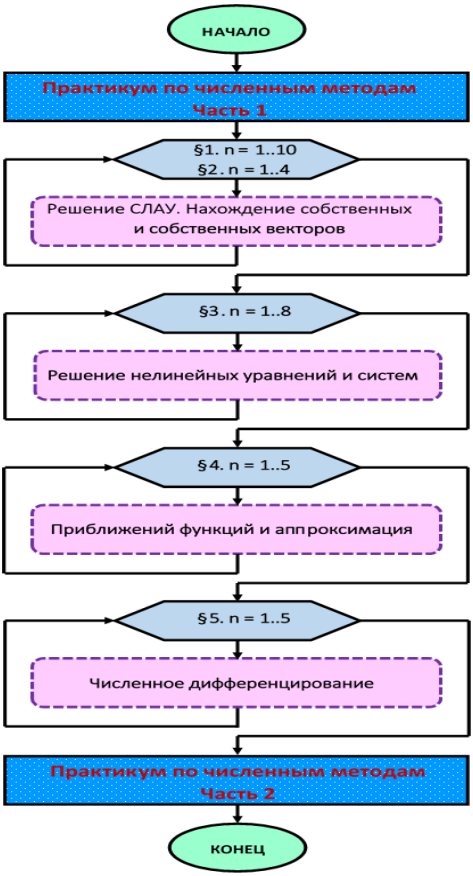
А. С. Банников, И. Г. Ким, Н. В. Латыпова

ЧИСЛЕННЫЕ МЕТОДЫ

Учебное пособие

Часть 1



Ижевск 2018

Министерство науки и высшего образования Российской Федерации
ФГБОУ ВО «Удмуртский государственный университет»
Институт математики, информационных технологий и физики

А. С. Банников, И. Г. Ким, Н. В. Латыпова

ЧИСЛЕННЫЕ МЕТОДЫ

Учебное пособие

Часть 1

Ижевск
2018

УДК 519.6(075)
ББК 22.19я7
Б 232

Рекомендовано к изданию Учебно-методическим советом УдГУ

Рецензент: д. ф.-м. н., профессор Н. Н. Петров

Банников А. С., Ким И. Г., Латыпова Н. В.

Б 232 Численные методы: учеб. пособие. Ч. 1. — 2-е изд., испр. и доп. — Ижевск: Издательский центр «Удмуртский университет», 2018.— 80 с.

ISBN 978-5-4312-0642-9
ISBN 978-5-4312-0643-6 (Часть 1)

В данном пособии излагаются основные понятия, формулы и алгоритмы курса «Численные методы». В первой части рассматриваются численное решение систем линейных алгебраических уравнений, обращение матриц, полная и частичная проблемы собственных значений, решение нелинейных уравнений и систем таких уравнений. Изучаются некоторые методы для задач аппроксимации и приближения функций, рассматриваются задачи численного дифференцирования. Даны варианты лабораторных работ для лабораторного практикума с рекомендациями по их решению в пакете `Mathematica`.

Пособие предназначено для студентов всех направлений института математики, информационных технологий и физики.

УДК 519.6(075)
ББК 22.19.1я7

ISBN 978-5-4312-0642-9
ISBN 978-5-4312-0643-6 (Часть 1)

Оглавление

Предисловие

Курс «Численные методы» традиционно считается сложным у студентов, так как в нем требуется сочетать умения и навыки программирования со знаниями базовых математических курсов. Поэтому данное учебное пособие призвано помочь студентам в освоении данной дисциплины. Основные цели и задачи курса: научить правильно оценивать результаты вычислений, выполненных на компьютере средствами языков программирования; ознакомить с методами получения аналитических зависимостей при обработке результатов экспериментов; ознакомить с признаками плохо обусловленных и некорректных задач и подходами к их решению; сформировать культуру критичного отношения к полученным результатам.

Предлагаемое пособие является вторым изданием к данному курсу, исправленным и дополненным.

Пособие снабжено теоретической частью, в которой кратко изложены формулы и алгоритмы классических методов. При оформлении справочной части были использованы материалы [3, 5, 6, 14]. Для самостоятельной работы студентов предложен лабораторный практикум с индивидуальными заданиями. Лабораторные работы курса направлены на углубление и расширение лекционного материала, приобретение вычислительного опыта при решении поставленных задач.

Дополнением первого издания являются методические рекомендации по выполнению лабораторных работ в пакете `Mathematica`.

Издание предназначено для студентов бакалавриата направлений по укрупненным группам: 01.00.00 «Математика и механика», 02.00.00 «Компьютерные и информационные науки». Кроме того, пособие может быть использовано и студентами инженерных направлений, а также будет полезно всем, кто интересуется вычислительной математикой.

1. Методы решения систем линейных алгебраических уравнений

Все прямые методы основаны на том, что исходная система приводится к эквивалентной, но имеющей более простую форму системе, которую легче решать. Рассмотрим систему

$$Ax = b, \text{ или } \begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1, \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2, \\ \cdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n = b_n, \end{cases} \tag{1}$$

где A — квадратная матрица порядка n, $\det A \neq 0$, x_k — неизвестные величины, a_{ij} — заданные элементы матрицы системы уравнений.

В отличие от прямых методов, итерационные методы могут давать точное решение системы лишь как результат единообразного процесса итераций. Методы порождают последовательность приближенных решений $x^{(k)}$, и исходная матрица участвует лишь в матрично–векторном умножениях. При оценке качества итерационного процесса самым важным вопросом является сходимость полученной последовательности векторов к решению системы и скорость этой сходимости. Не лишним здесь будет и свойство самоисправляемости таких методов. Это свойство делает их менее чувствительными по сравнению с точными методами к отдельным ошибкам, допущенным при вычислениях.

1.1. Метод Гаусса

Алгоритм метода Гаусса состоит из двух этапов. Первый этап называется *прямым ходом* метода и заключается в приведении матрицы системы к треугольному виду по формулам: $k = 1, 2, \ldots, n-1$

$$a_{ij} := a_{ij} - \frac{a_{ik}}{a_{kk}} a_{kj}, \ j = k, k+1, \ldots, n;$$

$$b_i := b_i - \frac{a_{ik}}{a_{kk}} b_k, \ i = k+1, k+2, \ldots, n.$$

Таким образом, выполнив $(n-1)$ шаг, мы получим систему с верхней треугольной матрицей, причем эта система эквивалентна исходной, а элемент a_{kk}, на который происходит деление, называется *ведущим*

элементом на k-м шаге. Решение системы с треугольной матрицей выписывается явно и называется *обратным ходом метода Гаусса*:

$$x_k = \frac{b_k - \sum\limits_{j=k+1}^{n} a_{kj}x_j}{a_{kk}},\ k = n, n-1, \ldots, 1.$$

Эти формулы имеют смысл, если только все ведущие элементы отличны от нуля: $a_{kk} \neq 0$. Но эти ограничения обременительны, поэтому применяется *метод Гаусса с выбором главного элемента.* На первом шаге среди всех элементов первого столбца находим максимальный по модулю элемент: $|a_{k1}| = \max\limits_{1 \leqslant i \leqslant n} |a_{i1}|$. Далее k-ю строку меняем с первой строкой местами, при этом найденный максимальный элемент станет ведущим элементом, и он отличен от нуля. Выполняем первый шаг метода Гаусса, то есть зануляем все элементы первого столбца под ведущим элементом. Далее ищем максимальный по модулю элемент среди a_{i2}, где $2 \leqslant i \leqslant n$. Следующие шаги делаются аналогично, и всегда максимальный по модулю элемент будет отличен от нуля. Если $\det A \neq 0$, то все ведущие элементы, полученные методом Гаусса с выбором главного элемента, будут отличны от нуля. Поэтому ни на каком шаге деления на нуль не будет. И наоборот, если на каком-то шаге найденный максимальный по модулю элемент окажется равным нулю, то это указывает, что определитель исходной матрицы был равен нулю. Точность результатов будет определяться точностью выполнения арифметических операций при преобразовании элементов матрицы.

Замечание. Вместо максимального по модулю элемента можно использовать любой ненулевой элемент. Но поскольку происходит деление на этот элемент, то лучше всего использовать максимальный, что даст минимальную погрешность.

Можно использовать *метод Жордано–Гаусса.* Этот метод отличается от обычного метода Гаусса тем, что на каждом шаге прямого хода, после нахождения ведущего элемента, зануляются элементы столбца не только ниже ведущего элемента, но и выше, то есть матрица A приводится к диагональному виду. Тогда решение записывается как $x_i = \dfrac{b_i}{a_{ii}}$, где $i = 1, 2, \ldots, n$. Если еще на каждом шаге перед занулением других элементов столбца ведущую строку делить на ведущий элемент, то в результате прямого хода получится единичная

матрица. А значит, обратный ход не потребуется, так как на месте столбца правой части получим решение исходной системы.

