы или числа неизвестных n+1:
const n=3;
//описание массивов: a – коэффициенты системы,
//b – свободные члены, al, bt – коэффициенты
//преобразованной системы; x – неизвестные:
var a,al:array[0..n,0..n] of real;b,bt,x:array[0..n] of real;
//описание счётчиков цикла i, j:
i,j:integer;
//описание заданной точности eps, вспомогательных
//переменных vsp (для проведения итераций) и max
//(для поиска максимума по критерию (94)):
```

```
vsp,max,eps:real;
//описание файловой переменной для текстового файла – fpt,
fpt:text;
//начало основной программы
begin
//ввод заданной точности eps:
write('eps=');readln(eps);
//инициализация работы с файлом исходных данных ish.txt:
assign(fpt,'ish.txt'); reset(fpt);
//ввод исходных данных из текстового файла 'ish.txt',
//расчёт преобразованной системы по (86) и (87)
//внешний цикл по строкам, индекс i:
for i:=0 to n do begin
//внутренний цикл по столбцам, индекс j:
for j:=0 to n do read(fpt,a[i,j]);
//расчёт матрицы al:
for j:=0 to n do al[i,j]:=-a[i,j]/a[i,i];
//ввод свободного члена в той же строке:
read(fpt,b[i]);
//расчёт вектора bt
bt[i]:=b[i]/a[i,i];
//задание начального приближения x_i^(0):
x[i]:=bt[i];
//обнуление диагональных элементов по (88):
al[i,i]:=0;
end;
//закрытие файла исходных данных:
close(fpt);
//итерационный цикл:
repeat
//обнуление переменной max в начале каждой итерации
//для поиска максимальной разности |x_i^(k+1) - x_i^(k)| по всем
//строкам i; все эти разности по модулю положительны,
//а 0 – минимальное положительное число:
max:=0;
```

```
//итерация (91) по каждой i-й строке:
for i:=0 to n do begin
//накопление суммы в переменной vsp по (91):
vsp:=bt[i];for j:=0 to n do vsp:=vsp+al[i,j]*x[j];
//поиск максимума для критерия (94):
if abs(x[i]-vsp)>max then max:= abs(x[i]-vsp);
//переприсваивание «освободившейся» переменной vsp:
x[i]:=vsp;end;
//проверка окончания итерационного цикла по критерию (94):
until max<eps;
//инициализация работы с файлом выходных данных tf1.txt:
assign(fpt,'tf1.txt'); rewrite(fpt);
//вывод корней x[i] на экран и в текстовый файл tf1.txt:
for i:=0 to n do begin
writeln('x[',i,']=',x[i]); writeln(fpt,'x[',i,']=',x[i]); end;
//проверка решения по первоначальной системе
//(с коэффициентами a и свободными членами b)
//и полученным корням x; для каждого уравнения
//подсчитываем левые части и сравниваем их со
//свободными членами b; для накопления суммы
//используем свободную переменную max:
for i:=0 to n do begin
max:=0; for j:=0 to n do max:=max+a[i,j]*x[j];
writeln('left[',i,']=',max,'  b[',i,']=',b[i]);
writeln(fpt,'left[',i,']=',max,'  b[',i,']=',b[i]);end;
//конец проверки решения
//закрытие файла tf1.txt и конец программы:
close(fpt);
end.
```

Пример 1. Решить систему при $\varepsilon = 10^{-4}$:

$$\begin{cases} 10x_0 + 2x_1 - x_2 + 2x_3 = -4, \\ x_0 + 5x_1 + x_2 + 0 \cdot x_3 = 1, \\ x_0 - 2x_1 - 5x_2 + x_3 = 2, \\ 3x_0 + 0 \cdot x_1 + 0 \cdot x_2 + 9x_3 = 10. \end{cases}$$

Решение (распечатка файла):
x[0]=-3.33333196810584E-0001
x[1]= 4.44444307539067E-0001
x[2]=-8.88888766237865E-0001
x[3]=-1.22222217671464E+0000
Проверка решения:
left[0]=-3.99999894021912E+0000
b[0]=-4.00000000000000E+0000
left[1]= 9.99999574646887E-0001
b[1]= 1.00000000000000E+0000
left[2]= 1.99999984258597E+0000
b[2]= 2.00000000000000E+0000
left[3]= 1.00000000000000E+0001
b[3]= 1.00000000000000E+0001

Пример 2. Решить систему при $\varepsilon = 10^{-6}$:

$$\begin{cases} 20x_0 + 21x_1 + 28.7x_2 + 44.1x_3 = 2.95 \\ 21x_0 + 28.7x_1 + 44.1x_2 + 72.2666x_3 = 27.254 \\ 28.7x_0 + 44.1x_1 + 72.2666x_2 + 123.333x_3 = 34.5436 \\ 44.1x_0 + 72.2666x_1 + 123.333x_2 + 216.45581x_3 = 50.61122 \end{cases}$$

Решение (распечатка файла):
x[0]= 2.05992607926701E+0000
x[1]=-5.20136175451107E-0001
x[2]=-6.63897851331623E-0002
x[3]= 2.56172574485678E-0002
Проверка решения:
left[0]= 2.94999961210270E+0001
b[0]= 2.95000000000000E+0001
left[1]= 2.72540220019206E+0001
b[1]= 2.72540000000000E+0001
left[2]= 3.45435623041693E+0001
b[2]= 3.45436000000000E+0001
left[3]= 5.06112200000000E+0001
b[3]= 5.06112200000000E+0001

§ 4. Решение СЛАУ с трёхдиагональной матрицей методом прогонки

Трёхдиагональной матрицей называется квадратная матрица, в которой заполнены главная диагональ и две соседние. (Все остальные элементы – нулевые). В системе с такой матрицей коэффициентов в каждой строке, т. е. в левых частях уравнений за исключением первой и последней имеются три члена. Поэтому соответствующие коэффициенты представляют собой векторы и обозначаются α_i, β_i, γ_i; вектор свободных членов обозначается φ_i, где i – номер строки. Тогда СЛАУ с трёхдиагональной матрицей будет выглядеть так (для наглядности приведём систему из пяти уравнений, $n = 4$):

$$\begin{cases} \beta_0 x_0 + \gamma_0 x = \varphi_0, \\ \alpha_1 x_0 + \beta_1 x_1 + \gamma_1 x_2 = \varphi_1, \\ \alpha_2 x_1 + \beta_2 x_2 + \gamma_2 x_3 = \varphi_2, \\ \alpha_3 x_2 + \beta_3 x_3 + \gamma_3 x_4 = \varphi_3, \\ \alpha_4 x_3 + \beta_4 x_4 = \varphi_4. \end{cases} \qquad (95)$$

В первой строке отсутствует член с α_0, а в последней (для общего случая) – с γ_n.

Такие системы очень часто получаются (как промежуточный результат) при решении численными методами дифференциальных уравнений, обыкновенных и в частных производных, в методах интерполяции, и в других случаях. Решать такие системы (особенно большой размерности) такими мощными методами, как например метод исключения Гаусса, крайне нерационально, так как подавляющее большинство членов левой части нулевые. Поэтому был найден значительно более простой метод – **метод прогонки**, который, как и метод Гаусса, один из немногих в прикладной математике **точных** методов.

Для программной реализации метода нужно привести первое и последнее уравнение системы (95) к общему виду, для чего в первое уравнение нужно добавить член $\alpha_0 x_{0-1}$, а в последнее (для общего случая) – $\gamma_n x_{n+1}$, положив $\alpha_0 = 0$ и $\gamma_n = 0$. (Разумеется, неизвестных x_{0-1} и x_{n+1} «в природе не существует», но это и не важно, поскольку коэффициенты – нулевые). Тогда система будет выглядеть так:

$$\begin{cases} \alpha_0 x_{0-1} + \beta_0 x_0 + \gamma_0 x_1 = \varphi_0, \\ \alpha_1 x_0 + \beta_1 x_1 + \gamma_1 x_2 = \varphi_1, \\ \alpha_2 x_1 + \beta_2 x_2 + \gamma_2 x_3 = \varphi_2, \\ \alpha_3 x_2 + \beta_3 x_3 + \gamma_3 x_4 = \varphi_3, \\ \alpha_4 x_3 + \beta_4 x_4 + \gamma_4 x_{4+1} = \varphi_4. \end{cases} \quad (96)$$

Метод прогонки был впервые предложен советскими учёными. Решение системы вида (95) ищется в виде рекуррентной формулы

$$\boxed{x_i = u_i + v_i x_{i+1}}. \quad (97)$$

(Это означает, что сначала решение **предполагается** из каких-то соображений, а потом **доказывается**). Рекуррентная формула – «обращающаяся к самой себе», т. е. в неё входит какая-то предыдущая и последующая величина (в нашем случае – x_i и x_{i+1}). В формуле (97) параметры u_i и v_i рассчитываются по рекуррентным формулам через коэффициенты системы (96):

$$\boxed{v_i = -\frac{\gamma_i}{\beta_i + \alpha_i v_{i-1}}, \quad i = 0, 1, ..., n}, \quad (98)$$

$$\boxed{u_i = \frac{\varphi_i - \alpha_i u_{i-1}}{\beta_i + \alpha_i v_{i-1}}, \quad i = 0, 1, ..., n}. \quad (99)$$

Поскольку, как мы уже видели,

$$\boxed{\alpha_0 = 0} \quad \text{и} \quad \boxed{\gamma_n = 0} \quad (100)$$

из (98) – (100) следует

$$\boxed{v_0 = -\frac{\gamma_0}{\beta_0}}, \quad (101)$$

$$\boxed{u_0 = \frac{\varphi_0}{\beta_0}}. \quad (102)$$

Также из (101) и (100) следует

$$\boxed{v_n = 0}, \quad (103)$$

а из (97) и (103)

$$\boxed{x_n = u_n}. \qquad (104)$$

Доказательство формул (98) и (99).

В общем виде каждое i-е уравнение системы (96) можно представить как

$$\alpha_i x_{i-1} + \beta_i x_i + \gamma_i x_{i+1} = \varphi_i. \qquad (105)$$

В (97) сдвигаем индексы налево ($i := i - 1$):

$$x_{i-1} = u_{i-1} + v_{i-1} x_i. \qquad (106)$$

В (105) подставляем (106) и получаем

$$\alpha_i \left(u_{i-1} + v_{i-1} x_i\right) + \beta_i x_i + \gamma_i x_{i+1} = \varphi_i.$$

Раскрываем скобки и группируем

$$\left(\alpha_i v_{i-1} + \beta_i\right) x_i + \gamma_i x_{i+1} + \left(\alpha_i u_{i-1} - \varphi_i\right) = 0.$$

В полученное уравнение подставляем (97)

$$\left(\alpha_i v_{i-1} + \beta_i\right)\left(u_i + v_i x_{i+1}\right) + \gamma_i x_{i+1} + \left(\alpha_i u_{i-1} - \varphi_i\right) = 0.$$

Раскрываем скобки и группируем

$$\left(\alpha_i v_{i-1} + \beta_i\right) u_i + \left(\alpha_i v_{i-1} + \beta_i\right) v_i x_{i+1} + \gamma_i x_{i+1} + \left(\alpha_i u_{i-1} - \varphi_i\right) = 0,$$
$$\left(\left(\alpha_i v_{i-1} + \beta_i\right) v_i + \gamma_i\right) x_{i+1} + \left(\left(\alpha_i v_{i-1} + \beta_i\right) u_i + \left(\alpha_i u_{i-1} - \varphi_i\right)\right) = 0. \qquad (107)$$

Чтобы в выражении (107) левая часть равнялась 0, необходимо, чтобы коэффициент при x_{i+1}

$$\left(\left(\alpha_i v_{i-1} + \beta_i\right) v_i + \gamma_i\right) = 0$$

и свободный член в общих скобках равнялся 0:

$$\left(\left(\alpha_i v_{i-1} + \beta_i\right) u_i + \left(\alpha_i u_{i-1} - \varphi_i\right)\right) = 0.$$

Из последних двух формул и получаются рекуррентные формулы (98) и (99).

Конец доказательства.

Алгоритм метода прогонки

1. Сначала рассчитываются вектора параметров u_i и v_i (**прямой ход прогонки**). Для начала расчётов по формулам (98) и (99) определяются v_0 и u_0 по (101) и (102). Затем в цикле с возрастающим индексом проводится расчёт u_i и v_i.

2. **Обратный ход прогонки**. Исходя из начального соотношения (104) в цикле с уменьшающимся индексом по (97) определяются корни уравнения вида (95) x_i.

Программа. Метод прогонки почти исключительно носит вспомогательный характер и используется во многих численных методах как один из этапов решения, поэтому целесообразно оформить его в виде процедуры. Такую процедуру, однажды написанную, можно было бы использовать в различных программах. Сама процедура (**implementation – реализация**) представляет собой **описание процедуры** и без изменений может целиком помещаться в раздел описаний тех или иных программ. Но для **согласованной работы** процедуры и различных программ в последние должна быть помещена информация. обеспечивающая эту согласованность (**interface – средства взаимодействия** или «общения» процедуры и программы). В нашем случае в программе должна быть описана и передана в процедуру размерность системы n, описан новый тип данных – одномерный массив, описаны и переданы в процедуру соответствующие массивы, входные **α, β, γ,** и **φ** и выходной вектор x. Перед использованием процедуры в программе входные **α, β, γ,** и **φ** должны быть рассчитаны. Следует также заметить, что в дальнейшем implementation может быть модифицирована (например, с целью улучшения алгоритма) и можно будет просто заменить ею старую версию в программах (если модификация не затронет interface). Обозначим вектора **α, β, γ,** и **φ** латинскими эквивалентами, например, соответственно, – al, bt, gm и fi.

Interface

Помимо всего прочего в основной программе должна быть описана размерность системы n (для работы с переменной размерностью массивов – через **const** или с использованием динамической памяти и вводом с клавиатуры), новый тип данных (например, mass) одномерный массив, и соответствующие массивы. Допустим, имена последних в процедуре и программе будут совпадать (что совершенно необязательно). Тогда в разделе описаний каждой программы должно быть:

```
…
//описание размерности системы:
const n=4; //
…
//описание нового типа данных одномерный массив:
type mass=array[0..n] of real;
…
var
…
//описание массивов системы (95):
al,bt,gm,fi,x:mass;
…
```

Implementation