1.2. Метод LU-разложения

Теорема. *Если все главные миноры квадратной матрицы A отличны от нуля, то существуют матрицы L и U, где L — нижняя треугольная, U — верхняя треугольная матрица, и $A = LU$. Если же диагональные элементы одной из матриц зафиксировать, то такое разложение единственно.*

Если зафиксированы элементы матрицы L, то матрицы L и U имеют вид:

$$L = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ l_{21} & 1 & 0 & \cdots & 0 \\ l_{31} & l_{32} & 1 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ l_{n1} & l_{n2} & l_{n3} & \cdots & 1 \end{pmatrix}, \; U = \begin{pmatrix} u_{11} & u_{12} & u_{13} & \cdots & u_{1n} \\ 0 & u_{22} & u_{23} & \cdots & u_{2n} \\ 0 & 0 & u_{33} & \cdots & u_{3n} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & 0 & u_{nn} \end{pmatrix},$$

где элементы вычисляются по формулам:

$$u_{ij} = a_{ij} - \sum_{k=1}^{i-1} l_{ik}u_{kj} \; (i \leqslant j), \quad l_{ij} = \frac{a_{ij} - \sum\limits_{k=1}^{j-1} l_{ik}u_{kj}}{u_{jj}} \; (i > j).$$

Заметим, что в этих формулах не будет деления на нуль.

Пользуясь этой теоремой, решение системы $Ax = b$ упрощается. Имеем представление $LUx = b$. Обозначив $Ux = y$, получим $Ly = b$. Решение СЛАУ сводится к последовательному решению двух систем с треугольными матрицами. Поэтому их решения выписываются явно. Сперва находим решение системы $Ly = b$:

$$y_k = b_k - \sum_{j=1}^{k-1} l_{kj}y_j, \; k = 1, 2, \ldots, n.$$

Если известны y_i, можно также легко найти x_i, как решение системы $Ux = y$:

$$x_k = \frac{y_k - \sum\limits_{j=k+1}^{n} u_{kj}x_j}{u_{kk}}, \; k = n, n-1, \ldots, 1.$$

Замечание. LU–метод решения СЛАУ — это тот же метод Гаусса, записанный в матричном представлении. Матрица U и вектор y есть не что иное, как полученные после прямого хода метода Гаусса треугольная матрица и вектор правой части.

Проверка условия, что все главные миноры отличны от нуля — довольно трудоемкий процесс, поэтому проверку можно выполнить по ходу вычислений. Если окажется, что на каком-то шаге $u_{ii} = 0$, то это будет означать, что не все главные миноры исходной матрицы были отличны от нуля и LU–метод здесь не работает.

Если найдены матрицы L и U, то легко вычислить определитель исходной матрицы $\det A = \det L \cdot \det U = \prod\limits_{i=1}^{n} u_{ii}$.

1.3. Метод квадратных корней (схема Холецкого)

Метод применяется в случае, когда матрица A **симметричная**, то есть $A = A^T$. Представим матрицу A в виде $A = S^T \cdot D \cdot S$, где D — диагональная матрица, причем элементы диагонали $d_i = \pm 1$, S — верхняя треугольная матрица, S^T — транспонированная к S матрица. Общий вид формул для нахождения элементов матриц D и S при $i = 1, 2, \ldots, n$

$$d_i = sign\left(a_{ii} - \sum_{p=1}^{i-1} s_{pi}^2 d_p\right), \quad s_{ii} = \sqrt{\left|a_{ii} - \sum_{p=1}^{i-1} s_{pi}^2 d_p\right|},$$

$$s_{ij} = \frac{a_{ij} - \sum\limits_{p=1}^{i-1} s_{pi} d_p s_{pj}}{d_i s_{ii}}, \; j = i+1, i+2, \ldots, n.$$

После нахождения элементов D и S решение исходной системы сводится, как и в LU–методе к решению двух систем с треугольными матрицами: $S^T D y = b$ и $Sx = y$. Решения обеих систем выписываются по явным формулам, так как матрица $S^T D$ — нижняя треугольная, а матрица S — верхняя треугольная.

Замечание. Равенство $A = S^T D S$ содержит всего $\frac{1}{2}n(n+1)$ уравнений для определения $\frac{1}{2}n(n+1) + n$ неизвестных, то есть на самом

деле n неизвестных d_i $(i = 1, 2, \ldots, n)$, как бы лишние. Они здесь введены искусственно, мы их выбираем сами, полагая $d_i = \pm 1$, чтобы можно было извлечь квадратные корни во всех формулах, где вычисляются диагональные элементы матрицы S. При таком выборе d_i всегда работаем только с действительными числами.

Можно рассмотреть и **другой метод** квадратных корней, когда ищут матрицу S такую, что $A = S^T S$. При этом формулы те же самые, только без матрицы D, поэтому подкоренные выражения будут без модуля. А значит, могут получиться комплексные числа. Однако если исходная матрица симметрична и правая часть системы действительна, а вычисления провести до конца, то в результате получится действительное решение даже если по ходу пришлось иметь дело с комплексными числами.

1.4. Метод ортогонализации

Для данного метода строится последовательность ортогональных векторов и на каком-то шаге очередной вектор дает решение системы (1). Обозначим $a_{i,n+1} = -b_i$, $i = 1, 2, \ldots, n$ и запишем исходную систему в развернутом виде:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n + a_{1,n+1} = 0, \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n + a_{2,n+1} = 0, \\ \ldots \\ a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n + a_{n,n+1} = 0. \end{cases} \tag{2}$$

Пусть $a_i = (a_{i1}, a_{i2}, \ldots, a_{in}, a_{i,n+1})$ — i-ая строка расширенной матрицы, $y = (x_1, x_2, \ldots, x_n, 1)$ — вектор неизвестных. Тогда система (2) примет эквивалентный вид, записанный через скалярные произведения:

$$(a_i, y) = 0, \ i = 1, 2, \ldots, n. \tag{3}$$

Запись системы в виде (3) означает, что если мы найдем $(n+1)$-мерный вектор y, последним элементом которого является 1, и он будет ортогонален всем векторам a_i, то первые n элементов этого вектора дадут решение исходной системы. Если $\det A \neq 0$, то используют *метод ортогонализации Грамма–Шмидта.*

Добавим к n линейно независимым векторам $\{a_1, a_2, \ldots, a_n\}$ еще один $(n+1)$-мерный вектор $a_{n+1} = (0, \ldots, 0, 1)$. Тогда систе-

ма $a_1,\ a_2, \ldots,\ a_{n+1}$ линейно независима. Будем строить вектора $u_1, v_1, u_2, v_2, \ldots, u_{n+1}$ по следующим правилам.

Берем первый вектор и нормализуем его: $u_1 = a_1,\ v_1 = \dfrac{u_1}{\|u_1\|}$, где $\|u_1\| = \sqrt{(u_1, u_1)}$. Далее u_2 строим как линейную комбинацию вектора a_2 и уже найденного v_1: $u_2 = a_2 - \gamma_{21} v_1$. Константу γ_{21} выбираем так, чтобы полученный вектор был ортогонален v_1, то есть $(u_2, v_1) = 0$. Откуда $\gamma_{21} = (a_2, v_1)$ и вектор $u_2 = a_2 - \gamma_{21} v_1$ определяется однозначно. Положим $v_2 = \dfrac{u_2}{\|u_2\|}$. Заметим, что v_2 ортогонален v_1. Аналогичным образом продолжая процесс, имеем при $j = 2, 3, \ldots, n+1$:

$$u_j = a_j - \sum_{i=1}^{j-1} \gamma_{ji} v_i,\ v_j = \frac{u_j}{\|u_j\|},\ \text{где } \gamma_{ji} = (a_j, v_i),\ i = 1, 2, \ldots, j-1.$$

Последний вектор $u_{n+1} = (c_1, c_2, \ldots, c_n, c_{n+1})$ ортогонален всем векторам a_i по построению. Если нормировать его последнюю компоненту, то первые n элементов полученного вектора дадут искомое решение $y = \left(\dfrac{c_1}{c_{n+1}}, \ldots, \dfrac{c_n}{c_{n+1}}, 1 \right)$.