```
//x – выходной параметр; описываем как параметр-переменную:
procedure prog(n:integer;al,bt,gm,fi:mass; var x:mass);
//v и u в основной программе не используются,
//описываем их в процедуре:
var v,u:mass;
//описываем счётчик цикла:
i:integer;
//начало основного алгоритма:
begin
//для перестраховки подтверждаем (100):
al[0]:=0;gm[n]:=0;
//рассчитываем начальные v[0] и u[0] по (101) и (102):
v[0]:=-gm[0]/bt[0];u[0]:=fi[0]/bt[0];
//прямой ход прогонки (98) и (99):
for i:=1 to n do begin
v[i]:=-gm[i]/(bt[i]+al[i]*v[i-1]);
u[i]:=(fi[i]-al[i]*u[i-1])/(bt[i]+al[i]*v[i-1])
end;
//определяем последний корень x[n] по (104):
x[n]:=u[n];
//обратный ход прогонки (97) и конец процедуры:
for i:=n-1 downto 0 do x[i]:=u[i]+v[i]*x[i+1];
end;
```

§ 5. Решение систем нелинейных уравнений. Метод Ньютона

Системы нелинейных уравнений очень часто получаются, как промежуточный результат, при решении дифференциальных уравнений численными методами. Кроме того, возможность их решения имеет самостоятельное значение, так как зачастую состояния термодинамических систем или процессы могут быть описаны такими нелинейными системами уравнений.

Решение систем нелинейных уравнений представляет собой очень сложную задачу. Существуют итерационные методы решения, применение которых часто сопряжено с некоторыми трудностями.

Если функции, входящие в систему, дифференцируемы, можно использовать **метод Ньютона**, который в этом случае достаточно универсален. Применение этого метода состоит в предварительных математических преобразованиях (дифференцирование), которые производятся без компьютерных вычислений, после чего задача сводится к решению полученной в результате этого СЛАУ (например, методом исключения Гаусса).

В общем виде нелинейную систему можно представить следующим образом:

$$\begin{cases} f_0\left(x_0, x_1, x_2, ..., x_n\right) = 0, \\ f_1\left(x_0, x_1, x_2, ..., x_n\right) = 0, \\ f_2\left(x_0, x_1, x_2, ..., x_n\right) = 0, \\ ... \\ f_n\left(x_0, x_1, x_2, ..., x_n\right) = 0. \end{cases} \qquad (108)$$

Если хотя бы одно уравнение системы (или даже только одно неизвестное в этом уравнении) является нелинейным, то и вся система будет нелинейной. В уравнения системы не обязательно должны входить все переменные x_i, одна или даже многие могут отсутствовать.

Итерационный процесс строится, исходя из предыдущих приближений корней $x_0^{(k)}, x_1^{(k)}, x_2^{(k)}, ..., x_n^{(k)}$ (k – номер итерации) на основе разложения всех функций f_i в ряд Тэйлора для функции нескольких переменных и отбрасывании членов со второй и более высокими производными:

$$\begin{aligned} & f_i\left(x_0, x_1, x_2, ..., x_n\right) = f_i\left(x_0^{(k)}, x_1^{(k)}, x_2^{(k)}, ..., x_n^{(k)}\right) + \\ & + \left(x_1 - x_1^{(k)}\right) \frac{\partial f_1\left(x_0^{(k)}, x_1^{(k)}, x_2^{(k)}, ..., x_n^{(k)}\right)}{\partial x_1} + \\ & + \left(x_2 - x_2^{(k)}\right) \frac{\partial f_2\left(x_0^{(k)}, x_1^{(k)}, x_2^{(k)}, ..., x_n^{(k)}\right)}{\partial x_2} + ... \approx \\ & \approx f_i\left(x_0^{(k)}, x_1^{(k)}, x_2^{(k)}, ..., x_n^{(k)}\right) + \\ & + \sum_{j=0}^{n} \left(\left(x_j - x_j^{(k)}\right) \frac{\partial f_i\left(x_0^{(k)}, x_1^{(k)}, x_2^{(k)}, ..., x_n^{(k)}\right)}{\partial x_j} \right) = 0. \end{aligned} \qquad (109)$$

Обозначим совокупность $x_0, x_1, x_2, ..., x_n$ как вектор X, тогда $x_0^{(k)}, x_1^{(k)}, x_2^{(k)}, ..., x_n^{(k)}$ будет $X^{(k)}$. Выражение (109) представляет собой систему уравнений; решение этой системы $\boldsymbol{X}$ примем за следующее приближение – $\boldsymbol{X^{(k+1)}}$. Тогда будем иметь

$$\boxed{\sum_{j=0}^{n} \left(\left(x_j^{(k+1)} - x_j^{(k)}\right) \frac{\partial f_i\left(X^{(k)}\right)}{\partial x_j} \right) = -f_i\left(X^{(k)}\right), \quad i = 0, 1, ..., n}. \qquad (110)$$

Выражение (110) представляет собой систему из $n+1$ уравнений, **линейных относительно приращений** $\left(x_j^{(k+1)} - x_j^{(k)}\right)$; частные производные

$$\frac{\partial f_i\left(X^{(k)}\right)}{\partial x_j}$$

являются коэффициентами этой системы (квадратная матрица), а величины

$$-f_i\left(X^{(k)}\right)$$

– свободными членами. Квадратная матрица коэффициентов (частные производные) представляет собой **якобиан**; он может быть рассчитан предварительно. Для наглядности приведём якобиан системы из трёх уравнений:

$$\begin{pmatrix} \frac{\partial f_0\left(X^{(k)}\right)}{\partial x_0} & \frac{\partial f_0\left(X^{(k)}\right)}{\partial x_1} & \frac{\partial f_0\left(X^{(k)}\right)}{\partial x_2} \\ \frac{\partial f_1\left(X^{(k)}\right)}{\partial x_0} & \frac{\partial f_1\left(X^{(k)}\right)}{\partial x_1} & \frac{\partial f_1\left(X^{(k)}\right)}{\partial x_2} \\ \frac{\partial f_2\left(X^{(k)}\right)}{\partial x_0} & \frac{\partial f_2\left(X^{(k)}\right)}{\partial x_1} & \frac{\partial f_2\left(X^{(k)}\right)}{\partial x_2} \end{pmatrix}. \qquad (111)$$

Общетеоретический подход к решению системы (110) состоит в нахождении матрицы, **обратной** якобиану; обращение матрицы – довольно сложная задача, поэтому на практике систему (110) решают как линейную (например, методом исключения Гаусса) относительно приращений $\left(x_j^{(k+1)} - x_j^{(k)}\right)$, а затем из них находят последующие приближения $x_j^{(k+1)}$.

Пример. Рассмотрим систему

$$\begin{cases} 0.5x_0^2 + 2x_1^2 + x_2 + 1 = 0, \\ \sin(x_1 - x_2) - x_0 x_2 - 3 = 0, \\ \ln x_1 + x_2^2 = 0. \end{cases} \qquad (112)$$

Найдём производные

$$\frac{\partial f_0}{\partial x_0} = x_0; \quad \frac{\partial f_0}{\partial x_1} = 4x_1; \quad \frac{\partial f_0}{\partial x_2} = 1;$$

$$\frac{\partial f_1}{\partial x_0} = -x_2; \quad \frac{\partial f_1}{\partial x_1} = \cos\left(x_1 - x_2\right); \quad \frac{\partial f_1}{\partial x_2} = -\cos\left(x_1 - x_2\right) - x_0;$$

$$\frac{\partial f_2}{\partial x_0} = 0; \quad \frac{\partial f_2}{\partial x_1} = \frac{1}{x_1}; \quad \frac{\partial f_2}{\partial x_2} = 2x_2.$$

Линейная система будет иметь вид:

$$\begin{cases} x_0^{(k)}\left(x_0^{(k+1)} - x_0^{(k)}\right) + 4x_1^{(k)}\left(x_1^{(k+1)} - x_1^{(k)}\right) + \left(x_2^{(k+1)} - x_2^{(k)}\right) = -f_0\left(X^{(k)}\right), \\ -x_2^{(k)}\left(x_0^{(k+1)} - x_0^{(k)}\right) + \cos\left(x_1^{(k)} - x_2^{(k)}\right)\left(x_1^{(k+1)} - x_1^{(k)}\right) + \\ \qquad + \left(-\cos\left(x_1^{(k)} - x_2^{(k)}\right) - x_0^{(k)}\right)\left(x_2^{(k+1)} - x_2^{(k)}\right) = -f_1\left(X^{(k)}\right), \\ 0 + \dfrac{1}{x_1^{(k)}}\left(x_1^{(k+1)} - x_1^{(k)}\right) + 2x_2^{(k)}\left(x_2^{(k+1)} - x_2^{(k)}\right) = -f_2\left(X^{(k)}\right). \end{cases}$$

Коэффициенты и свободные члены этой системы известны из предыдущего приближения (свободные члены рассчитываются по (112)); систему можно решить относительно приращений $\left(x_j^{(k+1)} - x_j^{(k)}\right)$, а из них рассчитать $x_j^{(k+1)}$.

Глава 4. ИНТЕРПОЛИРОВАНИЕ ФУНКЦИЙ

§ 1. Введение

Интерполяцией (**интерполированием**) **функции**, заданной дискретно на конечном числе точек (т. е. на **сети точек**), называется построение **интерполяционной функции** (интерполирующей функции), **точно** проходящей через заданные точки и **приближённой** к истинной кривой в промежутках между этими точками (рис. 37).

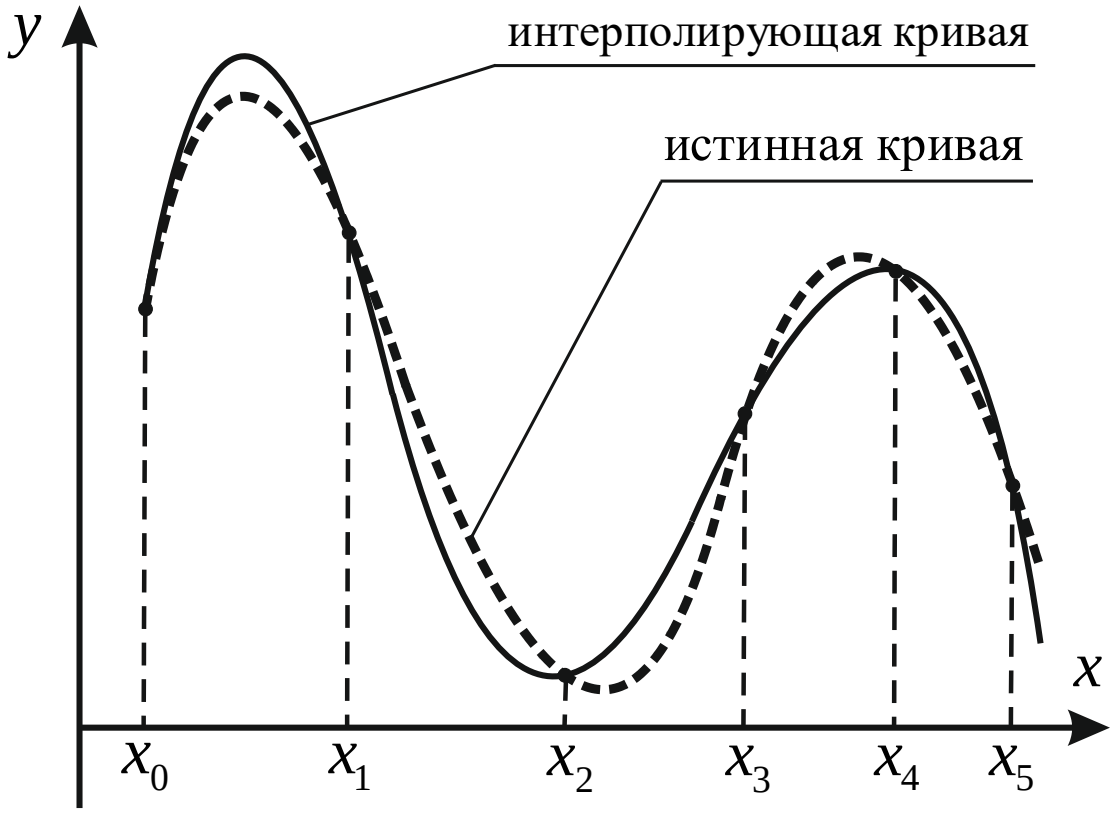

Рис. 37. Интерполирование функции (штриховая линия), заданной на сети точек x_0, x_1, x_2, x_3, x_4, x_5. Сеть точек в данном случае неравномерная. Интерполирующая кривая (сплошная линия) точно проходит через все узлы, но имеет отклонения от истинной кривой в промежутках

Таким образом, мы имеем дело с **таблично заданной** функцией. В отличие от задачи аппроксимации все дискретные точки считаются заданными **точно**. Интерполяционная функция строится **внутри** данного интервала точек.