1.5. Метод прогонки

Метод прогонки используется для решения систем линейных алгебраических уравнений (1) с трехдиагональной матрицей, которые в общем случае имеют вид

$$b_i x_{i-1} + c_i x_i + d_i x_{i+1} = r_i,\ i = 1, 2, \ldots, n, \tag{4}$$

где $b_1 = 0$, $d_n = 0$, или в векторно–матричном представлении

$$\begin{pmatrix} c_1 & d_1 & 0 & 0 & \cdots & 0 & 0 & 0 \\ b_2 & c_2 & d_2 & 0 & \cdots & 0 & 0 & 0 \\ 0 & b_3 & c_3 & d_3 & \cdots & 0 & 0 & 0 \\ . & . & . & . & \cdots & . & . & . \\ 0 & 0 & 0 & 0 & \cdots & b_{n-1} & c_{n-1} & d_{n-1} \\ 0 & 0 & 0 & 0 & \cdots & 0 & b_n & c_n \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \cdots \\ x_{n-1} \\ x_n \end{pmatrix} = \begin{pmatrix} r_1 \\ r_2 \\ r_3 \\ \cdots \\ r_{n-1} \\ r_n \end{pmatrix}.$$

Решение системы уравнений вида (4) методом прогонки состоит из двух этапов. *Прямая прогонка* — это нахождение прогоночных коэффициентов δ_i и λ_i при $i = 1, 2, \ldots, n$ по формулам

$$\delta_i = -\frac{d_i}{c_i + b_i\delta_{i-1}},\ \lambda_i = \frac{r_i - b_i\lambda_{i-1}}{c_i + b_i\delta_{i-1}}. \tag{5}$$

Так как $b_1 = 0$, то процесс вычисления δ_i и λ_i можно начать со значений $\delta_1 = -\dfrac{d_1}{c_1}$, $\lambda_1 = \dfrac{r_1}{c_1}$ и продолжить далее по формулам (5) последовательно при $i = 2, 3, \ldots, n$, причем при $i = n$ в силу $d_n = 0$ получим $\delta_n = 0$. *Обратная прогонка* — это получение неизвестных x_i при $i = n, n-1, \ldots, 1$ по формуле

$$x_i = \delta_i x_{i+1} + \lambda_i. \tag{6}$$

Тогда полагая в равенстве (6) $i = n$, будем иметь

$$x_n = \lambda_n = \frac{r_n - b_n\lambda_{n-1}}{c_n + b_n\delta_{n-1}},$$

где λ_{n-1} и δ_{n-1} — уже известные с предыдущего шага числа. Далее по формуле (6) последовательно находятся x_i при $i = n-1, n-2, \ldots, 1$ соответственно.

Для успешного применения метода прогонки нужно, чтобы в процессе вычислений не возникало ситуаций с делением на нуль, а при больших размерностях систем не было быстрого роста погрешности округлений.

Определение. Прогонка называется *корректной*, если знаменатели прогоночных коэффициентов (5) не обращаются в нуль, и *устойчивой*, если $|\delta_i| < 1$ при всех $i = 1, 2, \ldots, n$.

Приведем простые достаточные условия корректности и устойчивости прогонки, которые во многих приложениях метода выполняются автоматически. Эти условия гарантируют существование и единственность решения системы (4), а также возможность нахождения этого решения методом прогонки.

Теорема. *Пусть коэффициенты b_i и d_i при $i = 2, 3, \ldots, n-1$ уравнения (4) отличны от нуля, и пусть $|c_i| > |b_i| + |d_i|$, $i = 1, 2, \ldots, n$. Тогда прогонка по формулам (5), (6) корректна и устойчива.*

Замечание. Условия $|c_i| > |b_i| + |d_i|$ при $i = 1, 2, \ldots, n$ означают, что матрица системы имеет доминирующую главную диагональ.

1.6. Метод простых итераций

Рассмотрим систему (1), где $\det A \neq 0$. Пусть эту систему каким-то образом удалось записать в эквивалентном виде

$$x = Bx + c, \tag{7}$$

где B — матрица, c — вектор. Форма записи исходной системы (1) в виде (7) удобна тем, что она позволяет для ее решения строить итерационный процесс

$$x^{(n+1)} = Bx^{(n)} + c, \tag{8}$$

и получить, начиная с некоторого $x^{(0)}$, последовательность $x^{(1)}$, $x^{(2)}$, $\ldots$, $x^{(k)}$, $\ldots$, которая при определенных условиях сходится к решению (7).

Теорема 1 (Достаточное условие сходимости метода простых итераций). *Если* $\|B\| < 1$, *то итерационный процесс* (8) *для любого начального приближения* $x^{(0)}$ *сходится со скоростью геометрической прогрессии к единственному решению системы* (7).

Теорема 2 (Необходимое и достаточное условие сходимости метода простых итераций). *Метод простых итераций* (8) *сходится для любого начального приближения* $x^{(0)}$ *к единственному решению системы* (7) *тогда, и только тогда, когда* $|\lambda_B| < 1$.

Теорема 3. *Пусть* $\|B\| \leqslant q < 1$. *Тогда итерационный процесс* (8) *сходится к* x_* *при любом начальном приближении* $x^{(0)}$ *и имеют место оценки:*

$$\left\|x_* - x^{(k)}\right\| \leqslant \frac{q}{1-q}\left\|x^{(k)} - x^{(k-1)}\right\| \text{ — } \textbf{апостериори},$$

$$\left\|x_* - x^{(k)}\right\| \leqslant \frac{q^k}{1-q}\left\|x^{(1)} - x^{(0)}\right\| \text{ — } \textbf{априори}.$$

Если получена априорная оценка для какого-то метода, то она позволяет, выполнив всего один шаг метода, заранее узнать, сколько шагов достаточно будет сделать, чтобы получить решение с наперед заданной точностью. К сожалению, априорные оценки удается получить не для всякого итерационного метода. Апостериорной оценкой удобно пользоваться для оценки погрешности только что найденного k-го приближения и остановить итерационный процесс, как только будет выполняться условие

$$\left\|x^{(k)} - x^{(k-1)}\right\| \leqslant \frac{1-q}{q}\varepsilon. \tag{9}$$

При выполнении неравенства (9) получим, что $\left\|x_* - x^{(k)}\right\| \leqslant \varepsilon$, то есть найденное k-е приближение дает решение с точностью ε. В частности, получаем важный вывод: неравенство $\left\|x^{(k)} - x^{(k-1)}\right\| \leqslant \varepsilon$, которое часто используют для проверки окончания итерационного процесса, будет гарантией того, что и $\left\|x_* - x^{(k)}\right\| \leqslant \varepsilon$, только в том случае, если $q \leqslant \dfrac{1}{2}$.

Замечание. Не всякая эквивалентная система вида (7) дает сходящийся итерационный процесс для системы (1). Систему $Ax = b$ всегда можно записать в эквивалентном виде

$$x = x - S(Ax - b) = (E - SA)\,x + Sb,$$

то есть $B = E - SA$ и $c = Sb$, где S — любая невырожденная матрица, которую подбирают таким образом, чтобы выполнялись условия теорем сходимости.

1.7. Метод Якоби

Матрицу A представим в виде суммы матриц $A = L + D + R$, где

$$L = \begin{pmatrix} 0 & 0 & \dots & 0 \\ a_{21} & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots \\ a_{n1} & \dots & a_{n\,n-1} & 0 \end{pmatrix}; \quad D = \begin{pmatrix} a_{11} & 0 & \dots & 0 & 0 \\ 0 & a_{22} & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & \dots & a_{n\,n} \end{pmatrix};$$

$$R = \begin{pmatrix} 0 & a_{12} & \dots & a_{1n-1} & a_{1n} \\ 0 & 0 & \dots & a_{2n-1} & a_{2n} \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 0 & a_{n-1\,n} \\ 0 & 0 & \dots & 0 & 0 \end{pmatrix}.$$

Тогда $Dx = -(L + R)x + b$ или, если $a_{ii} \neq 0$, $\forall i = 1, 2, \dots, n$, то $x = -D^{-1}(L + R)x + D^{-1}b$. Обозначая $B = -D^{-1}(L + R)$ и $c = D^{-1}b$, получаем из системы (1) эквивалентную систему (7). Запишем полученный итерационный процесс (8) в развернутом виде

$$
\begin{cases}
x_1^{(k+1)} = -\left(a_{12}x_2^{(k)} + a_{13}x_3^{(k)} + \ldots + a_{1n}x_n^{(k)} - b_1\right)/a_{11}, \\
x_2^{(k+1)} = -\left(a_{21}x_1^{(k)} + a_{23}x_3^{(k)} + \ldots + a_{2n}x_n^{(k)} - b_2\right)/a_{22}, \\
\ldots \\
x_i^{(k+1)} = -\left(a_{i1}x_1^{(k)} + a_{i2}x_2^{(k)} + \ldots + a_{i\,i-1}x_{i-1}^{(k)} + \right. \\
\qquad \left. + a_{i\,i+1}x_{i+1}^{(k)} + \ldots + a_{in}x_n^{(k)} - b_i\right)/a_{i\,i}, \\
\ldots \\
x_n^{(k+1)} = -\left(a_{n1}x_1^{(k)} + \ldots + a_{n\,n-1}x_{n-1}^{(k)} - b_n\right)/a_{nn}.
\end{cases} \tag{10}
$$

Итерационный процесс, определяемый формулой (10) называется *методом Якоби* для решения задачи (1). Заметим, что метод Якоби — это частный случай метода простых итераций.

Теорема 1 (достаточное условие сходимости метода Якоби). *В случае диагонального преобладания элементов матрицы A итерационный процесс (10) сходится к решению задачи (1) при любом начальном приближении $x^{(0)}$.*

Теорема 2 (Необходимое и достаточное условие сходимости). *Метод Якоби сходится к решению задачи (1) тогда, и только тогда, когда $|\lambda_B| < 1$.*

1.8. Метод Зейделя

Метод простых итераций определяется формулой (8), где для получения всех компонент $(k+1)$-го приближения используются компоненты k-го приближения. Идея метода Зейделя заключается в том, чтобы при вычислении i-й компоненты $x_i^{(k+1)}$ $(k+1)$-го приближения используются уже вычисленные элементы $x_1^{(k+1)}$, $x_2^{(k+1)}, \ldots, x_{i-1}^{(k+1)}$, а остальные элементы $x_{i+1}^{(k)}, \ldots, x_n^{(k)}$ берутся из предыдущего k-го приближения.

Формулы метода простых итераций позволяют записать систему в виде

$$\begin{cases} x_1^{(k+1)} = b_{11}x_1^{(k)} + b_{12}x_2^{(k)} + \cdots + b_{1n}x_n^{(k)} + c_1, \\ x_2^{(k+1)} = b_{21}x_1^{(k+1)} + b_{22}x_2^{(k)} + \cdots + b_{2n}x_n^{(k)} + c_2, \\ \cdots \\ x_i^{(k+1)} = b_{i1}x_1^{(k+1)} + b_{i2}x_2^{(k+1)} + \cdots + b_{ii-1}x_{i-1}^{(k+1)} + \\ \qquad\qquad + b_{ii}x_i^{(k)} + \cdots + b_{in}x_n^{(k)} + c_i, \\ \cdots \\ x_n^{(k+1)} = b_{n1}x_1^{(k+1)} + b_{n2}x_2^{(k+1)} + \cdots + b_{nn-1}x_{n-1}^{(k+1)} + b_{nn}x_n^{(k)} + c_n. \end{cases}$$

Формулы метода Якоби, представление $A = L + D + R$ и условие $\det(L + D) \neq 0$ приводят к итерационному процессу метода Зейделя в виде

$$x^{(k+1)} = Bx^{(k)} + c, \text{ где } B = -(L + D)^{-1}R, \; c = (L + D)^{-1}b. \quad (11)$$

Теорема 1 (Достаточное условие сходимости). *Метод Зейделя сходится для любой симметричной и положительно определенной матрицы A.*

Теорема 2 (Необходимое и достаточное условие сходимости). *Метод Зейделя* (11) *сходится тогда, и только тогда, когда $|\lambda_B| < 1$.*

1.9. Вычисление определителя и обратной матрицы

Для вычисления определителя исходной матрицы можно воспользоваться свойством: при перестановке строк определитель матрицы меняет знак, а при других элементарных преобразованиях, которые применялись на этапе прямого хода метода Гаусса, определитель не меняется. Поэтому верна формула: $\det A = (-1)^s \det T$, где T — верхняя треугольная матрица после прямого хода, s — число перестановок строк в алгоритме. Определитель T равен произведению элементов главной диагонали (ведущих элементов).

Для определения элементов обратной матрицы достаточно решить матричное уравнение: $AX = E$, где A — данная матрица, E — единичная матрица. Тогда $A^{-1} = X$. Если воспользоваться определением произведения матриц, то для элементов j-го столбца обратной матрицы получаем систему уравнений с матрицей A и правой частью, равной j-му столбцу единичной матрицы: $Ax_{(j)} = e_j$, где e_j — j-й столбец матрицы E, а $x_{(j)}$ — j-й столбец обратной матрицы A^{-1}.

Если у нас есть алгоритм решения систем уравнений, то для определения элементов обратной матрицы достаточно решить n систем, причем все эти системы имеют одинаковую матрицу A и отличаются лишь правой частью.

1.10. Решение СЛАУ в комплексном пространстве

Пусть дана система $Ax = b$, где A и b имеют комплексные элементы. Тогда решение такой системы можно свести к решению другой системы в действительном пространстве. Пусть $A = A_1 + iA_2$, $b = b_1 + ib_2$, где элементы A_j и b_j $(j = 1, 2)$ — действительные числа. Решение системы представим в виде $x = x_1 + ix_2$. Тогда получим систему $(A_1 + iA_2)(x_1 + ix_2) = b_1 + ib_2$. Приравнивая действительную и мнимую части и обозначая $y = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$, $C = \begin{pmatrix} A_1 & -A_2 \\ A_2 & A_1 \end{pmatrix}$, $d = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$, исходную систему запишем как $Cy = d$.

Итак, исходная система n-го порядка $Ax = b$ на множестве комплексных чисел сводится к системе $Cy = d$ с матрицей C порядка $2n \times 2n$ и вектором d длиной $2n$ на множестве действительных чисел. Решив систему $Cy = d$, найдем вектор y, причем его первые n компонент дают действительную часть, последние n компонент — мнимую часть искомого вектора x.

2. Собственные значения и собственные векторы

Вычисление собственных значений и собственных векторов матрицы является одной из самых сложных задач линейной алгебры. Для решения проблемы собственных значений в случае матриц больших порядков используются только итерационные методы, в которых собственные значения определяются непосредственно, минуя характеристический многочлен. Различают две задачи, связанные с собственными векторами и значениями: *полная проблема* собственных значений и собственных векторов, когда требуется определить *все* собственные значения и *все* собственные векторы; *частичная проблема* собственных значений и собственных векторов, когда нужно найти одно или несколько собственных значений и, возможно, соответствующих собственных векторов.

2.1. Степенной метод

Пусть вещественная квадратная матрица A порядка $n \times n$ является матрицей простой структуры, то есть имеет ровно n линейно независимых собственных векторов: $x_1, x_2, \ldots, x_n$. Пусть нумерация этих векторов отвечает упорядочению соответствующих им собственных чисел по убыванию модулей (где первое из неравенств — строгое): $|\lambda_1| > |\lambda_2| \geqslant \ldots \geqslant |\lambda_n|$.

Алгоритм нахождения максимального по модулю собственного значения для данной матрицы A. Берется произвольный вектор $y^{(0)} \neq 0$, строится последовательность векторов по правилу $y^{(k)} = Ay^{(k-1)}$, $k = 1, 2, \ldots$, и параллельно рассматриваются последовательности отношений соответствующих компонент векторов k-й и $(k-1)$-й итераций $\dfrac{y_i^{(k)}}{y_i^{(k-1)}}$, и находится предел при $k \to \infty$. Если предел существует, то он равен λ_1, если он не существует, то рекомендуется взять другой начальный вектор.

Замечание. Более логичным для расчетов является среднее арифметическое этих компонент $\dfrac{1}{n} \sum\limits_{j=1}^{n} \dfrac{y_j^{(k)}}{y_j^{(k-1)}} \to \lambda_1, \ k \to \infty$.

Алгоритм нахождения минимального собственного значения для знакоопределенной матрицы. Пусть дана матрица A и ее максимальное по модулю собственное значение $|\lambda_1| = \max\limits_i |\lambda_i|$. Для нахождения λ_n нужно: 1) найти λ_1 — максимальное собственное значение исходной матрицы A; 2) найти $\overline{\lambda}$ — максимальное собственное значение матрицы $(A - \lambda_1 E)$; 3) положить $\lambda_n = \lambda_1 + \overline{\lambda}$. Это и есть минимальное собственное значение A. Описанный метод выгодно использовать, когда требуется найти и минимальное, и максимальное по модулю собственные значения одновременно. Если требуется найти только минимальное по модулю собственное значение, то можно обойтись без нахождения λ_1: 1) находим μ_1 — максимальное собственное значение матрицы A^{-1}; 2) полагаем $\lambda_n = \dfrac{1}{\mu_1}$ минимальное собственное значение матрицы A.

Степенной метод позволяет находить и собственные вектора, со-

ответствующие найденным собственным значениям:

$$y^{(k)} = \lambda_1^k \left(c_1 x_1 + c_2 \left(\frac{\lambda_2}{\lambda_1} \right)^k x_2 + \ldots + c_n \left(\frac{\lambda_n}{\lambda_1} \right)^k x_n \right) \to c_1 \lambda_1^k x_1$$

при $k \to \infty$, то есть сам вектор $y^{(k)}$ стремится к собственному вектору x_1 с некоторым сомножителем. Но так как собственный вектор определяется с точностью до сомножителя, то в пределе $y^{(k)}$ дает собственный вектор. За собственный вектор $x_1 \approx y^{(k)}$.

Недостатком степенного метода является то, что если $|\lambda_1| > 1$, то компоненты вектора $y^{(k)}$ быстро растут и получается часто переполнение памяти машины. Чтобы устранить этот недостаток, используют нормировку получившихся векторов. Для вектора $y^{(0)}$ найдем $z^{(0)} = \dfrac{y^{(0)}}{\|y^{(0)}\|}$, а затем вычислим $y^{(1)} = Az^{(0)}$, $z^{(1)} = \dfrac{y^{(1)}}{\|y^{(1)}\|}$, $y^{(2)} = Az^{(1)}, \ldots, \dfrac{y_j^{(k+1)}}{z_j^{(k)}} \to \lambda_1$, $z^{(k)} \to x_1$ при $k \to \infty$.