Практическое использование полученной интерполяционной функции, которая лучше или хуже соответствует истинной зависимости, состоит в возможности **вычисления промежуточных значений функции** на данной сети точек с некоторой погрешностью.

Экстраполяцией называется расширение действия интерполяционной функции за пределы диапазона сети точек (рис. 38). Строго говоря, эта задача некорректно поставленная, так как за пределами диапазона истинная функция может резко изменить свой вид (рис. 39), а это невозможно предугадать, исходя из имеющихся дискретных точек таблицы. На практике экстраполяцию применяют очень осторожно и только вблизи границ интервала.

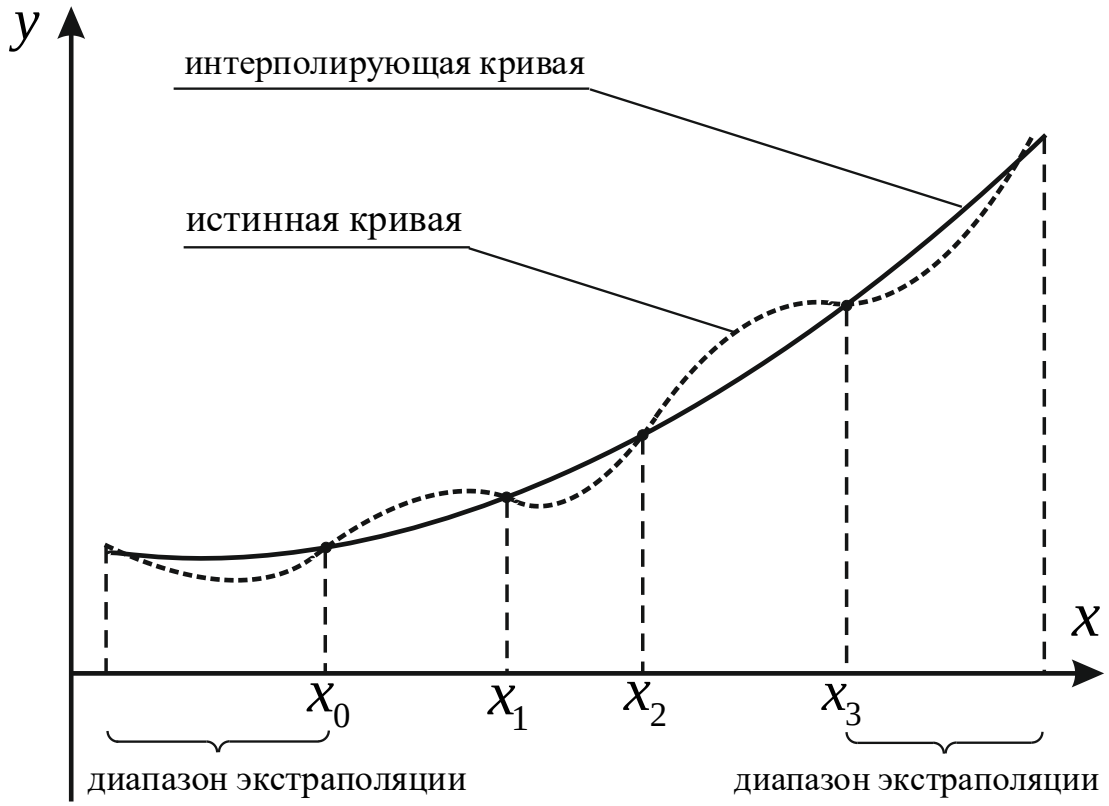

Рис. 38. Экстраполяция функции (штриховая линия), заданной на сети точек x_0, x_1, x_2, x_3. Интерполирующая кривая (сплошная линия) может быть продолжена за диапазон заданных точек; если характер кривых не меняется, можно рассчитывать приближённое значение функции в диапазонах экстраполяции

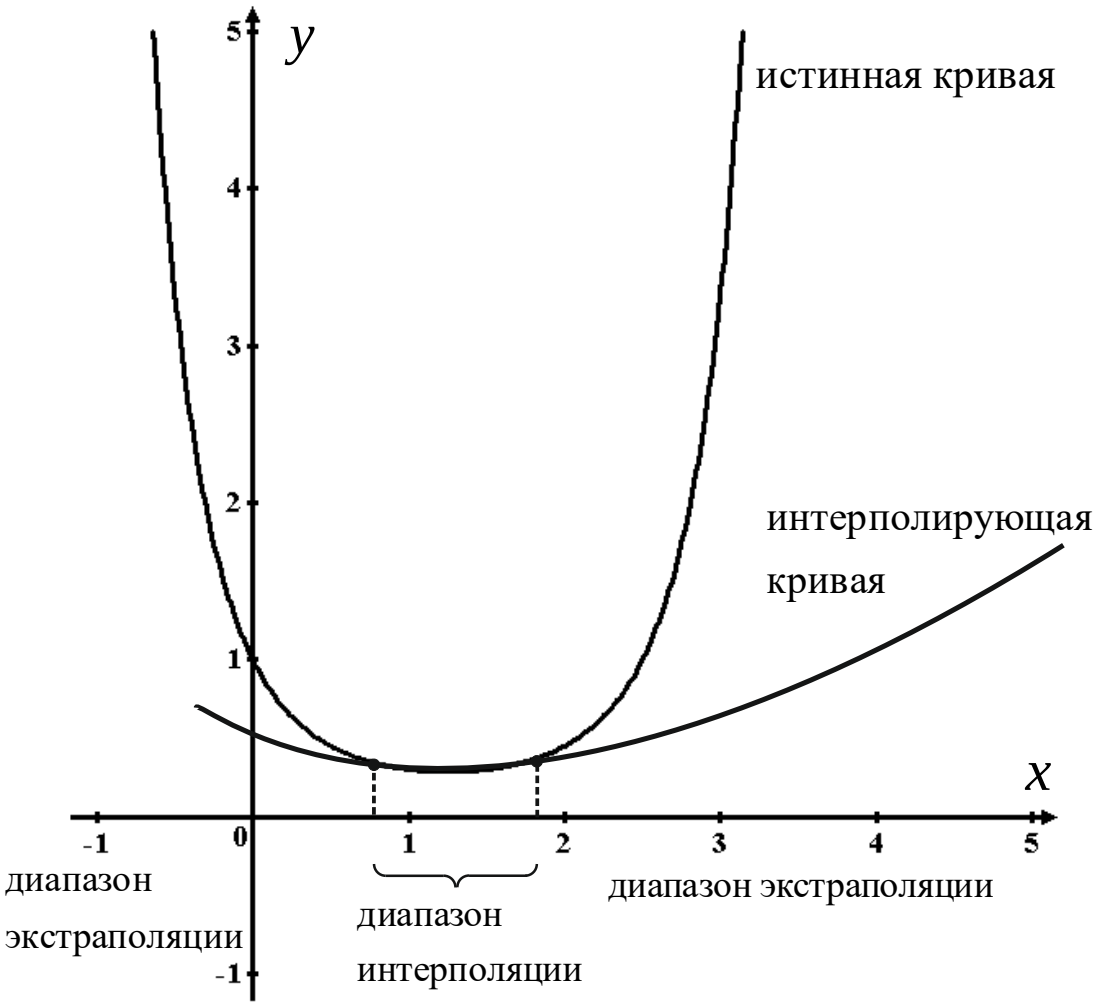

Рис. 39. Случай, когда экстраполяцию применять нельзя. За пределами диапазона интерполяции, которая проведена корректно, кривые резко расходятся

Задачу интерполяции можно проиллюстрировать на следующем примере. В химической и физической лабораторной практике очень часто бывает нужным определить плотность воды при комнатной температуре. При этом пользуются справочными таблицами, в которых значения плотности даны для дискретного набора температур с каким-то шагом (табл. 3). Чтобы определить плотность воды при промежуточной не заданной в таблице температуре (например, 24.3 °C), делают упрощённую линейную интерполяцию. Из таблицы выписывают две пары значений температур и плотностей таким образом, чтобы нужная

температура находилась внутри диапазона табличных (24 – 0.99732; 25 – 0.99707).

Таблица 3. Плотность воды в интервале 15 – 30 °С

t, ºС	$\rho \cdot 10^{-3}$, кг/м^3	*t*, ºС	$\rho \cdot 10^{-3}$, кг/м^3
15	0.99913	23	0.99756
16	0.99897	24	0.99732
17	0.99880	25	0.99707
18	0.99862	26	0.99681
19	0.99843	27	0.99654
20	0.99823	28	0.99626
21	0.99802	29	0.99597
22	0.99780	30	0.99567

Затем делят диапазон температур (аргументов) и плотностей (функций), например на десять частей (одна часть по температуре – 0.1; одна часть по плотности – (0.99707 – 0.99732) : 10 = –0.000025), т. е. находят, какое изменение плотности соответствует десятой части изменения температур. После этого определяют, на сколько частей заданная температура отличается от одной из табличных (левой или правой; в нашем случае заданная температура ближе к левой – 24 °С, отличается на три части), и полагают изменение плотности на такое же количество частей по функции. Одну часть по функции умножают на количество частей (т. е. $–0.000025 \cdot 3 = –0.000075$) и прибавляют или вычитают (в зависимости от того, возрастающая функция или убывающая) от соответствующего значения табличной функции (0.99732 + (–0.000075) = 0.997245). Полученное значение находится между 0.99732 и 0.99707, как и должно быть.

Конечно, линейная интерполяция весьма неточна (рис. 40), поэтому большой практический интерес представляют более точные, нелинейные методы.

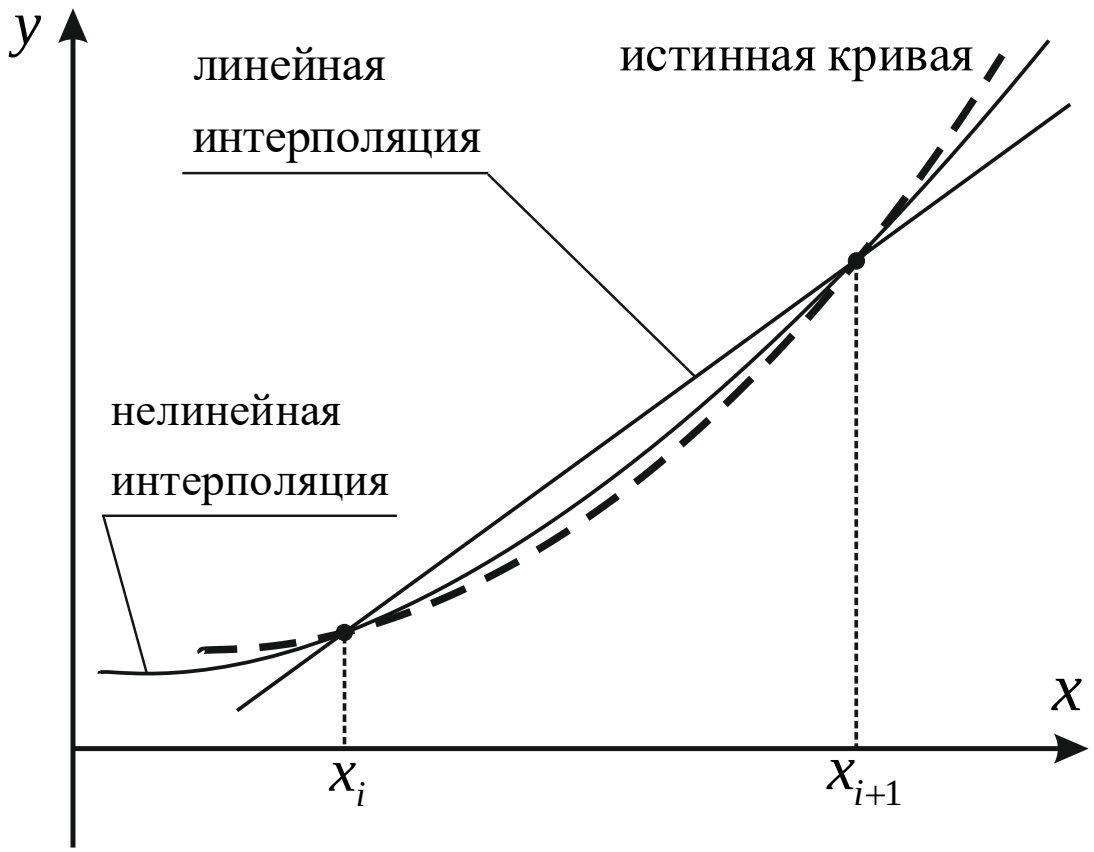

Рис. 40. В общем случае линейная интерполяция даёт большую погрешность, чем нелинейная; x_i и x_{i+1} – соседние точки интерполяционной сетки

§ 2. Конечные разности различных порядков. Таблица конечных разностей

Если функция $y = f(x)$ задана на сети точек, например в реальной таблице, то для проведения интерполяции из таблицы выбирают несколько пар значений аргумента и функции. следующих подряд, и в дальнейшем работают с ними (табл. 4). При этом первый аргумент (x_0) называется **вхождением в таблицу**. Выбранный диапазон должен охватывать все интересующие нас значения, в частности, если требуется найти значение функции при промежуточном x, не входящем в таблицу, этот x должен находиться внутри диапазона.

Таблица 4. Таблично заданная функция $y = f(x)$. Выборка из более общей таблицы $n+1$ пары значений функции и аргумента подряд (x_0 – вхождение в таблицу)

x	x_0	x_1	x_2	x_3	…	x_{n-1}	x_n
y	y_0	y_1	y_2	y_3	…	y_{n-1}	y_n

Если сеть точек **равномерная**, то можно охарактеризовать изменение функции конечными разностями по y, которые показывают изменения между соседними y. Например, если из таблицы взяты пять пар значений, то **конечными разностями первого порядка** будут:

$$\Delta y_0 = y_1 - y_0, \quad \Delta y_1 = y_2 - y_1, \quad \Delta y_2 = y_3 - y_2, \quad \Delta y_3 = y_4 - y_3. \quad (113)$$

Аналогично записываются **конечные разности второго порядка**:

$$\Delta^2 y_0 = \Delta y_1 - \Delta y_0, \quad \Delta^2 y_1 = \Delta y_2 - \Delta y_1, \quad \Delta^2 y_2 = \Delta y_3 - \Delta y_2. \quad (114)$$

Здесь верхний индекс у Δ означает не степень, а **порядок конечной разницы**. Количество конечных разностей каждого следующего порядка на единицу меньше, чем предыдущего. Значения самих табличных y формально можно считать **конечными разницами нулевого порядка**. Таким образом, можно построить **таблицу конечных разностей** (одного порядка в каждом столбце):

$$\begin{aligned}
&\Delta^0 y_0 = y_0; \quad \Delta y_0 = y_1 - y_0; \quad \Delta^2 y_0 = \Delta y_1 - \Delta y_0; \quad \Delta^3 y_0 = \Delta^2 y_1 - \Delta^2 y_0;\\
&\Delta^0 y_1 = y_1; \quad \Delta y_1 = y_2 - y_1; \quad \Delta^2 y_1 = \Delta y_2 - \Delta y_1;\\
&\Delta^0 y_2 = y_2; \quad \Delta y_2 = y_3 - y_2;\\
&\Delta^0 y_3 = y_3.
\end{aligned} \quad (115)$$

Таблица составляется по столбцам, и это легко может быть запрограммировано.

§ 3. Интерполяция по Ньютону

Интерполяционные формулы Ньютона основываются на ряде Тэйлора (30). Перед их использованием должна быть построена таблица конечных разностей (115).

Первая интерполяционная формула Ньютона для $n+1$ пар значений

$$\boxed{\begin{array}{l} y = y_0 + q\,\Delta y_0 + \dfrac{q(q-1)}{2!}\Delta^2 y_0 + \dfrac{q(q-1)(q-2)}{3!}\Delta^3 y_0 + \ldots \\ \ldots + \dfrac{q(q-1)\ldots(q-n+1)}{n!}\Delta^n y_0; \quad q = \dfrac{x - x_0}{x_1 - x_0} \end{array}}, \quad (116)$$

где x – заданный промежуточный аргумент (не содержащийся в таблице), для которого нужно найти значение функции y; x_0 – вхождение в таблицу; параметр q – число шагов, необходимых для достижения точки x, исходя из точки x_0.

Формулу (116) наиболее рационально использовать вблизи точки x_0, где q мало.

Если в правой части (116) ограничиться двумя членами ряда (т. е. $n=1$), будет иметь место **линейная интерполяция** (аналог описанной во введении), если тремя ($n=2$), – **квадратичная**.

В первой интерполяционной формуле используется только первая строка таблицы конечных разностей (115), но таблицу необходимо составлять полностью, так как расчёт её ведётся по столбцам.

Вблизи конца таблицы лучше применять **вторую интерполяционную формулу Ньютона**:

$$\boxed{\begin{array}{l} y = y_n + q\,\Delta y_{n-1} + \dfrac{q(q+1)}{2!}\Delta^2 y_{n-2} + \dfrac{q(q+1)(q+2)}{3!}\Delta^3 y_{n-3} + \ldots \\ \ldots + \dfrac{q(q+1)\ldots(q+n-1)}{n!}\Delta^n y_0; \quad q = \dfrac{x - x_0}{x_1 - x_0} \end{array}}. \quad (117)$$

Таким образом, первую формулу Ньютона лучше использовать для **интерполирования вперёд** и **экстраполирования назад**, а вторую – для **интерполирования назад** и **экстраполирования вперёд**.

Однако на практике чаще всего ограничиваются использованием первой интерполяционной формулы Ньютона, так как обычно n не превышает значения 3 или 4, и разницы в результатах почти нет.

Интерполяция по Ньютону используется для монотонных функций с сеткой с постоянным шагом. Периодические и быстро меняющиеся функции обрабатываются плохо.

Составим программу для первой формулы Ньютона (116).

В программе сначала рассчитывается таблица конечных разностей (115). а затем производится подсчёт по формуле (116). При подсчёте таблицы (115) автоматически обеспечивается её треугольный вид.