Алгоритм нахождения второго по модулю собственного значения: 1) берем произвольный вектор $y^{(0)}$ и построим последовательность $y^{(k)} = Ay^{(k-1)}, k = 1, 2, \ldots$; 2) находим степенным методом λ_1 — максимальное собственное значение A; 3) составим отношение

$$\frac{y_j^{(k+1)} - \lambda_1 y_j^{(k)}}{y_j^{(k)} - \lambda_1 y_j^{(k-1)}} \to \lambda_2 \text{ при } k \to \infty.$$

2.2. Метод скалярных произведений (SP-метод)

Пусть A — симметричная, положительно определенная матрица. Вычислим векторы $y^{(0)}$, $y^{(1)} = Ay^{(0)}, \ldots, y^{(k+1)} = Ay^{(k)}, \ldots$ Рассмотрим отношения скалярных произведений:

$$\frac{\left(y^{(k)}, y^{(k)}\right)}{\left(y^{(k)}, y^{(k-1)}\right)} = \frac{\sum\limits_{i=1}^{n} c_i^2 \lambda_i^{2k} x_i^2}{\sum\limits_{i=1}^{n} c_i^2 \lambda_i^{2k-1} x_i^2} \to \lambda_1, \ k \to \infty.$$

При таком подходе, так же как и в степенном методе, можно получить большие числа, поэтому используют нормировку. Возьмем

$y^{(0)},\ z^{(0)} = \dfrac{y^{(0)}}{\|y^{(0)}\|},\ y^{(1)} = Az^{(0)},\ z^{(1)} = \dfrac{y^{(1)}}{\|y^{(1)}\|}, \ldots,\ y^{(k)} = Az^{(k-1)},$ $z^{(k)} = \dfrac{y^{(k)}}{\|y^{(k)}\|}, \ldots$ Тогда $\dfrac{\left(y^{(k)}, y^{(k)}\right)}{\left(y^{(k)}, z^{(k-1)}\right)} \to \lambda_1,\ k \to \infty$. Получаем также, что $z^{(k)} = \dfrac{y^{(k)}}{\|y^{(k)}\|} \to x_1$ при $k \to \infty$, то есть стремится к собственному вектору.

2.3. Метод вращений Якоби

Метод вращений Якоби решает полную проблему нахождения собственных значений и собственных векторов. Рассматриваются только симметричные вещественные матрицы. Всякая симметричная матрица ортогонально подобна диагональной. Поэтому если найдется преобразование подобия, которое приведет матрицу к диагональному виду, то можно найти все собственные значения и собственные векторы матрицы. Наша задача — реализовать, хотя бы приближенно, равенство $\Lambda = diag(\lambda_1, \lambda_2, \ldots, \lambda_n) = X^T A X$, которое позволило бы найти все собственные значения матрицы A (элементы диагональной матрицы Λ) и все соответствующие им собственные векторы матрицы A (столбцы матрица X).

Матрицы преобразования плоских вращений имеют вид

$$Q_{ij} = \begin{pmatrix} 1 & : & \cdots & : & 0 \\ \cdot & c & \cdots & -s & \cdot \\ \cdot & \cdot & \cdots & \cdot & \cdot \\ \cdot & s & \cdots & c & \cdot \\ 0 & : & \cdots & : & 1 \end{pmatrix} \tag{12}$$

и получаются из единичной матрицы заменой двух единиц и двух нулей на пересечениях i-х и j-х строк и столбцов числами c и $\pm s$ такими, что $c = \cos\alpha$ и $s = \sin\alpha$, где угол α определяется как $\operatorname{tg} 2\alpha = \dfrac{2a_{ij}}{a_{ii} - a_{jj}}$. Умножение любой матрицы на матрицу Q_{ij} изменяет у нее только две строки и два столбца по формулам поворота на угол α в плоскости, определяемой парой индексов i и j, поэтому данная матрица называется *матрицей вращений*. Матрица Q_{ij} ортогональна при любых $i, j = 1, 2, \ldots, n$, и, значит, матрица $B = Q_{ij}^T \cdot A \cdot Q_{ij}$ подобна A, т. е. имеет тот же набор собственных значений, что и A.

Идея метода Якоби заключается в том, чтобы построить последовательность подобных матриц $B_0 = A, B_1, \ldots, B_k, \ldots$ таких, что каждое преобразование подобия $B_k = Q_{ij}^T B_{k-1} Q_{ij}$ обнуляет максимальный по модулю внедиагональный элемент матрицы B_{k-1}. С помощью преобразований подобия

$$B_k = Q_{ij}^T \cdot B_{k-1} \cdot Q_{ij}, \tag{13}$$

при котором на k-м шаге обнуляется максимальный по модулю элемент матрицы B_{k-1}, получаем последовательность матриц, подобных исходной матрице A. На каждом шаге для имеющегося B_{k-1} определяем максимальный по модулю внедиагональный элемент (*обреченный элемент*), по нему вычисляем числа c и s, далее, используя матрицу (12) находим по формуле (13) новую матрицу B_k и т. д. Заметим, что на каждом шаге таких преобразований пересчитываются только две строки (столбца) предыдущей матрицы. При умножении некоторой матрицы B справа на матрицу вида (12) изменяются только два столбца, а именно — столбцы с номерами i и j матрицы B. Другими словами, если, например, надо вычислить $X = BQ$, то $x_{li} = c \cdot b_{li} + s \cdot b_{lj}$, $x_{lj} = -s \cdot b_{li} + c \cdot b_{lj}$, где $l = 1, 2, \ldots, n$, а остальные элементы матрицы X такие же, что и у матрицы B.

Теорема. *Если на каждом шаге в качестве обреченного элемента брать максимальный по модулю внедиагональный элемент, то последовательность матриц B_k сходится к диагональной матрице.*

Собственными векторами матрицы A будут столбцы матрицы $X = Q_{i_0 j_0} Q_{i_1 j_1} \cdots Q_{i_{k-1} j_{k-1}}$. Здесь k — количество итераций, которые потребовались для получения из A диагональной матрицы Λ с заданной точностью.

2.4. LU-алгоритм для несимметричных задач

Чаще (по крайней мере, в несимметричном случае) алгоритмы приближенного решения полных проблем собственных значений основываются на приведении данных матриц к подобным им матрицам не диагонального, а треугольного вида. Наиболее простой из таких алгоритмов вычисления собственных чисел опирается на LU–разложение матриц.

Пусть данная $n \times n$ матрица A представлена в виде $A = LU$, где L и U — соответственно нижняя и верхняя треугольные матрицы

(см. п. 1.2). LU–алгоритм определяется фактически двумя формулами:

$$A_k = L_k U_k, \;\; A_{k+1} = U_k L_k, \tag{14}$$

где $A_0 := A$, $k = 0, 1, \ldots$, причем первая из этих формул означает процедуру треугольной факторизации матрицы A_k на k-ом шаге, а вторая — простое умножение верхней треугольной матрицы на нижнюю.

При ряде ограничений на матрицу A (простейшим из которых является, в частности, требование, чтобы все ее собственные числа были различны по модулю) итерационный процесс (14) осуществим, и формируемая им последовательность $\{A_k\}$ сходится к треугольной матрице вида

$$\begin{pmatrix} \lambda_1 & * & \cdots & * \\ 0 & \lambda_2 & \cdots & * \\ \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & \lambda_n \end{pmatrix} \quad \text{или} \quad \begin{pmatrix} \lambda_1 & 0 & \cdots & 0 \\ * & \lambda_2 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots \\ * & * & \cdots & \lambda_n \end{pmatrix}$$

в зависимости от того, фиксируется единичная диагональ при LU–факторизации у матрицы L и U соответственно, где $*$ обозначены некоторые числа.

Одним из серьезных факторов, ограничивающих сферу применения LU–алгоритмов, является их недостаточно хорошая численная устойчивость (улучшение этого параметра путем перестановок строк и столбцов сильно отражается на экономичности метода). Этот фактор может играть особенно существенную роль на фоне возможностей неустойчивой самой несимметричной проблемы собственных значений.

3. Методы решения нелинейных уравнений и систем

Уравнение вида

$$f(x) = 0 \tag{15}$$

называется *нелинейным*, если функция f — нелинейная. *Решить уравнение* — это значит найти такое x, при котором уравнение превращается в тождество, или доказать, что таких x не существует. В общем случае уравнение может иметь $0, 1, 2, \ldots$ корней.

Будем рассматривать как уравнения относительно одной переменной, так и системы из n уравнений с n неизвестными:

$$\begin{cases} f_1(x_1, x_2, \ldots, x_n) = 0, \\ f_2(x_1, x_2, \ldots, x_n) = 0, \\ \ldots \\ f_n(x_1, x_2, \ldots, x_n) = 0. \end{cases} \tag{16}$$

Полагая $x = (x_1, x_2, \ldots, x_n)^T$, $f(x) = (f_1(x), f_2(x), \ldots, f_n(x))^T$, система (16) примет вид (15).

Как известно, только некоторые нелинейные уравнения могут быть решены точно (за конечное число действий). Э. Галуа показал: если используются обычные арифметические операции и операция извлечения корня, то не существует формул для нулей произвольного многочлена степени 5 и выше. Поэтому решить произвольные нелинейные уравнения (даже с одной переменной) за конечное число операций не представляется возможным. Таким образом, все методы решения нелинейных уравнений и систем, которые будут рассмотрены, являются итерационными. В этом случае приходится расширить понятие решения.