```
program IntplNewt;
//описание количества пар точек, взятых для интерполяции. n+1:
const n=3;
//описание массива таблицы конечных разностей,
//как двумерной матрицы:
var dy:array[0..n,0..n] of real;
//описание заданного промежуточного значения x,
//вхождения в таблицу x0, следующего за ним значения x1,
//вспомогательной переменной для накопления произведения p,
//параметра q и искомого значения функции y:
x,x0,x1,p,q,y real;
//описание счётчиков цикла i и j:
i,j:integer;
//начало основной программы
begin
//ввод вхождения в таблицу и следующего за ним x1:
write('x0=');readln(x0); write('x1=');readln(x1);
//ввод заданного промежуточного x:
write('x=');readln(x);
//ввод в цикле табличных значений y как элементов
//первого столбца матрицы dy[i,0]:
for i:=0 to n do begin
write('y[',i,']=');readln(dy[i,0]);end;
//расчёт параметра q:
q:=(x-x0)/(x1-x0);
```

```
//расчёт таблицы конечных разностей по столбцам,
//начиная со второго (с индексом 1) столбца:
for j:=1 to n do
//во внутреннем цикле рассчитываются элементы столбца;
//в каждом следующем столбце будет на одну строку меньше
//(см. 115), это обеспечивается переменным значением
//конечного значения параметра цикла (n-j):
for i:=0 to n-j do dy[i,j]:=dy[i+1,j-1]-dy[i,j-1];
//задание начального значения переменной
//для накопления произведения p:
p:=1;
//накопление суммы y, первоначальное присваивание
//свободного слагаемого:
y:=dy[0,0];
//цикл вычисления по формуле (116):
for i:=1 to n do begin
p:=p*(q-i+1)/i;y:=y+p*dy[0,i];end;
//вывод заданного x и соответствующего ему значения
//функции y:
writeln('x=',x,'  y=',y);
end.
```

Пример 1

Проверим ручной расчёт, рассмотренный во введении (§ 1). Для значения температуры $24.3\ ^{o}C$ результат линейной интерполяции полностью совпадает с ручным расчётом – 99.7245, а кубическая интерполяция даёт значение 99.724605. Возможно, этот пример не очень показателен.

Пример 2

Отличия линейной и кубической интерполяции лучше проявляются для сильно меняющихся функций. Рассмотрим пример табл. 5 (фрагмент реальной таблицы). Для температуры $-15.3\ ^{o}C$ программа даёт значения: линейная интерполяция – 161.61 Па, кубическая – 161.022 Па.

Следует отметить, что, кроме интерполяции по Ньютону, для таблиц с **постоянным шагом** (т. е. – **равномерной сеткой**) существуют также **интерполяционные формулы Гаусса, Стирлинга и Бесселя**, основанные на таблицах с **центральными разностями**.

Таблица 5. Давление насыщенного водяного пара в равновесии со льдом

t, °C	p, Па
–18	125.2
–16	150.9
–14	181.5
–12	217.6

§ 4. Интерполяция по Лагранжу

В отличие от интерполяционных формул Ньютона и других вышеупомянутых, **интерполяция по Лагранжу** применяется для **монотонных функций** с **неравномерными сетками** (или, что то же самое, – для таблиц с **непостоянным шагом**). Формула Лагранжа в законченном виде представляется так:

$$\begin{aligned} y = {} & \frac{(x-x_1)(x-x_2)\ldots(x-x_n)}{(x_0-x_1)(x_0-x_2)\ldots(x_0-x_n)}\,y_0 + \frac{(x-x_0)(x-x_2)\ldots(x-x_n)}{(x_1-x_0)(x_1-x_2)\ldots(x_1-x_n)}\,y_1 + \ldots \\ \ldots + {} & \frac{(x-x_0)(x-x_1)\ldots(x-x_{k-1})(x-x_{k+1})\ldots(x-x_n)}{(x_k-x_0)(x_k-x_1)\ldots(x_k-x_{k-1})(x_k-x_{k+1})\ldots(x_k-x_n)}\,y_k + \ldots \\ \ldots + {} & \frac{(x-x_0)(x-x_1)\ldots(x-x_{n-1})}{(x_n-x_0)(x_n-x_1)\ldots(x_n-x_{n-1})}\,y_n \end{aligned} \qquad (118)$$

Или, в общем виде, как полином Лагранжа

$$L_n(x) = \sum_{k=0}^{n}\left(\frac{\prod_{j\neq k}(x-x_j)}{\prod_{j\neq k}(x_k-x_j)}\,f(x_k)\right). \qquad (119)$$

В случае равномерной сетки интерполяционный полином Лагранжа совпадает с интерполяционным полиномом Ньютона.

Программа. Интерполяционная формула Лагранжа (118) не очень удобна для программирования. Каждое слагаемое должно рассчитываться отдельно, дроби – с накоплением произведения, причём в последнем случае – с исключением множителей с одинаковыми номерами слагаемого и табличного аргумента.

```
program lagrange;
//описание количества пар значений аргументов и функций,
//взятых для интерполирования n+1:
const n=3;
//описание табличных значений аргумента и функции,
//как одномерных массивов:
var x,y:array[0..n] of real;
//описание счётчиков цикла i, j:
i,j:integer;
//описание заданного промежуточного аргумента xx
//и соответствующего ему искомого значения функции yy,
//а также вспомогательной переменной p для расчёта
// коэффициентов Лагранжа по (118):
xx,yy,p:real;
//описание файловой переменной для текстового файла – fpt,
fpt:text;
//начало основной программы
begin
//инициализация работы с файлом исходных данных ish.txt:
assign(fpt,'ish.txt'); reset(fpt);
//ввод исходных данных из текстового файла 'ish.txt',
for i:=0 to n do readln(fpt,x[i],y[i]);
//ввод заданного промежуточного значения x:
write('x=');readln(xx);
//вычисление значения yy, как суммы, обнуление:
yy:=0;
//внешний цикл по слагаемым, индекс i:
for i:=0 to n do begin
//расчёт дроби для i-го слагаемого:
p:=1;
for j:=0 to n do if i<>j then p:=p*(xx-x[j])/(x[i]-x[j]);
//расчёт i-го слагаемого:
p:=p*y[i];
//накопление суммы yy:
yy:=yy+p;
end;
```

```
//вывод полученного результата:
writeln('x=',xx,'  y=',yy);
//закрытие файла tf1.txt и конец программы:
close(fpt);
end.
```

Пример 1 (см. пример 2 предыдущего параграфа). Кубическая интерполяция для равномерной сетки (см. табл. 5) при температуре –15.3 ºС даёт значение 161.016 Па, что, после округления до сотых, совпадает с интерполяцией по Ньютону.

Пример 2. Для неравномерной сетки (табл. 6; это фрагмент реальной таблицы) программа интерполяции по Лагранжу даёт следующие значения:

концентрация: 0.03 364.66
0.08 354.18
0.40 335.95

Это вполне согласуется с таблицей.

Таблица 6. Молярная электропроводность Λ HCl при 18 ºС

Концентрация, моль/л	0.01	0.05	0.1	0.5
Λ, см^2 / Ом/ моль	370	360	351	327

Пример 3. Однако интерполяция по Лагранжу неприменима для быстроменяющихся функций (а тем более – для периодических). Рассмотрим пример из той же самой справочной таблицы, но для другого вещества – табл. 7 (возьмём только первые 4 точки).

Таблица 7. Молярная электропроводность Λ CH_3COOH при 18 ºС

Концентрация, моль/л	0.01	0.05	0.1	0.5	1	3	5
Λ, см^2 / Ом/ моль	14.3	6.48	4.60	2.01	1.32	0.54	0.285

Результат расчётов

концентрация: 0.03 электропроводность: 9.59
0.08 4.44
0.40 36.7

Из всех значений только первое более или менее согласуется с таблицей.

Таким образом, интерполяция по Лагранжу применима для монотонных и неприменима для периодических и быстроменяющихся функций. Количество пар значений, взятых из таблицы для интерполирования, не должно быть большим, – обычно не более 4 – 5 точек. Корректность полученных результатов должна постоянно контролироваться.

§ 5. Интерполяция кубическими сплайнами

Это сравнительно новый метод, он был разработан в середине XX века. В переводе с английского (spline) это слово означает гибкую металлическую линейку (или лекало), с помощью которой, если её поставить на ребро и изогнуть, можно провести плавную кривую, проходящую через несколько точек на плоскости (рис. 41). При этом, если точек достаточно много, эта кривая практически совпадёт с истинной кривой.

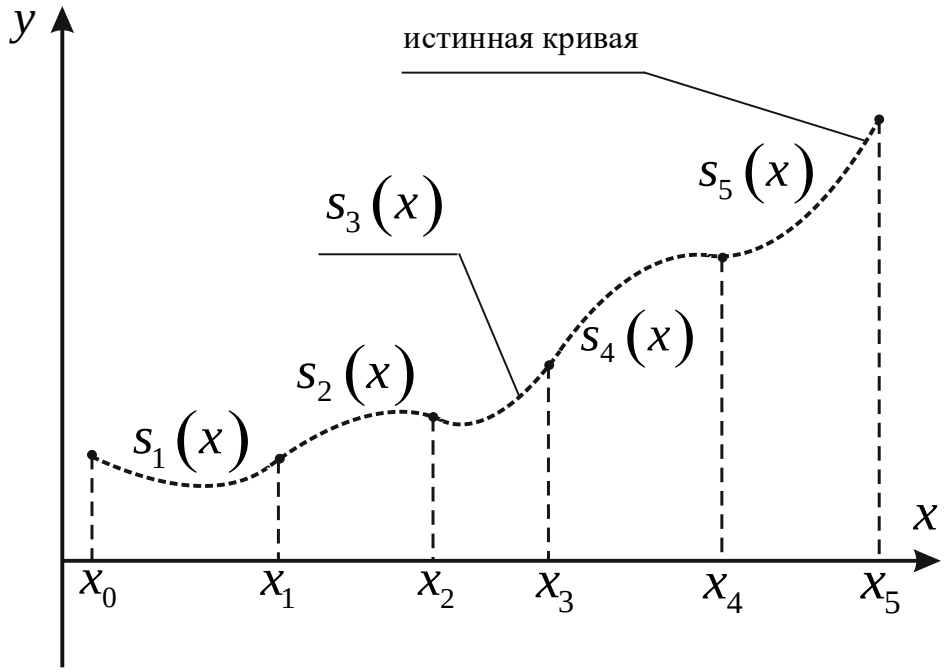

Рис. 41. Интерполяция сплайнами. С их помощью через узлы можно провести плавную кривую (не показана), практически совпадающую с истинной кривой; x_i – неравномерная сеть точек; жирные точки – узлы интерполяции; $s_i(x)$ – сегменты, приближённые сплайнами

Таким образом, весь диапазон интерполирования ($n+1$ точка) разбивается на сегменты (n сегментов), каждый из которых приближается некоторой функцией, чаще всего полиномом, – **кусочно-полиномиальная интерполяция**. В подавляющем большинстве случаев такой функцией и является кубический сплайн. Чтобы сплайны плавно переходили один в другой, необходимо выполнение ряда условий. Будет недостаточно, если в узлах интерполяции будут совпадать значения самих функций сплайнов, как это показано на рис. 42 для точки x_i. Можно заметить, что в этой точке у соседних сплайнов разные производные, которые определяются касательными. Чтобы кривая была плавная, необходимо равенство этих производных. То же относится и к производным более высоких порядков.

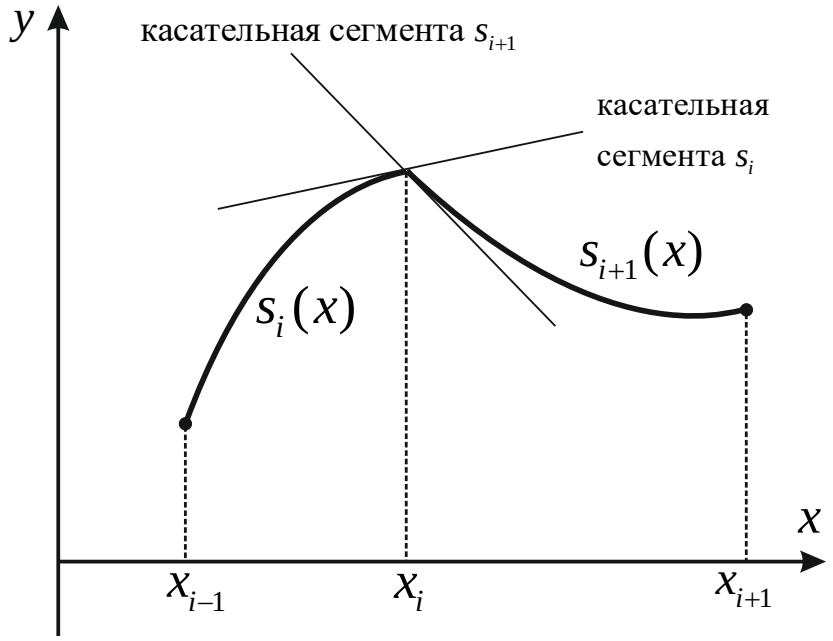

Рис. 42. Соседние сегменты при интерполировании сплайнами. Значения сплайнов для точки x_i совпадают, но не совпадают значения производных

Итак, пусть функция $y = f(x)$ задана на неравномерной сети из $n+1$ точки ($x_0 = a$ и $x_n = b$, см. рис. 41). Необходимо построить сплайн-функцию $y = s(x)$, удовлетворяющую следующим трём условиям:

1. На каждом сегменте

$$[x_{i-1}, x_i], i = 1, 2, ..., n \qquad (120)$$

$s(x)$ многочлен третьей степени.

2. Соблюдается **условие интерполирования**, т. е. в узлах сплайн точно равен табличному y_i:

$$s(x_i) = y_i,\ i = 0, 1, 2, ..., n. \tag{121}$$

3. На отрезке $[a, b]$ должны быть непрерывны:

а) функция $s(x)$: $s_i(x_i) = s_{i-1}(x_i),\ i = 1, 2, ..., n-1;$ (122)

б) первая производная $s'(x)$: $s_i'(x_i) = s_{i-1}'(x_i),\ i = 1, 2, ..., n-1;$ (123)

в) первая производная $s''(x)$: $s_i''(x_i) = s_{i-1}''(x_i),\ i = 1, 2, ..., n-1.$ (124)

На каждом сегменте $[x_{i-1}, x_i],\ i = 1, 2, ..., n$ нужно построить функцию $s(x) = s_i(x)$ для $x_{i-1} \le x \le x_i$ в виде многочлена третьей степени (из формулы Тэйлора (30)):

$$\boxed{s_i(x) = a_i + b_i(x - x_i) + \frac{c_i}{2}(x - x_i)^2 + \frac{d_i}{6}(x - x_i)^3;\ i = 1, 2, ..., n}, \tag{125}$$

где коэффициенты a_i b_i c_i и d_i **нужно определить**; если они станут известными, то по формуле (125) можно рассчитывать промежуточные значения функции $y = f(x)$. Неизвестных будет $3n$.

Определение коэффициентов a_i b_i c_i и d_i.

Продифференцируем (125)

$$s_i'(x) = b_i + c_i(x - x_i) + \frac{d_i}{2}(x - x_i)^2, \tag{126}$$

$$s_i''(x) = c_i + d_i(x - x_i), \tag{127}$$

$$s_i'''(x) = d_i. \tag{128}$$

Для узлов сетки $x = x_i$, тогда из (126) – (128) имеем

$$a_i = s_i(x_i),\ b_i = s_i'(x_i),\ c_i = s_i''(x_i),\ d_i = s_i'''(x_i). \tag{129}$$

Для $x = x_{i-1}$ из (125) и (129) получаем

$$a_{i-1} = a_i + b_i(x_{i-1} - x_i) + \frac{c_i}{2}(x_{i-1} - x_i)^2 + \frac{d_i}{6}(x_{i-1} - x_i)^3. \tag{130}$$

Из (121) вытекает

$$a_i = y_i,\ i = 1, 2, ..., n,\ \text{доопределим}\ a_0 = y_0, \tag{131}$$

т. е. коэффициенты a_i определять не надо, они равны табличным y_i.

Обозначим шаги по аргументу на каждом сегменте

$$h_i \equiv x_i - x_{i-1} \tag{132}$$

и с учётом (131) имеем (133)

$$\begin{cases} h_i b_i - \dfrac{h_i^2}{2} c_i + \dfrac{h_i^3}{6} d_i = y_i - y_{i-1};\ i = 1, 2, ..., n, & (133) \\ h_i c_i - \dfrac{h_i^2}{2} d_i = b_i - b_{i-1};\ i = 2, 3, ..., n, & (134) \\ h_i d_i = c_i - c_{i-1};\ i = 2, 3, ..., n, & (135) \\ c_0 = c_n = 0. & (136) \end{cases}$$

Аналогично из (126), (129) и (123) получаем (134), а из (127), (129) и (124) – (135). Из предположения $f''(a) = f''(b) = 0$ следует $s''(a) = s''(b) = 0$, тогда из (129) вытекает (136) (это предположение не всегда верно).

В итоге мы получили СЛАУ (133) – (136); она может быть решена, например, методом исключения Гаусса, но есть более простой способ. Из системы можно выделить систему, содержащую только c_i, она будет представлять собой систему с трёхдиагональной матрицей, которую можно решить методом прогонки.

Сведение к системе с трёхдиагональной матрицей

Вычтем два соседних (со сдвигом индекса) уравнения (133):

$$b_i = \frac{h_i}{2} c_i - \frac{h_i^2}{6} d_i + \frac{y_i - y_{i-1}}{h_i}$$

–

$$b_{i-1} = \frac{h_{i-1}}{2} c_{i-1} - \frac{h_{i-1}^2}{6} d_{i-1} + \frac{y_{i-1} - y_{i-2}}{h_{i-1}},$$

в результате получится

$$b_i - b_{i-1} = \frac{1}{2}(h_i c_i - h_{i-1} c_{i-1}) - \frac{1}{6}(h_i^2 d_i - h_{i-1}^2 d_{i-1}) + \frac{y_i - y_{i-1}}{h_i} - \frac{y_{i-1} - y_{i-2}}{h_{i-1}}. \tag{137}$$

Подставляем в (134)

$$h_i c_i + h_{i-1} c_{i-1} - \frac{h_{i-1}^2}{3} d_{i-1} - \frac{2h_i^2}{3} d_i = 2\left(\frac{y_i - y_{i-1}}{h_i} - \frac{y_{i-1} - y_{i-2}}{h_{i-1}} \right). \tag{138}$$

Из (135):

$d_i h_i^2 = (c_i - c_{i-1})h_i$ и $h_{i-1}^2 d_{i-1} = h_{i-1}(c_{i-1} - c_{i-2})$,

подставляем в (138):

$$h_{i-1}c_{i-2} + 2(h_{i-1} + h_i)c_{i-1} + h_i c_i = 6\left(\frac{y_i - y_{i-1}}{h_i} - \frac{y_{i-1} - y_{i-2}}{h_{i-1}}\right). \qquad (139)$$

Сдвигаем индексы и получаем окончательно

$$\begin{cases} h_i c_{i-1} + 2(h_i + h_{i+1})c_i + h_{i+1}c_{i+1} = 6\left(\frac{y_{i+1} - y_i}{h_{i+1}} - \frac{y_i - y_{i-1}}{h_i}\right) \\ c_0 = c_n = 0 \qquad\qquad i = 1, 2, ..., n-1. \end{cases} \qquad (140)$$

Эта система с трёхдиагональной матрицей позволяет рассчитать коэффициенты c_i. Тогда остальные коэффициенты можно будет определить по явным формулам (из (135) и (133)):

$$\begin{array}{l} d_i = \frac{c_i - c_{i-1}}{h_i}; \qquad b_i = \frac{h_i}{2}c_i - \frac{h_i^2}{6}d_i + \frac{y_i - y_{i-1}}{h_i} \\ i = 1, 2, ..., n. \end{array} \qquad (141)$$

При использовании полученной сплайн-функции нужно сначала определить, в каком сегменте оказывается заданный промежуточный *x*, а потом вычислять соответствующее значение функции.

Программа

```
program spline;
//описание количества точек (пар значений аргументов
//и функций) в таблице (n+1):
const n=30;
//описание (как одномерных массивов, с образованием
//нового типа данных) – 1) табличных значений аргумента x
//и функции y, 2) шагов по аргументу h, 3) коэффициентов метода
//сплайнов b, c и d, 4) коэффициентов и свободных членов метода
//прогонки al, bt, gm и fi:
type mass=array[0..n] of real;
var x,y,h,b,c,d,al,bt,gm,fi:mass;
//описание заданного промежуточного аргумента xx,
```

```
//соответствующего ему искомого значения функции yy,
//и шага постоянной сетки hh:
xx,yy,hh:real;
//описание счётчика цикла i и местоположения
//xx на сети точек (обозначим ixx),
//ключей выбора в интерфейсе k1 и k2,
//и числа разбиений заданного интервала nn:
i,ixx,k1,k2,nn:integer;
//описание файловой переменной для текстового файла – fpt,
fpt:text;
//процедура метода прогонки (implementation), см. § 4:
procedure prog(n:integer;al,bt,gm,fi:mass; var x:mass);
var v,u:mass;
i:integer;
begin
al[0]:=0;gm[n]:=0;
v[0]:=-gm[0]/bt[0];u[0]:=fi[0]/bt[0];
for i:=1 to n do begin
v[i]:=-gm[i]/(bt[i]+al[i]*v[i-1]);
u[i]:=(fi[i]-al[i]*u[i-1])/(bt[i]+al[i]*v[i-1])
end;
x[n]:=u[n];
for i:=n-1 downto 0 do x[i]:=u[i]+v[i]*x[i+1];
end;

//начало основной программы
begin
//инициализация работы с файлом исходных данных ish.txt:
assign(fpt,'ish.txt'); reset(fpt);
//ввод исходных данных из текстового файла 'ish.txt',
for i:=0 to n do readln(fpt,x[i],y[i]);
//закрытие файла исходных данных:
close(fpt);
//расчёт массива шагов по x (132);
h[0]:=0;
for i:=1 to n do h[i]:=x[i]-x[i-1];
```

```
//подготовка интерполирования методом кубических сплайнов;
//расчёт одномерных массивов b, c и d;
//определение массива c решением системы с трёхдиагональной
//матрицей (140); рассчитываем al,bt,gm,fi и вызываем
//процедуру prog (см. § 4);
//поскольку при этом определяются элементы массива c
//с индексами от 1 до n-1 (140), а в процедуре прогонки требуются
//индексы от 0, сдвигаем индексы у al,bt,gm,fi влево на единицу,
//то есть они будут от 0 до n-2;
//расчёт коэффициентов первого и последнего уравнения
//системы (140):
bt[0]:=2*(h[1]+h[2]);gm[0]:=h[2];
fi[0]:=6*((y[2]-y[1])/h[2]-(y[1]-y[0])/h[1]);
al[n-2]:=h[n-1];bt[n-2]:=2*(h[n-1]+h[n]);
fi[n-2]:=6*((y[n]-y[n-1])/h[n]-(y[n-1]-y[n-2])/h[n-1]);
//расчёt коэффициентов уравнений с индексами 1 – (n-3):
for i:=2 to n-2 do begin
al[i-1]:=h[i];
bt[i-1]:=2*(h[i]+h[i+1]);
gm[i-1]:=h[i+1];
fi[i-1]:=6*((y[i+1]-y[i])/h[i+1]-(y[i]-y[i-1])/h[i]);
end;
//вызов процедуры prog;
prog(n-2,al,bt,gm,fi,c);
//в массиве c значения, индексы которых должны быть от
//1 до n-1, по факту записаны в ячейки памяти с индексами от
//0 до n-2, поэтому сдвигаем их индексы вправо на единицу:
for i:=n-1 downto 1 do c[i]:=c[i-1];
//доопределяем c[0] и c[n] по (140):
c[0]:=0;c[n]:=0;
//рассчитываем коэффициенты b и d по (141), чтобы на
//последнем сегменте получались корректные результаты,
//цикл делаем до n:
for i:=1 to n do begin d[i]:=(c[i]-c[i-1])/h[i];
b[i]:=h[i]/2*c[i]-sqr(h[i])/6*d[i]+(y[i]-y[i-1])/h[i];end;
//конец подготовки интерполирования

//интерполирование возможно в двух режимах: для всего
//диапазона с постоянным шагом и для отдельных значений x
```

```
//Интерполяция для всего диапазона с постоянным шагом
writeln('Интерполяция для всего диапазона’);
writeln(‘с постоянным шагом? Да: 1, нет: 0');
//ввод ключа выбора:
write('k1=');readln(k1);
if k1=1 then begin
write('Число разбиений n=');readln(nn);
//инициализация работы с выходным файлом rs2.txt:
assign(fpt,'rs2.txt'); rewrite(fpt);
//обнуление начального значения аргумента xx и расчёт шага hh:
xx:= x[0];hh:=(x[n]-x[0])/nn;
//интерполяция с постоянным шагом в цикле:
for i:=1 to nn do begin
xx:=xx+hh;
//поиск номера сегмента (номера сплайна) на сети аргументов
//для xx:
ixx:=0;repeat ixx:=ixx+1;until (xx<x[ixx]) or (ixx>n);
//контроль выхода аргумента за диапазон значений:
if ixx>n then ixx:=ixx-1;
//расчёт yy по (125):
yy:=y[ixx]+b[ixx]*(xx-x[ixx])+c[ixx]/2*sqr(xx-x[ixx])+
d[ixx]/6*sqr(xx-x[ixx])*(xx-x[ixx]);
//вывод полученного результата:
writeln(fpt,'x=',xx,' y=',yy);
end;
//закрытие файла rs2.txt:
close(fpt);
end;

//Интерполяция для отдельных значений x
writeln('Интерполяция для отдельных значений x? Да: 1, нет: 0');
//ввод ключа выбора:
write('k1=');readln(k1);
if k1=1 then begin
//инициализация работы с выходным файлом rs1.txt:
assign(fpt,'rs1.txt'); rewrite(fpt);
repeat
```