Пусть x_* удовлетворяет уравнению (15), то есть $f(x_*) = 0$. Будем говорить, что ξ — *обобщенное решение уравнения* (15), если $\forall \varepsilon > 0$ выполняется хотя бы одно из неравенств $|f(\xi)| < \varepsilon$ или $|\xi - x_*| < \varepsilon$. Так как арифметические операции на компьютере выполняются с ошибками округления, то это определение вполне логично. Дело в том, что если строго придерживаться понятия решения, то в машинной арифметике, где имеется лишь конечное число точек, может не найтись такого числа с плавающей точкой (запятой) x, что $f(x) = 0$, хотя x является решением в классическом понимании.

3.1. Локализация корней

В случае, когда на отрезке несколько корней (корни не отделены), можно «удалить» уже найденный корень, чтобы, используя тот же алгоритм еще раз, найти другой корень. Если x_1 — простой корень уравнения (15) и $f(x)$ удовлетворяет условию Липшица на $[a, b]$, то вспомогательная функция $g_1(x) = \dfrac{f(x)}{x - x_1}$ непрерывна, причем все нули функции $f(x)$ и $g_1(x)$ совпадают, за исключени-

ем корня x_1. Если x_1 — кратный корень, то он будет нулем функции $g_1(x)$ кратности, на единицу меньше, остальные нули у обеих функций по-прежнему будут совпадать. Поэтому найденный корень можно удалить, если далее решать уравнение $g_1(x) = 0$. Если x_2 — корень уравнения $g_1(x) = 0$, то далее решаем уравнение $g_2(x) = \dfrac{g_1(x)}{x - x_2} = \dfrac{f(x)}{(x - x_1)(x - x_2)} = 0$, и т. д.

3.2. Метод половинного деления

Наиболее примитивным, и в то же время и надежным алгоритмом определения вещественного корня является метод половинного деления (или дихотомия, бисекция, метод взятия в вилку). Пусть известно, что непрерывная на отрезке $[a, b]$ функция $f(x)$ в точках a и b имеет разные знаки, то есть $f(a) \cdot f(b) < 0$. Тогда, как известно из курса математического анализа, на этом отрезке имеется хотя бы один корень.

Идея метода: отрезок, где находится хотя бы один корень, делится на два равных отрезка и из них выбирается тот, на концах которого функция опять имеет разные знаки. Далее, с полученным отрезком опять проделываем ту же операцию. Так как длина отрезка каждый раз уменьшается в два раза, то после n шагов получим отрезок длины $\dfrac{b - a}{2^n}$, который содержит корень x_*. Понятно, что какова бы ни была длина исходного отрезка, для $\forall \varepsilon > 0$ через конечное число шагов получим отрезок длины меньше ε, который содержит корень. Следовательно, любая точка ξ этого последнего отрезка удовлетворяет неравенству $|\xi - x_*| < \varepsilon$, то есть является корнем уравнения с точностью $\varepsilon > 0$. К простому корню метод приводит для любой непрерывной, в том числе и недифференцируемой функции. К сожалению, метод имеет медленную скорость сходимости, но точность ответа всегда можно гарантировать.

3.3. Метод Ньютона–Рафсона (метод касательных)

Пусть функция $f(x)$ на отрезке $[a, b]$, содержащем корень ξ уравнения (15), дважды дифференцируема. Возьмем произвольную точ-

ку $x_0 \in [a, b]$ и воспользуемся итерационным процессом:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)},\ n = 0, 1, \ldots \tag{17}$$

Теорема 1. *Пусть на отрезке* $[a, b]$ *функция* $f(x)$ *имеет корень, причем* $f'(x)$ *и* $f''(x)$ *на этом отрезке сохраняют определенные знаки. Тогда для любого начального приближения* $x_0 \in [a, b]$ *такого, что* $f(x_0) \cdot f''(x_0) > 0$, *процесс Ньютона–Рафсона* (17) *сходится к единственному корню уравнения* (15).

Для оценки погрешности используют неравенство:

$$|\xi - x_n| \leqslant \frac{|f(x_n)|}{m},\ \text{где } m = \min_{[a,b]} |f'(x)|\,.$$

Из формулы Ньютона–Рафсона видно, что если производная вблизи корня становится малой, то поправки могут быть очень большими и очень трудно будет приблизиться к корню с такими большими шагами. Чем больше значение производной $f'(x)$ в окрестности данного корня, тем ближе можно к нему подойти.

Замечание 1. Если корень, к которому сходится метод Ньютона–Рафсона, простой, то последовательность сходится гораздо быстрее, чем в методе половинного деления или в методе простых итераций.

Замечание 2. В методе Ньютона–Рафсона на каждом шаге требуется вычисление значения производной. Но во многих прикладных задачах нет возможности аналитически задавать выражение для производной функции. В таких случаях может оказаться полезным *модифицированный метод Ньютона–Рафсона*

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_0)},\ n = 0, 1, \ldots. \tag{18}$$

Одно из важных достоинств метода Ньютона–Рафсона — это то, что он легко обобщается на системы нелинейных уравнений (16). Итерационный процесс в этом случае имеет вид

$$x^{(k+1)} = x^{(k)} - \left(I\left(x^{(k)}\right)\right)^{-1} f\left(x^{(k)}\right),\ k = 0, 1, \ldots, \tag{19}$$

где $x^{(k)} = \left(x_1^{(k)}, x_2^{(k)}, \ldots, x_n^{(k)}\right)$ — вектор решения, $I\left(x^{(k)}\right)$ — матрица

Якоби частных производных первого порядка в точке $x^{(k)}$ с элементами

$$\left(I\left(x^{(k)}\right)\right)_{ij} = \left(\frac{\partial f_i}{\partial x_j}\right)_{x=x^{(k)}}.$$

3.4. Метод секущих (хорд)

Пусть, как и в методе половинного деления, имеем отрезок $[a, b]$, где $f(a) \cdot f(b) < 0$. Для определенности будем полагать, что $f(a) < 0$, $f(b) > 0$ и $f''(x) > 0$ на всем отрезке. Тогда итерационный процесс для метода секущих имеет вид:

$$x_{n+1} = x_n - \frac{f(x_n)}{f(b) - f(x_n)}(b - x_n), \text{ где } x_0 = a. \tag{20}$$

Для данного случая задания итерационного процесса точка b является неподвижной точкой.

Случай $f''(x) < 0$ сводится к рассматриваемому, если уравнение записать в виде $-f(x) = 0$. Тогда итерационный процесс примет вид:

$$x_{n+1} = x_n - \frac{f(x_n)}{f(x_n) - f(a)}(x_n - a), \text{где } x_0 = b. \tag{21}$$

Здесь точка a является неподвижным концом отрезка $[a, b]$.

Теорема 2. *Пусть на отрезке $[a, b]$ уравнение (15) имеет единственный корень и $f''(x)$ сохраняет знак на $[a, b]$. Тогда метод секущих сходится.*

3.5. Комбинированный метод

Рассмотрим для определенности, как и выше, случай $f'(x) > 0$, $f''(x) > 0$, $f(a) < 0$, $f(b) > 0$. Возьмем в качестве начального приближения $x_0 = a$ и выполним один шаг метода секущих

$$x_1 = x_0 - \frac{f(x_0)}{f(b) - f(x_0)}(b - x_0).$$

Далее возьмем в качестве начального приближения $\bar{x}_0 = b$ и выполним один шаг метода Ньютона–Рафсона

$$\bar{x}_1 = \bar{x}_0 - \frac{f(\bar{x}_0)}{f'(\bar{x}_0)}.$$

Теперь в качестве нового отрезка возьмем $[x_1, \bar{x}_1]$, который целиком содержится в исходном отрезке $a < x_1 < \xi < \bar{x}_1 < b$. Таким образом, скомбинировав метод секущих и метод Ньютона–Рафсона, получим итерационный процесс:

$$x_0 = a,\ \bar{x}_0 = b,\ x_{n+1} = x_n - \frac{f(x_n)}{f(\bar{x}_n) - f(x_n)}(\bar{x}_n - x_n),$$
$$\bar{x}_{n+1} = \bar{x}_n - \frac{f(\bar{x}_n)}{f'(\bar{x}_n)},\ n = 0, 1, \ldots, \tag{22}$$

при котором на каждом шаге будем приближаться к корню слева и справа, то есть сужать исходный отрезок с двух сторон, а из неравенства $x_n < \xi < \bar{x}_n$ получаем возможность на каждом шаге оценить, насколько приблизились к точному корню уравнения. Очевидно, при выполнении условий теорем 1 и 2 сходимости итерационный процесс (22) сходится к корню уравнения. При этом если на каком-то шаге $\bar{x}_n - x_n < \varepsilon$, то в качестве приближенного решения уравнения можно взять любую точку $c \in [x_n, \bar{x}_n]$ и погрешность $|c - \xi| < \varepsilon$.

Замечание. Формулы (22) выведены для определенного случая знакопостоянства производных $f'(x) > 0$, $f''(x) > 0$. Однако по этим же формулам можно строить итерационный процесс и в остальных случаях знакопостоянства производных первого и второго порядков функции $f(x)$:
1) если $f'(x) < 0$, $f''(x) < 0$, то решаем уравнение $g(x) = 0$, где $g(x) = -f(x)$;
2) если $f'(x) < 0$, $f''(x) > 0$, то решаем уравнение $g(x) = 0$, где $g(x) = f(-x)$;
3) если $f'(x) > 0$, $f''(x) < 0$, то решаем уравнение $g(x) = 0$, где $g(x) = -f(-x)$.
Функция $g(x)$ во всех трех случаях обладает свойством $g'(x) > 0$, $g''(x) > 0$, поэтому можно воспользоваться формулами (22).

3.6. Метод простых итераций

Пусть с точностью ε необходимо найти корень уравнения (15), принадлежащий интервалу изоляции $[a, b]$. Функция $f(x)$ и ее первая производная непрерывны на этом отрезке. Для применения метода итераций (метода последовательных приближений) исходное урав-

нение $f(x) = 0$ должно быть приведено к виду

$$x = \varphi(x). \tag{23}$$

В качестве начального приближения x_0 выбираем любую точку интервала $[a, b]$. Далее итерационный процесс поиска корня строится по схеме:

$$x_{n+1} = \varphi(x_n),\ n = 0, 1, \ldots \tag{24}$$

Процесс поиска прекращается, как только выполняется условие $|x_{n+1} - x_n| < \varepsilon$ или число итераций превысит заданное число N.

При определенных условиях на функцию $\varphi(x)$ итерационная последовательность (24) сходится к корню уравнения (23).

Теорема 3. *Пусть ξ — корень уравнения (23) и пусть функция $\varphi(x)$ удовлетворяет в некоторой окрестности точки ξ условию Липшица с постоянной $q < 1$. Тогда при любом выборе начального приближения x_0 из этой окрестности можно построить итерационную последовательность x_n, которая сходится к единственному корню ξ, и имеют место оценки*

$$|x_n - \xi| \leqslant \frac{q^n |x_1 - x_0|}{1 - q} \quad \text{— } \textbf{априори};$$

$$|x_n - \xi| \leqslant \frac{q |x_n - x_{n-1}|}{1 - q} \quad \text{— } \textbf{апостериори}.$$

3.7. Метод спуска

Метод спуска применим к решению систем нелинейных уравнений. Ограничимся системой двух уравнений с двумя неизвестными:

$$\begin{cases} f(x, y) = 0, \\ g(x, y) = 0, \end{cases} \tag{25}$$

где функции f и g дважды непрерывно дифференцируемы по своим аргументам. Пусть

$$\Psi(x, y) = f^2(x, y) + g^2(x, y).$$

Так как $\Psi(x, y)$ неотрицательна по определению, то $\exists (x_*, y_*)$: $\Psi(x, y) \geqslant \Psi(x_*, y_*) \geqslant 0, \forall (x, y) \in \mathbb{R}^2$, т. е. (x_*, y_*) является точкой минимума функции $\Psi(x, y)$. Если каким-то образом мы найдем точку

минимума, в которой обращается в нуль функция $\Psi(x, y)$, то данная точка будет искомым решением системы (25).

Для нахождения этой точки будем использовать классический итерационный метод спуска: строим последовательность (x_k, y_k) приближений к (x_*, y_*) при $k \to \infty$ по рекурректным формулам:

$$\begin{pmatrix} x_{k+1} \\ y_{k+1} \end{pmatrix} = \begin{pmatrix} x_k \\ y_k \end{pmatrix} + \alpha_k \begin{pmatrix} p_k \\ q_k \end{pmatrix}, \quad k = 0, 1, 2, \ldots, \tag{26}$$

где вектор $\begin{pmatrix} p_k \\ q_k \end{pmatrix}$ определяет на плоскости направление движения от k-го приближения к следующему $(k+1)$-му, а величина α_k регулирует длину шага в этом направлении.

Направление выбирается таким образом, чтобы минимизируемая функция убывала как можно быстрее. Функция нескольких переменных растет наиболее быстро в направлении вектора градиента. Поэтому в качестве вектора направления возьмем антиградиент, то есть

$$\begin{pmatrix} p_k \\ q_k \end{pmatrix} = -\text{grad}\,(\Psi(x_k, y_k)) = -\begin{pmatrix} \Psi'_x(x_k, y_k) \\ \Psi'_y(x_k, y_k) \end{pmatrix}. \tag{27}$$

Подставим выбранное направление в формулу (26):

$$\begin{pmatrix} x_{k+1} \\ y_{k+1} \end{pmatrix} = \begin{pmatrix} x_k \\ y_k \end{pmatrix} - \alpha_k \begin{pmatrix} \Psi'_x(x_k, y_k) \\ \Psi'_y(x_k, y_k) \end{pmatrix}, \quad k = 0, 1, 2, \ldots \tag{28}$$

Для выбора числового параметра α_k оптимальным шагом в направлении антиградиента будет такой шаг, при котором функция $\Psi(x_k, y_k)$ будет принимать наименьшее значение среди всех значений в этом фиксированном направлении. При этом точка (x_{k+1}, y_{k+1}) будет точкой условного минимума:

$$\Psi(x_{k+1}, y_{k+1}) = \min_{\alpha > 0} \Psi(x_k + \alpha p_k, y_k + \alpha q_k). \tag{29}$$

Метод по формулам (28), (29) называется *методом наискорейшего спуска.*

На практике шаг α_k не всегда удается вычислить аналитически, поэтому применяют численный метод одномерной минимизации и находят α_k приближенно. Задача минимизации функции одной переменной $\psi_k(\alpha) = \Psi(x_k + \alpha p_k, y_k + \alpha q_k)$ проще, чем решение нелинейной системы. Эффективным может оказаться такой простой способ,

когда на каждом шаге выбирают достаточно малое число $\alpha > 0$ таким образом, чтобы выполнялось условие $\Psi(x_{k+1}, y_{k+1}) < \Psi(x_k, y_k)$. Если оно выполняется, то переходим к следующему шагу, если не выполняется, берем α в два раза меньше и снова проверяем неравенство. Такое α_k существует по определению градиента, если вектор градиента не равен нулю. Если же градиент равен нулю, то мы уже находимся в стационарной точке.

Главное достоинство — глобальная сходимость: для любого начального приближения метод приведет к стационарной точке (x_*, y_*). Для найденного значения нужно только проверить, что $\Psi(x_*, y_*) = 0$. Если оно выполняется, то это ответ, если нет, нужно взять другое начальное приближение.

3.8. Метод Брауна

Систему можно записать одним уравнением $f(x) = 0$ относительно векторной функции f векторного аргумента x. В методе Ньютона–Рафсона рассматривалась пошаговая линеаризация этой векторной функции $f(x)$ — см. (19). В отличии от метода Ньютона–Рафсона, Брауном был предложен следующий метод. На каждом итерационном шаге проводим поочередную линеаризацию компонент вектор–функции $f(x)$, то есть линеаризовываем в системе (16) сначала функцию f_1, затем f_2 и т. д., и последовательно решаем получаемые таким образом уравнения. Рассмотрим данную идею в двумерном случае при решении системы (25). При выбранных начальных значениях x_0, y_0 каждое последующее приближение по методу Брауна находится при $k = 0, 1, 2, \ldots$ с помощью формул

$$\tilde{x}_k = x_k - \frac{f(x_k, y_k)}{f'_x(x_k, y_k)}, \;\; q_k = \frac{g(\tilde{x}_k, y_k) f'_x(x_k, y_k)}{f'_x(x_k, y_k) g'_y(x_k, y_k) - f'_y(x_k, y_k) g'_x(\tilde{x}_k, y_k)},$$

$$p_k = \frac{f(x_k, y_k) - q_k f'_y(x_k, y_k)}{f'_x(x_k, y_k)}, \;\; x_{k+1} = x_k - p_k, \;\; y_{k+1} = y_k - q_k,$$

счет по которым должен выполняться в той очередности, в которой они записаны. Вычисления в методе Брауна естественно заканчивать, когда выполнится неравенство $\max\{|p_{k-1}|, |q_{k-1}|\} < \varepsilon$ с результатом $(x_*, y_*) \approx (x_k, y_k)$. В ходе вычислений следует контролировать немалость знаменателей расчетных формул. Заметим, что

функции f и g в этом методе неравноправны, и перемена их ролями может изменить ситуацию со сходимостью.

4. Приближение функций и аппроксимация

На практике часто возникают и изучаются функции, описывающие сложные реальные процессы (физические, химические и пр.). Обычно эти функции не удобны для практических расчетов по той причине, что вычисление их значений требует больших затрат машинного времени или их запоминание требует достаточно больших объемов памяти ЭВМ. Иногда возникает необходимость определить функцию по сеточным значениям, найденным приближенно с помощью эксперимента. Используя методы теории приближения функций, удается подобрать простые и удобные для расчетов функции, близкие к исходным. Такая подмена «плохой» функции $f(x)$ «хорошей» функцией $\varphi(x)$ называется *аппроксимацией* функции $f(x)$ функцией $\varphi(x)$. В качестве «хороших» функций будем использовать многочлены: их достаточно легко вычислять, они однозначно определяются набором своих коэффициентов и многочлен линейно зависит от своих коэффициентов. Конкретный способ аппроксимации будет зависеть от того, к какому классу принадлежит аппроксимируемая функция $f(x)$, и что понимается под «близостью» между функциями $f(x)$ и $\varphi(x)$.