```
//ввод заданного промежуточного значения xx:
write('x=');readln(xx);
//поиск номера сегмента (номера сплайна) на сети аргументов
//для xx:
ixx:=0;repeat ixx:=ixx+1;until (xx<x[ixx]) or (ixx>n);
//контроль выхода аргумента за диапазон значений:
if ixx>n then ixx:=ixx-1;
//расчёт yy по (125):
yy:=y[ixx]+b[ixx]*(xx-x[ixx])+c[ixx]/2*sqr(xx-x[ixx])+
d[ixx]/6*sqr(xx-x[ixx])*(xx-x[ixx]);
//вывод полученного результата на экран и в файл:
writeln('x=',xx,'  y=',yy);
writeln(fpt,'x=',xx,'  y=',yy);
//повторение или окончание вычислений:
writeln('Закончить вычисления? Да: 1, нет: 0');
//ввод ключа выбора:
write('k2=');readln(k2);
until k2=1;
//закрытие файла rs1.txt:
close(fpt);
end;
end.
```

Пример 1. Интерполяция кубическими сплайнами монотонной зависимости с неравномерным шагом (табл. 8).

Таблица 8. Давление насыщенного водяного пара в равновесии с водой (фрагмент таблицы)

t, ºC	*p*, кПа
24	2.982
25	3.166
26	3.360
28	3.778
32	4.753
34	5.318
38	6.623
40	7.374

Результаты представлены на рис. 43.

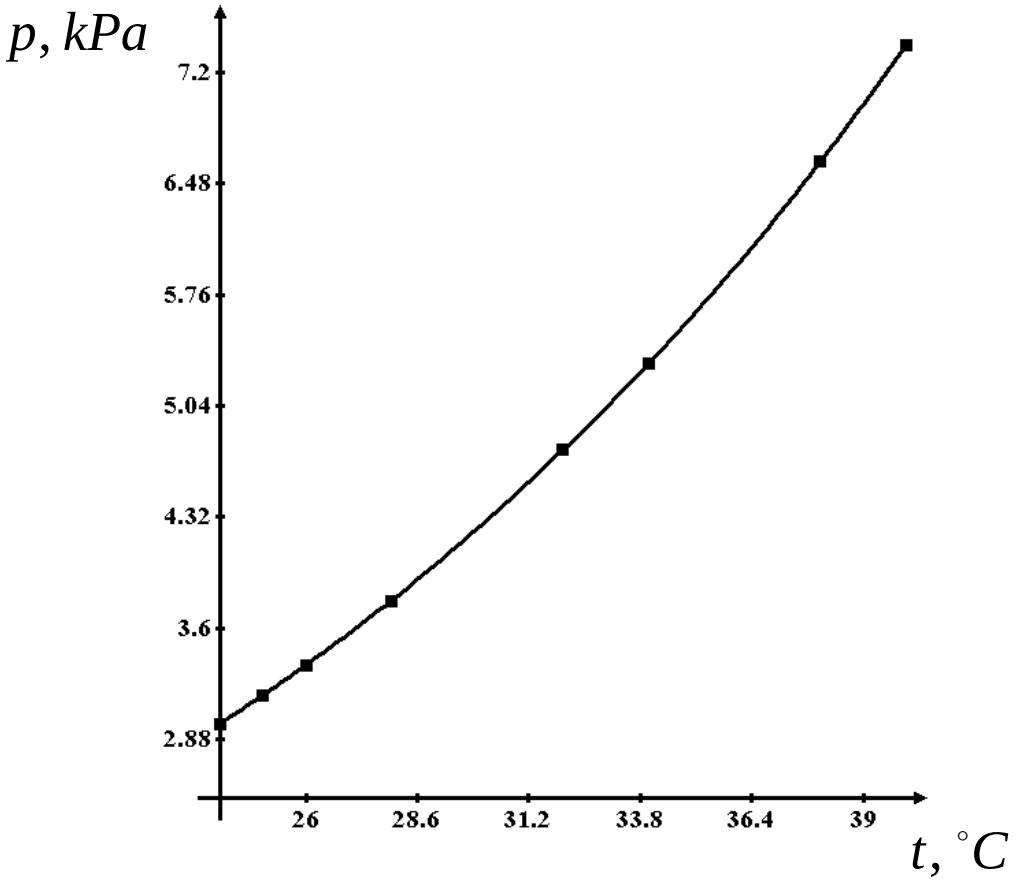

Рис. 43. Интерполяция кубическими сплайнами зависимости табл. 8. Жирные точки – табличные значения; сплошная линия – интерполирующая кривая

Пример 2. Очень часто для быстро меняющихся функций результаты интерполяции бывают неудовлетворительными. Примером может служить табл. 9. Как видно из рис. 44, интерполирующая кривая, проходя точно через узловые точки, не является плавной монотонной кривой и не отвечает реалиям.

Таблица 9. Молярная электропроводность Λ CH_3COOH при 18 ºС (фрагмент таблицы)

Концентрация, моль/л	0.05	0.1	0.5	1	3	5
Λ, $см^2$ / Ом/ моль	6.48	4.60	2.01	1.32	0.54	0.285

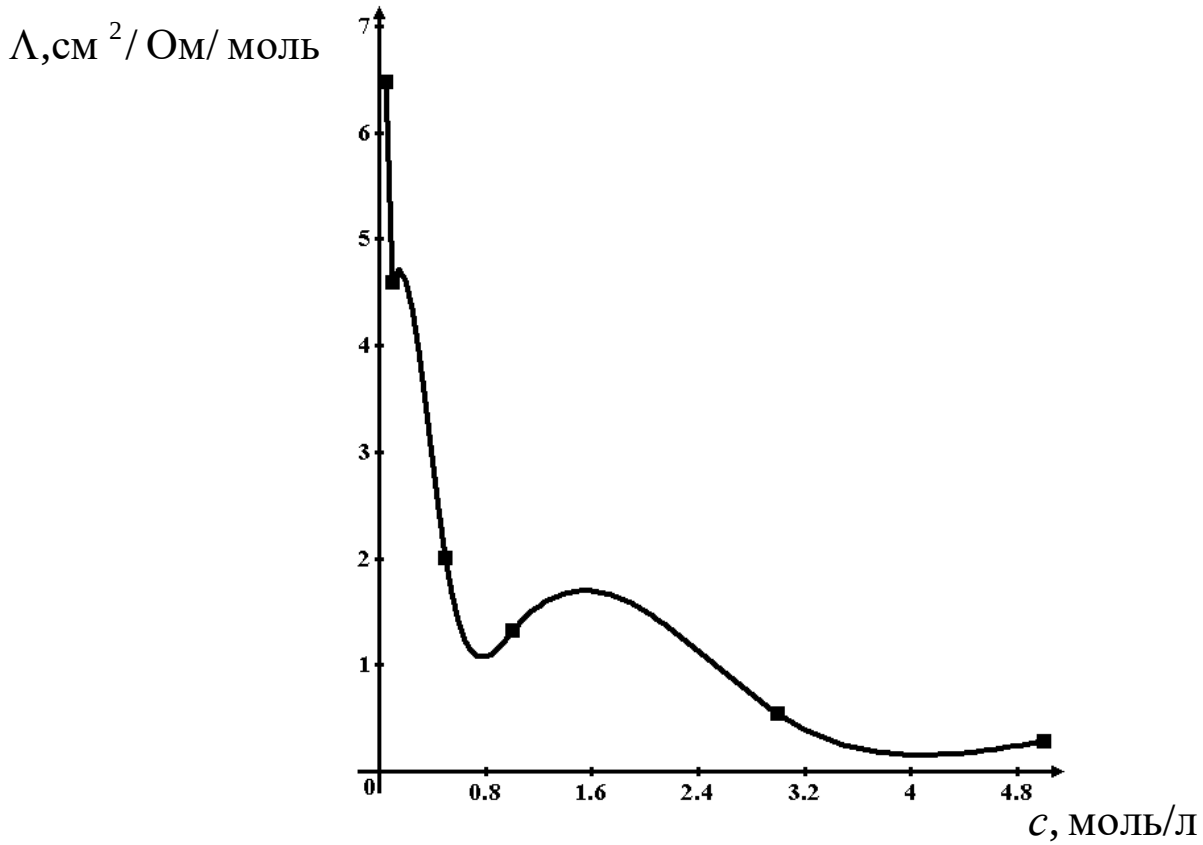

Рис. 44. Интерполяция сплайнами зависимости табл. 9. Жирные точки – табличные значения; сплошная линия – интерполирующая кривая

Пример 3. Положение дел в предыдущем примере можно поправить, если в том же диапазоне взять не 6 узловых точек, а больше, например – 34. В этом случае результаты интерполяции вполне удовлетворительны (рис. 45).

Из примеров 2 и 3 следует, что результаты интерполяции различными методами должны обязательно проверяться, хотя бы графически. Неудовлетворительные интерполяции должны пересчитываться с бóльшим числом узловых точек; если же такой возможности не существует (например, при отсутствии данных), можно применить методы аппроксимации. В последнем случае узловые точки будут воспроизводиться с погрешностью.

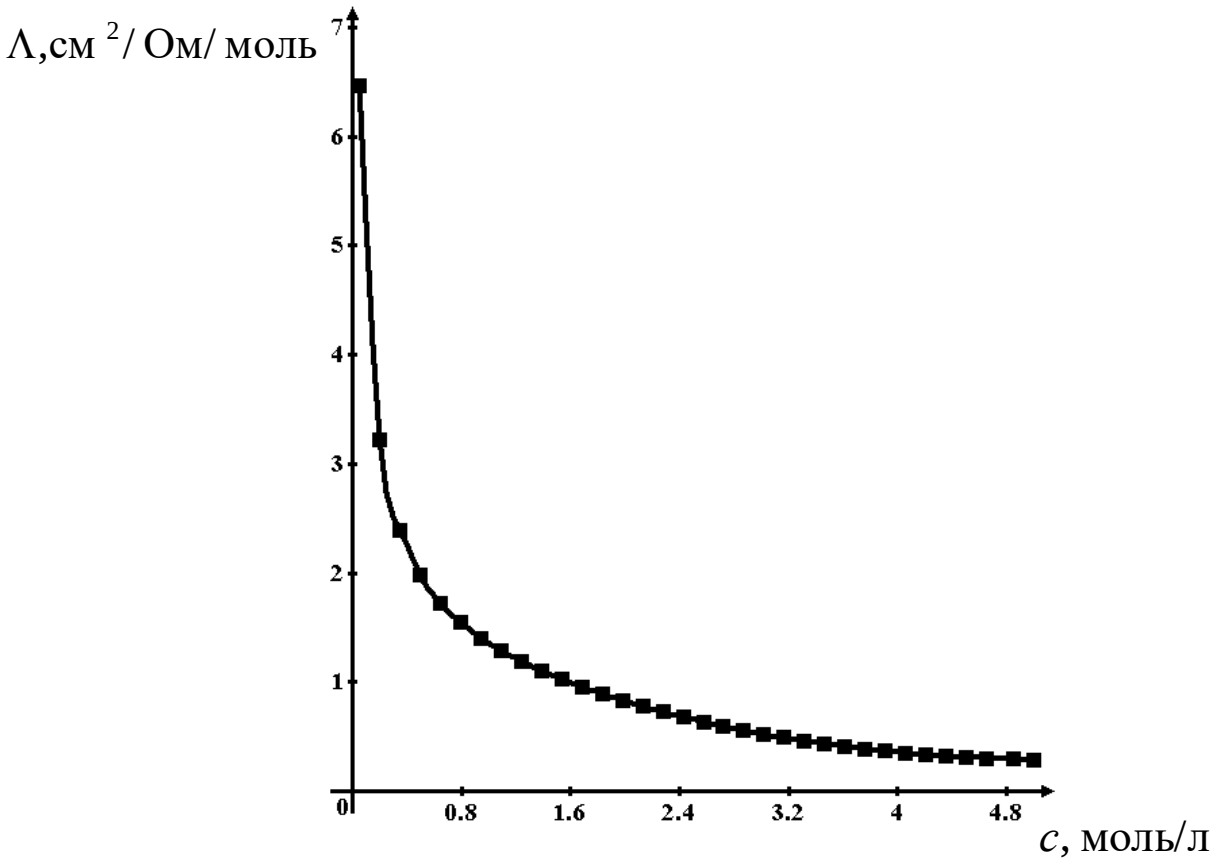

Рис. 45. Интерполяция сплайнами зависимости рис. 44 при 34 узловых точках

Глава 5. ЧИСЛЕННОЕ ИНТЕГРИРОВАНИЕ

§ 1. Введение

Взятие интеграла относится к операциям, которые далеко не всегда могут быть осуществлены методами высшей математики. В этом плане использование численных методов безальтернативно. Определённый интеграл на отрезке $[a, b]$ численно равен площади фигуры, ограниченной графиком функции, перпендикулярами, восстановленными из концов интервала до пересечения с графиком, и осью абсцисс (рис. 46), т. е. площади **криволинейной трапеции**. Причём, если график лежит выше оси абсцисс, то интеграл будет положительный, если ниже – отрицательный.

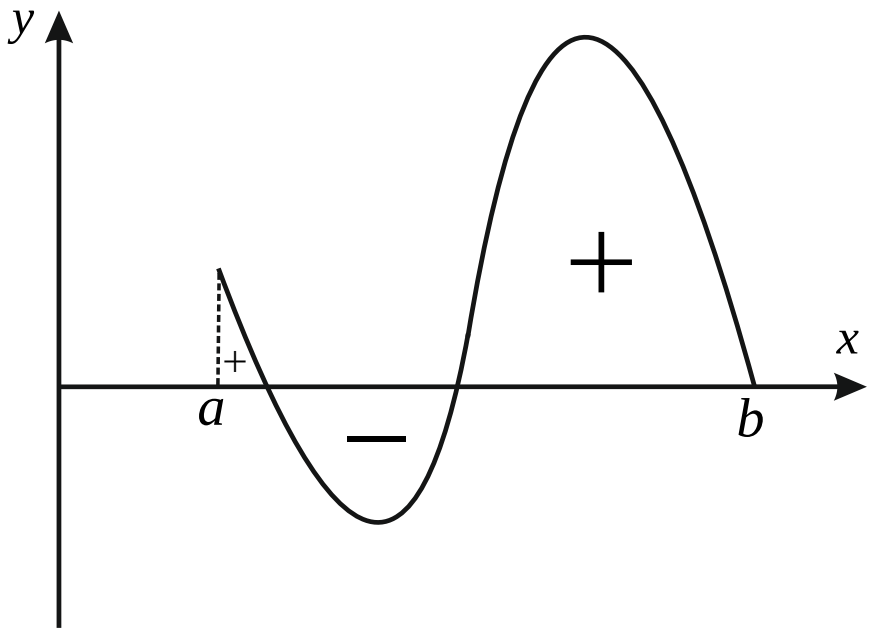

Рис. 46. Определённый интеграл численно равен площади, ограниченной графиком функции, перпендикулярами, восстановленными из концов интервала до пересечения с графиком, и осью абсцисс

В данном пособии будут рассмотрены три метода: прямоугольников, трапеций и Симпсона. Кроме этого существуют методы Ньютона – Котеса, Гаусса, Чебышёва и др. Некоторые из них более точные, но неудобны для программирования. Для большинства случаев рассмотренных методов будет достаточно.

§ 2. Метод прямоугольников

В этом методе площадь криволинейной трапеции заменяется площадью прямоугольников, построенных по $f(a)$ (рис. 47), по $f(b)$ (рис. 48) или по середине отрезка $f\left(\frac{a+b}{2}\right)$ (рис. 49). Во всех трёх случаях это будет площадь фигуры a-b-c-d.

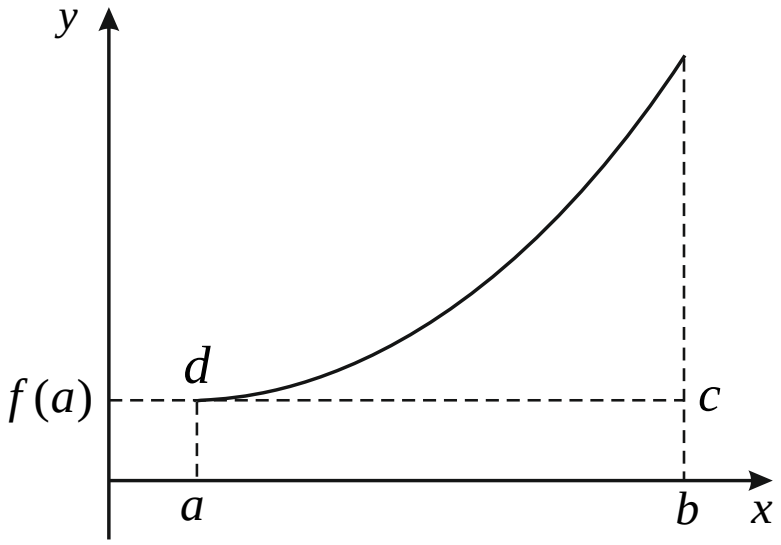

Рис. 47. Вариант метода прямоугольников
по левому концу интервала

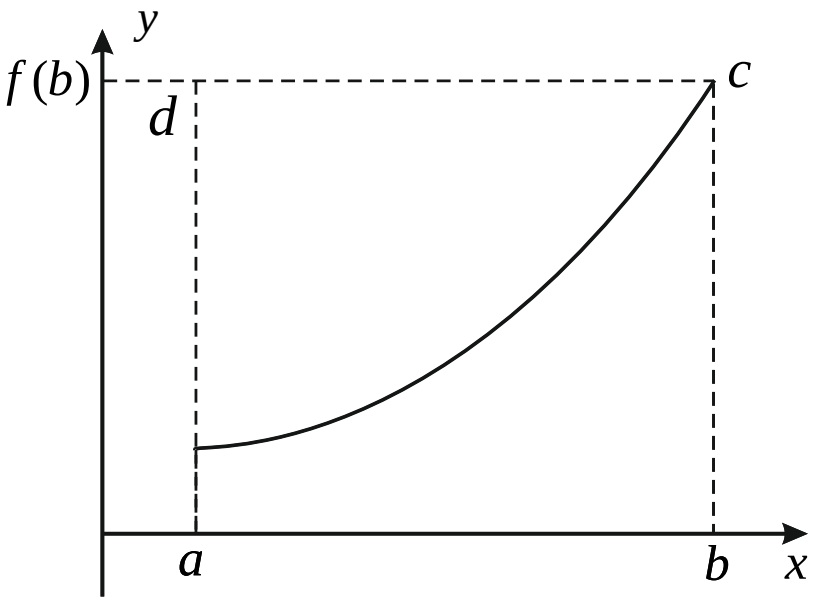

Рис. 48. Вариант метода прямоугольников
по правому концу интервала

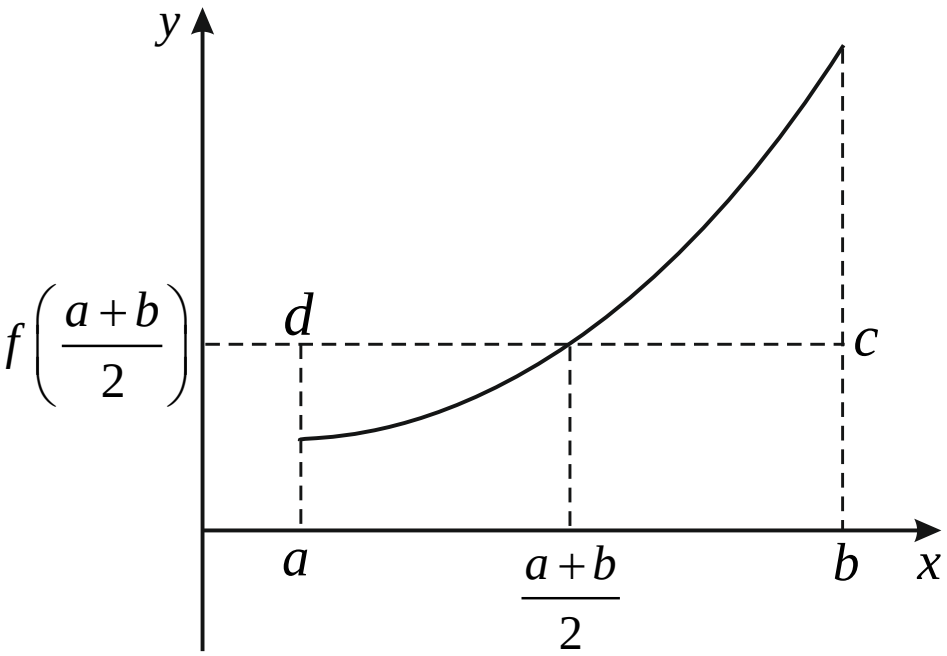

Рис. 49. Вариант метода прямоугольников по середине отрезка

Интегралы рассчитываются по формулам

$$I=(b-a)\,f(a);\quad I=(b-a)\,f(b);\quad I=(b-a)\,f\left(\frac{a+b}{2}\right). \quad (142)$$

Как видно из рис. 47 – 49, метод очень неточный; можно применить разбиение отрезка интегрирования на части, но это нецелесообразно, так как существуют более точные простые методы. Метод прямоугольников иногда используется как вспомогательный, например, для начального приближения интеграла.

§ 3. Метод трапеций

В методе трапеций площадь криволинейной трапеции заменяется площадью прямоугольной трапеции *a-b-c-d* (рис. 50). Как известно, площадь трапеции равна произведению полусуммы оснований на высоту:

$$I=(b-a)\frac{f(a)+f(b)}{2} \quad (143)$$

Из рис. 50 видно, что погрешность определения интеграла значительно уменьшилась. Ещё большую точность можно получить, если отрезок $[a, b]$ разбить на равные отрезки (рис. 51), задав число разбиений n, тогда мы получим $n + 1$ узел сетки, а любой x_i будет вычисляться по формуле

$$x_i = a + i \cdot h, \tag{144}$$

где h – шаг, определяется как

$$h = \frac{b-a}{n} \tag{145}$$

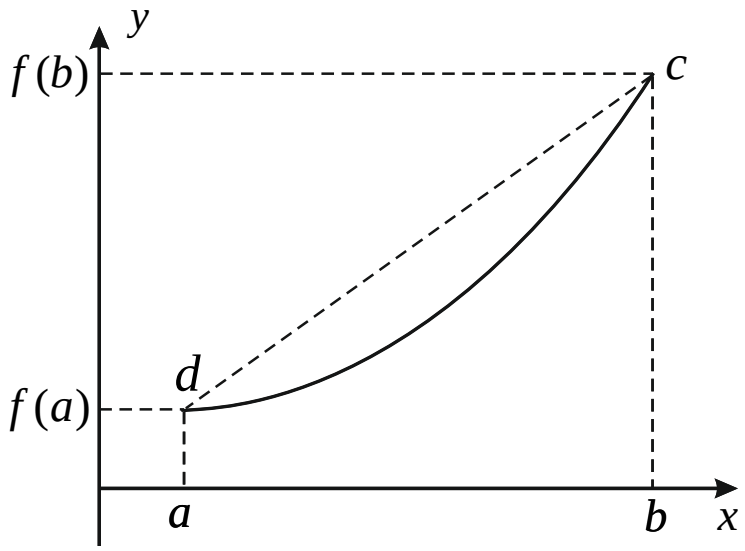

Рис. 50. Метод трапеций; интеграл приближённо равен площади трапеции *a-b-c-d*

Интеграл в этом случае определяется как сумма всех трапеций. Из рис. 51 видно, что погрешность интеграла сильно снизилась, причём число разбиений можно задать большим. **Расчётную формулу метода трапеций** легко получить, просуммировав площади трапеций (см. рис. 51 и (143)) для $n = 4$ с учётом $h = (b-a)/n$:

$$I = \frac{(b-a)}{n}\frac{f(a)+f(x_1)}{2} + \frac{(b-a)}{n}\frac{f(x_1)+f(x_2)}{2} +$$
$$+\frac{(b-a)}{n}\frac{f(x_2)+f(x_3)}{2} + \frac{(b-a)}{n}\frac{f(x_3)+f(b)}{2} =$$
$$= \frac{h}{2}\left(f(a)+f(b)+2\sum_{i=1}^{n-1} f(x_i)\right),$$

т. е., для общего случая

$$I = \frac{h}{2}\left(f(a) + f(b) + 2\sum_{i=1}^{n-1} f(x_i) \right). \qquad (146)$$

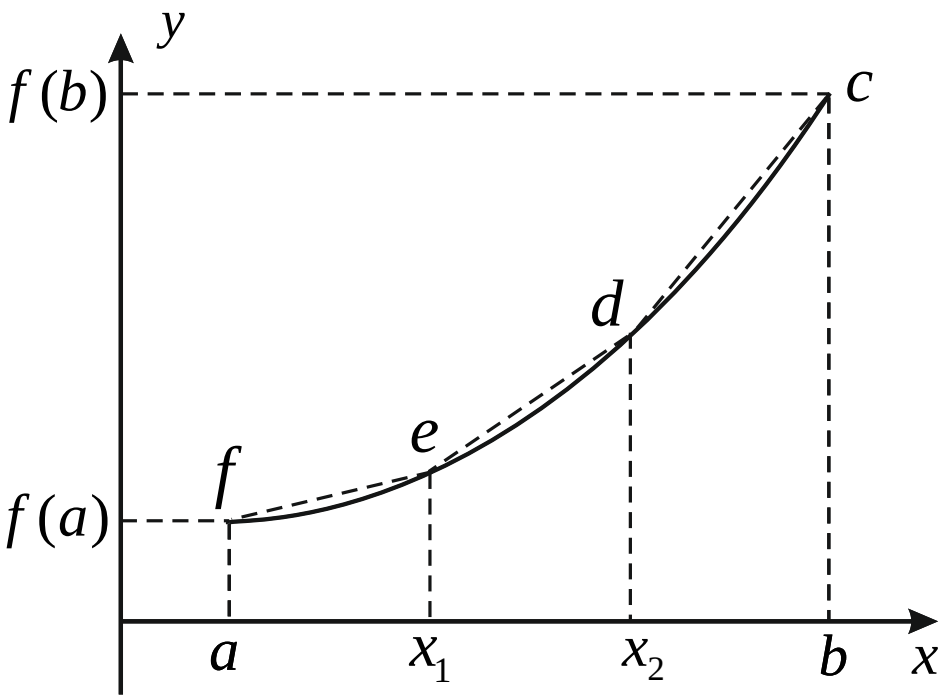

Рис. 51. Метод трапеций с разбиениями; интеграл приближённо равен сумме площадей всех трапеций

Программа. В программе задаётся точность, с которой нужно вычислить интеграл, это делается в итерационном цикле, в каждом следующем цикле увеличивается число разбиений.

```
program trapez;
var
//описание числа разбиений интервала от a до b и счётчика цикла;
n,i:integer;
//описание переменных: eps – точность, a и b – пределы
//интегрирования, h – шаг по x при числе разбиений n,
//I1и I2 – предыдущее и последующее значение интеграла:
eps,a,b,h,I1,I2: real;
```

```
//описание подинтегральной функций f(x) (например f),
//конкретный вид функций не приведён:
function f(x:real):real;begin <…>end;
//начало основной программы
begin
//ввод исходных данных
write('a=');readln(a);
write('b=');readln(b);
write('eps=');readln(eps);
//рассчитываем начальное приближение интеграла по методу
//прямоугольников (142); записываем его в переменную I2,
//так как в начале итерационного цикла идёт переприсваивание,
//и это значение попадёт в переменную I1:
I2:=f(a)*(b-a);
//начальное число разбиений:
n:=1;
//итерационный цикл (например – repeat);
repeat
//в начале цикла – переприсваивание:
I1:=I2;
//увеличение числа разбиений и расчёт нового шага:
n:=n+1;h:=(b-a)/n;
//вычисление нового приближения интеграла по (146);
//выход из итерационного цикла происходит, когда
//два последних приближения интеграла отличаются меньше,
//чем на eps:
I2:=f(a)+f(b); for i:=1 to n-1 do
I2:=I2+2*f(a+i*h);
I2:=I2*h/2;
until abs(I1-I2)<eps;
//вывод полученного результата и числа разбиений:
writeln('I=',I2,'  n=',n);
end.
```

Метод трапеций с разбиениями вполне удовлетворителен для большинства практических расчётов. Также его очень удобно приме-

нять как вспомогательный в численных решениях интегральных и интегро-дифференциальных уравнений, так как расчётная формула достаточно проста (146) и легко может быть встроена в основной алгоритм этих методов.

§ 4. Метод Симпсона

В методе Симпсона отрезок интегрирования делится пополам (это не считается разбиением!) и истинная кривая интерполируется квадратным трёхчленом (рис. 52) (в методе трапеций мы имели дело с линейной интерполяцией). Как видно из рисунка, даже в отсутствие разбиений интерполирующая кривая очень близко подходит к истинной, если та достаточно гладкая. Если же сделать разбиения, то на графике расхождений вообще не будет видно.

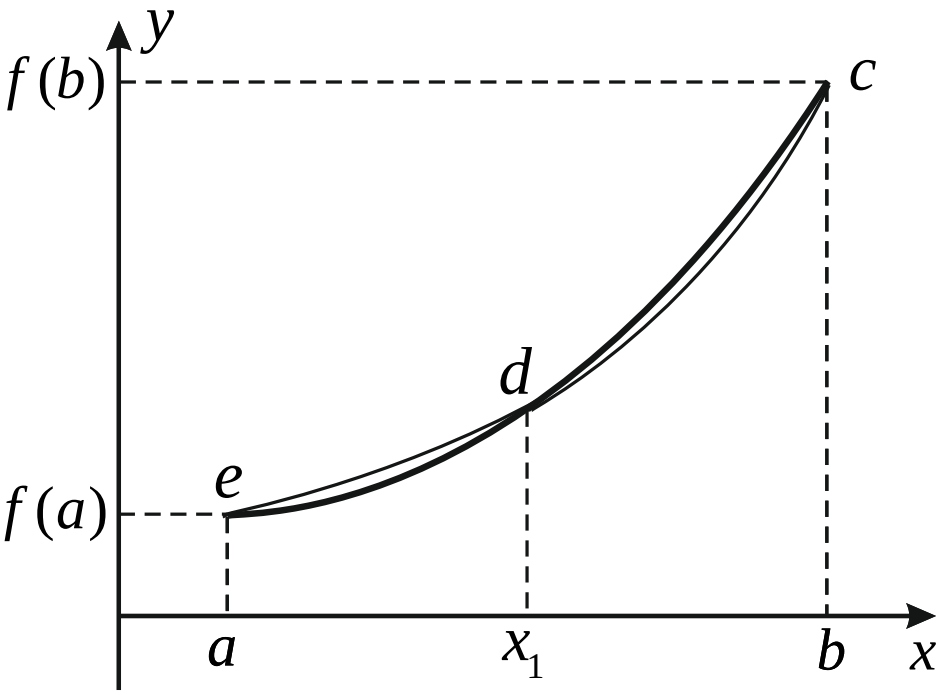

Рис. 52. Метод Симпсона (без разбиений). Жирная кривая – истинная функция, тонкая – интерполяционная кривая

Выведем расчётную формулу метода Симпсона, основываясь на первой интерполяционной формуле Ньютона (116). Сначала рассмотрим ситуацию без разбиений (см. рис. 52). Формально шагом будет величина $h=(b-a)$; временно примем за шаг величину, вдвое меньшую,

т. е. проведём интерполяцию по точкам $a,\ x_1,\ b;\ n=2$. В данном случае формула Ньютона примет вид

$$y = y_0 + q\Delta y_0 + \frac{q(q-1)}{2!}\Delta^2 y_0, \tag{147}$$

где

$$q = \frac{x - x_0}{h} \tag{148}$$

Продифференцируем (148); промежуточное значение x считается переменной:

$$dq = d\,\frac{x - x_0}{h} = \frac{dx}{h}. \tag{149}$$

Распишем конечные разности в (147) по (117)

$$\Delta y_0 = (y_1 - y_0),$$
$$\Delta^2 y_0 = (y_2 - y_1) - (y_1 - y_0) = (y_2 - 2y_1 + y_0)$$

и подставим их в (147):

$$y = y_0 + q(y_1 - y_0) + \frac{1}{2}q(q-1)(y_2 - 2y_1 + y_0). \tag{150}$$

Интеграл (см. рис. 49) с учётом $a = x_0$ и $b = x_2$ будет выражаться

$$I = \int_{x_0}^{x_2} y\,dx = h\int_{x_0}^{x_2} y\,\frac{dx}{h}. \tag{151}$$

(умножим и разделим на h). В (151) заменим y на (150), а $\frac{dx}{h}$ на dq (см. (149)):

$$\begin{aligned} I &= h\int_{x_0}^{x_2}\left(y_0 + q(y_1 - y_0) + \frac{q}{2}(q-1)(y_0 - 2y_1 + y_2)\right)dq = \\ &= h\int_{x_0}^{x_2}\left(\frac{y_0 - 2y_1 + y_2}{2}q^2 + \left((y_1 - y_0) - \frac{y_0 - 2y_1 + y_2}{2}\right)q + y_0\right)dq, \end{aligned} \tag{152}$$

здесь применено равенство

$$\left((y_1-y_0)-\frac{y_0-2y_1+y_2}{2}=\frac{1}{2}\left(4y_1-3y_0-y_2\right)\right).$$

Тогда с помощью последовательных преобразований окончательно получаем

$$I=\frac{h\left(y_0-2y_1+y_2\right)}{2\cdot 3}q^3\Big|_{x_0}^{x_2}+\frac{h\left(4y_1-3y_0-y_2\right)}{2\cdot 2}q^2\Big|_{x_0}^{x_2}+h\,y_0 q\Big|_{x_0}^{x_2}=$$

$$=\frac{h\left(y_0-2y_1+y_2\right)}{6}\left(\frac{(x_2-x_0)^3}{h^3}-0\right)+$$

$$+\frac{h\left(4y_1-3y_0-y_2\right)}{4}\left(\frac{(x_2-x_0)^2}{h^2}-0\right)+h\,y_0\left(\frac{x_2-x_0}{h}-0\right)=$$

$$=\frac{h\left(y_0-2y_1+y_2\right)}{6}\cdot\frac{8h^3}{h^3}+\frac{h\left(4y_1-3y_0-y_2\right)}{4}\cdot\frac{4h^2}{h^2}+h\,y_0\,\frac{2h}{h}=$$

$$=\frac{h}{3}\left(4y_0-8y_1+4y_2+12y_1-9y_0-3y_2+6y_0\right)=$$

$$=\frac{h}{3}\left(y_0+4y_1+y_2\right). \qquad (153)$$

Увеличиваем шаг в два раза, так как ранее мы его уменьшили:

$$h:=2h; \quad \text{т. е.} \quad h_{new}=2h_{old} \qquad (154)$$

и получаем для общего случая

$$\boxed{I=\frac{h}{6}\sum_{i=1}^{n}\left(f(x_i)+4f\left(x_i-\frac{h}{2}\right)+f(x_{i-1})\right)}. \qquad (155)$$

Программа. Как и в предыдущем методе, в программе задаётся точность, с которой нужно вычислить интеграл, это делается в итерационном цикле, в каждом следующем цикле увеличивается число разбиений.

```
program simpson;
var
//описание числа разбиений интервала от a до b и счётчика цикла;
n,i:integer;
//описание переменных: eps – точность, a и b – пределы
//интегрирования, h – шаг по x при числе разбиений n,
//I1и I2 – предыдущее и последующее значение интеграла:
eps,a,b,h,I1,I2: real;
//описание подинтегральной функций f(x) (например f),
//конкретный вид функций не приведён:
function f(x:real):real;begin <…>end;
//начало основной программы
begin
//ввод исходных данных
write('a=');readln(a);
write('b=');readln(b);
write('eps=');readln(eps);
//рассчитываем начальное приближение интеграла по методу
//прямоугольников; записываем его в переменную I2,
//так как в начале итерационного цикла идёт переприсваивание,
//и это значение попадёт в переменную I1:
I2:=f(a)*(b-a);
//начальное число разбиений:
n:=1;
//итерационный цикл (например – repeat);
repeat
//в начале цикла – переприсваивание:
I1:=I2;
//увеличение числа разбиений и расчёт нового шага:
n:=n+1;h:=(b-a)/n;
//вычисление нового приближения интеграла по (155);
//выход из итерационного цикла происходит, когда
//два последних приближения интеграла отличаются меньше,
//чем на eps:
I2:=0; for i:=1 to n do
I2:=I2+f(a+i*h)+4*f((a+i*h)-h/2)+f((a+(i-1)*h));
I2:=I2*h/6;
```

```
until abs(I1-I2)<eps;
//вывод полученного результата и числа разбиений:
writeln('I=',I2,'  n=',n);
end.
```

Пример

Вычислим интеграл

$$I = \int_{0.6}^{1.4} x^2 \cos x$$

по методам трапеций и Симпсона с заданной точностью 0.00001 и выведем количество разбиений n, которое будет необходимо сделать для достижения этой точности по одному и другому методу.

Результаты расчётов:

по методу трапеций $I = 0.37207,\ n = 63$;

по методу Симпсона $I = 0.37210,\ n = 4$.

КОНТРОЛЬНЫЕ ЗАДАНИЯ

Задание № 1. Графически отделить корни и решить нелинейные уравнения методами деления отрезка пополам, хорд, простых итераций и Ньютона с заданной точностью $\varepsilon = 0.000001$. Сделать необходимое предварительное математическое исследование и сравнить полученные результаты.

1. $3x = 1 + \cos x$.

2. $x - \sin x = 0.25$.

3. $3x - e^x = 0$.

4. $2\sin(x - 0.6) = 1.5 - x$.

5. $(x-1)^2 = x/2$.

6. $\sin(x + 0.5) = 2x - 0.5$.

Задание № 2. Считая x и y результатами экспериментов, провести нелинейную аппроксимацию для $n = 1$, $n = 2$, $n = 3$ и получить нормальные системы уравнений для этих степеней полинома. Результаты вывести на экран, в текстовый файл (для контроля) и в бинарный файл. Решить системы методом исключения Гаусса (задание № 3) и записать аппроксимирующие уравнения. Построить график, где в одних осях отметить экспериментальные точки и провести аппроксимирующие кривые (для $n = 1$ – прямая) в экспериментальном диапазоне. Сравнить между собой результаты трёх аппроксимаций и сделать вывод.

1.

№	1	2	3	4	5	6	7	8	9	10
x	8	10	12	14	16	18	20	22	24	26
y	0.04	0.47	0.78	1.01	1.19	1.60	1.93	2.22	2.50	3.01

№	11	12	13	14	15	16	17	18	19	20
x	28	30	32	34	36	38	40	42	44	46
y	3.22	3.71	4.23	4.78	5.27	5.75	6.16	6.76	7.30	8.00

2.

№	1	2	3	4	5	6	7	8
x	0.03	0.06	0.09	0.12	0.15	0.18	0.21	0.24
y	-1.89	-2.07	-2.30	-2.26	-2.34	-2.66	-2.88	-2.85

№	9	10	11	12	13	14	15	16
x	0.27	0.30	0.33	0.36	0.39	0.42	0.45	0.48
y	-3.16	-3.49	-3.88	-4.22	-4.45	-4.99	-5.36	-5.71

№	17	18	19	20
x	0.51	0.54	0.57	0.60
y	-6.51	-6.76	-7.35	-8.02

3.

№	1	2	3	4	5	6	7	8	9	10
x	14	16	18	20	22	24	26	28	30	32
y	2.09	2.31	2.72	2.77	2.78	2.97	3.00	3.51	3.43	3.58

№	11	12	13	14	15	16	17	18	19	20
x	34	36	38	40	42	44	46	48	50	52
y	3.58	3.54	3.82	3.90	3.77	3.81	4.00	3.97	4.08	4.08

4.

№	1	2	3	4	5	6	7	8
x	0.12	0.15	0.18	0.21	0.24	0.27	0.30	0.33
y	-1.90	-1.80	-1.82	-1.86	-1.83	-2.00	-2.01	-2.05

№	9	10	11	12	13	14	15	16
x	0.36	0.39	0.42	0.45	0.48	0.51	0.54	0.57
y	-2.46	-2.68	-2.85	-2.98	-3.30	-3.40	-3.90	-4.37

№	17	18	19	20
x	0.60	0.63	0.66	0.69
y	-4.65	-5.00	-5.42	-6.13

5.

№	1	2	3	4	5	6	7	8	9	10
x	0.8	1.0	1.2	1.4	1.6	1.8	2.0	2.2	2.4	2.6
y	2.15	2.41	2.58	2.84	3.28	3.46	4.02	4.11	4.61	5.03

№	11	12	13	14	15	16	17	18	19	20
x	2.8	3.0	3.2	3.4	3.6	3.8	4.0	4.2	4.4	4.6
y	5.34	5.86	6.33	6.81	7.21	7.67	8.23	8.68	9.35	9.93

6.

№	1	2	3	4	5	6	7	8	9	10
x	0.08	0.10	0.12	0.14	0.16	0.18	0.20	0.22	0.24	0.26
y	2.18	2.43	2.40	2.43	2.65	2.75	2.67	2.66	2.63	2.75

№	11	12	13	14	15	16	17	18	19	20
x	0.28	0.30	0.32	0.34	0.36	0.38	0.40	0.42	0.44	0.46
y	2.41	2.24	2.12	1.74	1.57	1.17	0.96	0.63	0.25	0.08

Задание № 3. Решить методом исключения Гаусса СЛАУ, полученные в задании № 2 (для $n=1,\ n=2,\ n=3$). Матрицу при неизвестных и вектор свободных членов считать из соответствующего бинарного файла. Сделать проверку решения.

Задание № 4. Решить методом Гаусса – Зейделя относительно x_0, x_1, x_2 СЛАУ, для которых приведены матрицы коэффициентов и векторы свободных членов:

1. $\left(\begin{array}{ccc|c} -6 & 3 & -1 & 2 \\ 2 & 8 & 4 & 6 \\ -1 & 2 & 7 & 3 \end{array}\right)$.
2. $\left(\begin{array}{ccc|c} 4 & 2 & -1 & 5 \\ 3 & 6 & -2 & 7 \\ -1 & 2 & 8 & 9 \end{array}\right)$.
3. $\left(\begin{array}{ccc|c} 7 & -1 & 3 & 4 \\ 1 & 6 & -2 & 3 \\ 2 & -4 & 10 & 5 \end{array}\right)$.
4. $\left(\begin{array}{ccc|c} 5 & 2 & -1 & 6 \\ 2 & 10 & 4 & 5 \\ -1 & 6 & 9 & 3 \end{array}\right)$.
5. $\left(\begin{array}{ccc|c} 20 & 5 & -10 & 10 \\ 1 & 6 & -2 & 3 \\ 3 & 3 & -15 & 5 \end{array}\right)$.
6. $\left(\begin{array}{ccc|c} 4 & 1 & 2 & -1 \\ 2 & 8 & 3 & -5 \\ 4 & -1 & 9 & 4 \end{array}\right)$.

Задание № 5. Для приведённых табличных данных «Давления насыщенного водяного пара в равновесии с водой» найти значение функции для указанной температуры по интерполяционной формуле Ньютона.

1. Для t = 11.4 ºС

t, ºС	10	12	14	16
p, кПа	1.2270	1.4014	1.597	1.817

2. Для t = 20.8 ºС

t, ºС	18	20	22	24
p, кПа	2.062	2.337	2.642	2.982

3. Для t = 31.3 ºС

t, ºС	26	28	30	32
p, кПа	3.360	3.778	4.241	4.753

4. Для t = 37.2 ºС

t, ºС	34	36	38	40
p, кПа	5.318	5.940	6.623	7.374

5. Для t = 74.3 ºС

t, ºС	50	60	70	80
p, кПа	12.334	19.92	31.16	43.36

6. Для $t = 92.6$ ºС

t, ºС	90	100	110	120
p, кПа	70.10	101.32	142.26	198.5

Задание № 6. Найти определённый интеграл с заданной точностью $\varepsilon = 0.000001$ по методам трапеций и Симпсона с выводом числа разбиений. Сравнить полученные результаты.

1. $\int_{1.2}^{2.3} \frac{\ln(x+2)}{x} dx;$

2. $\int_{0.1}^{4.3} x \cos x^3 dx;$

3. $\int_{1.1}^{3.8} x^2 e^{-x} dx;$

4. $\int_{0.8}^{1.9} \frac{\cos x^2}{x+1} dx;$ при $x = 1;$

5. $\int_{0.4}^{2.1} \cos(x + x^3) dx;$ при $x = 3;$

6. $\int_{0.2}^{3.1} x^2 \cos x \, dx;$ при $x = 1;$

ЗАКЛЮЧЕНИЕ

Представленный в учебном пособии курс – вторая часть математически ориентированных химических дисциплин, таких как численные методы в химии, математические методы в химии и химической технологии, математическое моделирование, системное моделирование химико-технологических процессов, компьютерная химия. Также он может использоваться в таких дисциплинах, как аналитическая химия, физическая химия, строение вещества, коллоидная химия, химическая технология.

В пособии изложены численные методы решения алгебраических задач, к которым сводятся реальные задачи химии и химической технологии – такие, как решение нелинейных уравнений, систем линейных и нелинейных уравнений, взятие определённого интеграла; рассмотрена практическая реализация этих методов на Паскале как базовом языке для Delphi и Lasarus, в виде конкретных отлаженных программ, которые могут быть в основном без изменений прокомпилированы в указанных средах программирования. По большинству методов приведены примеры решения задач с использованием этих программ.

Методы решения нелинейных уравнений – дихотомии, хорд, простых итераций и Ньютона не охватывают всех существующих методов, но достаточны для профессионального выбора метода в зависимости от поставленной конкретной задачи. Если скорость решения не критична, а предварительное математическое исследование нелинейного уравнения невозможно или затруднительно, выбор следует остановить на первых двух методах. Их же лучше использовать как вспомогательные при решении других вычислительных задач. Если требуется быстрота и точность решения, а функции, входящие в уравнения, дифференцируемы, применять лучше последние два метода.

Методы аппроксимации функций показаны наиболее распространённым на практике методом – методом наименьших квадратов (м. н. к.) – основным методом при обработке результатов химического

эксперимента, когда снимается функциональная зависимость. Описаны также часто встречающиеся на практике неполиномиальная аппроксимация и линеаризация, недостатки м. н. к.

Задача решения систем линейных и нелинейных уравнений чаще всего вспомогательная. К ней часто сводится, например, решение дифференциальных уравнений и интерполирование функций. Рассмотрены точные методы – Гаусса и прогонки, а также приближённый метод – метод Гаусса – Зейделя. Для решения систем линейных уравнений разобран один из самых действенных методов – Ньютона.

Методы интерполирования функций представлены в пособии интерполяцией по Ньютону, Лагранжу и методом кубических сплайнов. Освоение этих методов позволит будущим специалистам профессионально сделать выбор того или иного метода в зависимости от стоящей перед ними задачи. Специально отмечена необходимость проверки полученных результатов интерполяции.

Изложены методы численного интегрирования – трапеций и Симпсона, как наиболее простые, обладающие достаточной точностью и могущие использоваться как вспомогательные при решении других задач. Другие методы, оставлены на самостоятельное изучение.

РЕКОМЕНДАТЕЛЬНЫЙ БИБЛИОГРАФИЧЕСКИЙ СПИСОК*

Основной

1. Самарский, А. А. Численные методы / А. А. Самарский, А. В. Гулин. – М. : Наука, 1989.

2. Джонсон, К. Численные методы в химии / К. Джонсон. – М. : Мир, 1983.

3. Вержбицкий, В. М. Основы численных методов / В. М. Вержбицкий. – М. : Высш. шк., 2002.

4. Кларк, Т. Компьютерная химия / Т. Кларк. – М. : Мир, 1990.

5. Бахвалов, Н. С. Численные методы / Н. С. Бахвалов, Н. П. Жидков, Г. М. Кобельков. – М. ; СПб. : Физматлит, 2001.

6. Калиткин, Н. Н. Численные методы / Н. Н. Калиткин. – М. : Наука, 1978.

7. Демидович, Б. П. Основы вычислительной математики / Б. П. Демидович, И. А. Марон. – М. : Наука, 1966.

Дополнительный

1. Пирумов, У. Г. Численные методы / У. Г. Пирумов. – М. : Дрофа, 2003.

2. Тихонов, А. Н. Уравнения математической физики / А. Н. Тихонов, А. А. Самарский. – М. : Наука, 1977.

3. Плис, А. И. Лабораторный практикум по вычислительной математике / А. И. Плис, Н. А. Сливина. – М. : Высш. шк., 1984.

4. Численные методы алгебры и приближения функций: метод. указания к выполнению лаб. работ по курсу «Численные методы» / сост.: Г. А. Кокотушкин, А. А. Федотов, П. В. Храпов. – М. : Изд-во МГТУ им. Н. Э. Баумана, 2011.

* Публикуется в авторской редакции.

Учебное издание

ЛОБКО Владимир Николаевич

МАТЕМАТИЧЕСКИЕ МЕТОДЫ В ХИМИИ И ХИМИЧЕСКОЙ ТЕХНОЛОГИИ

Численные методы решения алгебраических задач и обработки функций

Учебное пособие

Редактор Е. В. Невская

Технический редактор С. Ш. Абдуллаева

Корректор Н. В. Пустовойтова

Компьютерная верстка Л. В. Макаровой, А. Н. Герасина

Выпускающий редактор А. А. Амирсейидова

Подписано в печать 27.12.19.
Формат 60×84/16. Усл. печ. л. 8,37. Тираж 50 экз.
Заказ
Издательство
Владимирского государственного университета
имени Александра Григорьевича и Николая Григорьевича Столетовых.
600000, Владимир, ул. Горького, 87.