4.1. Интерполяционный многочлен Лагранжа

Пусть некоторая функция $f(x)$ задана таблицей своих значений в некоторых точках x_i, называемых *узлами*, $f(x_i) = y_i$, $i = 0, 1, \ldots, n$:

x	x_0	x_1	$\ldots$	x_n
y	y_0	y_1	$\ldots$	y_n

.

Задача интерполяции заключается в том, чтобы построить такую функцию $\varphi(x)$, чтобы выполнялись условия $\varphi(x_i) = f(x_i)$ в узлах интерполяции, а в остальных точках функция $\varphi(x)$ была близка к функции $f(x)$. Если в качестве $\varphi(x)$ взять алгебраический многочлен, удовлетворяющий условиям $\varphi(x_i) = y_i$, $i = 0, 1, \ldots, n$, то его называют *интерполяционным многочленом Лагранжа* и обознача-

ют $L_n(x)$, причем для любого набора узлов $\{x_i\}_{i=0}^n$: $x_i \neq x_j$ при $i \neq j$ и для любого набора значений функции $\{y_i\}_{i=0}^n$ многочлен степени n $L_n(x)$ существует и однозначно определяется условиями $L_n(x_i) = y_i$, $i = 0, 1, \ldots, n$.

Интерполяционный многочлен Лагранжа имеет вид

$$L_n(x) = \sum_{i=0}^{n} y_i \frac{\prod\limits_{j \neq i} (x - x_j)}{\prod\limits_{j \neq i} (x_i - x_j)} = \sum_{i=0}^{n} y_i \frac{\omega(x)}{(x - x_i)\omega'(x_i)}, \tag{30}$$

где $\omega(x) = \prod\limits_{i=0}^{n} (x - x_i) = (x - x_0)(x - x_1) \cdots (x - x_n)$.

Погрешность $r(x) = f(x) - L_n(x)$ в узлах интерполяции по определению равна $r(x_i) = 0$ при $i = 0, 1, \ldots, n$. Пусть $x \in [a, b]$ и $x \neq x_i$ при $i = 0, 1, \ldots, n$, и пусть $f(x)$ имеет непрерывные производные до $(n+1)$-го порядка на отрезке $[a, b]$. Тогда погрешность определяется как $r(x) = f(x) - L_n(x) = \dfrac{f^{(n+1)}(\xi)}{(n+1)!}\omega(x)$, где $\xi \in [a, b]$. Обозначив через $M_{n+1} = \sup\limits_{x \in [a,b]} \left| f^{(n+1)}(x) \right|$, имеем оценку погрешности многочлена Лагранжа

$$|r(x)| \leqslant \frac{M_{n+1}}{(n+1)!} |\omega(x)| . \tag{31}$$

Замечание. Если $f(x) = P_n(x)$ — многочлен n-ой степени, то $r(x) = 0$, то есть $P_n(x) = L_n(x)$. Любой многочлен n-ой степени однозначно восстанавливается по своему $n + 1$ значению, взятому в любых различных точках.

Интерполяционный многочлен Лагранжа можно построить при любом расположении узлов интерполяции, однако он имеет один существенный недостаток. Если понадобится увеличить степень прибавлением нового узла, то коэффициенты многочлена Лагранжа придется вычислять заново, так как каждый его член зависит от всех узлов интерполирования. Интерполяционный многочлен Ньютона лишен этого недостатка. Его коэффициенты выражаются через разностные отношения различных порядков.

4.2. Интерполяционный многочлен Ньютона

Пусть функция $f(x)$ задана таблицей

x	x_0	x_1	$\ldots$	x_n
$f(x)$	$f(x_0)$	$f(x_1)$	$\ldots$	$f(x_n)$

Значения табличной функции $f(x_0)$, $f(x_1), \ldots, f(x_n)$ называются разделенными разностями нулевого порядка. Разделенная разность первого порядка равна $f(x_i, x_{i+1}) = \dfrac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i}$. Отношение

$$f(x_i, x_{i+1}, x_{i+2}) = \frac{f(x_{i+1}, x_{i+2}) - f(x_i, x_{i+1})}{x_{i+2} - x_i}$$

называется разделенной разностью второго порядка. Разделенная разность k-го порядка равна

$$f\left(x_i, x_{i+1}, x_{i+2}, \ldots, x_{i+k+1}\right) =$$

$$= \frac{f\left(x_{i+1}, x_{i+2}, \ldots, x_{i+k+1}\right) - f\left(x_i, x_{i+1}, \ldots, x_{i+k}\right)}{x_{i+k+1} - x_i},$$

где $k = 0, 1, \ldots, n$; $i = 0, 1, \ldots, n - k - 1$. Условимся располагать таблицу разделенных разностей следующим образом:

x_0	$f(x_0)$				
		$f(x_0, x_1)$			
x_1	$f(x_1)$		$f(x_0, x_1, x_2)$		
		$f(x_1, x_2)$		$f(x_0, x_1, x_2, x_3)$	
x_2	$f(x_2)$		$f(x_1, x_2, x_3)$		$f(x_0, x_1, \ldots, x_4)$
		$f(x_2, x_3)$		$f(x_1, x_2, x_3, x_4)$	
x_3	$f(x_3)$		$f(x_2, x_3, x_4)$		
		$f(x_3, x_4)$			
x_4	$f(x_4)$				

Разделенные разности $f(x_0, x_1)$, $f(x_0, x_1, x_2), \ldots$, $f(x_0, x_1, \ldots, x_n)$ являются числами, которые однозначно определяются заданной таблицей. *Интерполяционным многочленом Ньютона* называется выражение

$$N_n(x) = f(x_0) + (x - x_0)f(x_0, x_1) + (x - x_0)(x - x_1)f(x_0, x_1, x_2) + \\ + \ldots + (x - x_0)(x - x_1)\cdots(x - x_{n-1})f(x_0, x_1, \ldots, x_n)$$

или

$$N_n(x) = f(x_n) + (x - x_n)f(x_{n-1}, x_n) + (x - x_{n-1})(x - x_n)\cdot \\ \cdot f(x_{n-2}, x_{n-1}, x_n) + \ldots + (x - x_1)\cdots(x - x_n)f(x_0, x_1, \ldots, x_n).$$

Заметим, что многочлены Лагранжа и Ньютона совпадают, это просто различная форма записи и вычисления. Если по данной таблице построен многочлен Лагранжа, то при добавлении нового узла требуется полный пересчет многочлена Лагранжа, в то время как для многочлена Ньютона добавится всего лишь один член, так как новый узел можно приписать в конце таблицы, независимо от его величины. Поскольку многочлены Лагранжа и Ньютона отличаются только формой записи, представление погрешности в виде (31) справедливо и для формулы Ньютона.

4.3. Интерполяционный многочлен Эрмита

При построении многочленов Лагранжа и Ньютона требовалось лишь совпадение значений функции в точках. Иногда, если имеется информация о производных функции $f(x)$, требуется обеспечить совпадение и производных до некоторого порядка. В более общей постановке задача интерполяции состоит в следующем. Имеем набор узлов x_0, x_1, $x_2, \ldots, x_n : x_i \neq x_j$ при $i \neq j$. Пусть в некоторых узлах задана таблица значений функции и ее производных:

$$\begin{cases} f(x_i) = y_i, & \forall i = 0, 1, \ldots, n, \\ f^{(j)}(x_i) = y_i^{(j)}, & j = 1, 2, \ldots, N_i - 1, \ \forall i = 0, 1, \ldots, n. \end{cases} \tag{32}$$

Многочлен, который в узлах совпадает не только со значениями функции, но и производными заданного порядка называется *интерполяционным многочленом Эрмита*. Заметим, что для каждого узла может быть свое количество заданных производных. Обозначим многочлен Эрмита через $H_m(x)$:

$$H_m(x) = a_m x^m + a_{m-1} x^{m-1} + \ldots + a_1 x + a_0,$$

где степень многочлена $m = N_0 + N_1 + \ldots + N_n - 1$. Существование и единственность многочлена Эрмита следует из решения соответствующей СЛАУ для коэффициентов a_i при использовании условий интерполяции (32).

Пример. Построим интерполяционный многочлен Эрмита по таблице значений:

x_k	$f(x_k)$	$f'(x_k)$
1	2	0
3	6	–

.

Многочленом Эрмита будет квадратичная функция

$$H_2(x) = a_2x^2 + a_1x + a_0$$

с производной $H_2'(x) = 2a_2x + a_1$. Используем условия интерполяции

$$\begin{cases} H_2(1) = a_2 + a_1 + a_0 = 2, \\ H_2(3) = 9a_2 + 3a_1 + a_0 = 6, \\ H_2'(1) = 2a_2 + a_1 = 0, \end{cases} \quad \begin{cases} a_2 = 1, \\ a_1 = -2, \\ a_0 = 3. \end{cases}$$

Тогда $H_2(x) = x^2 - 2x + 3$.

Обозначим $\Omega(x) = (x - x_0)^{N_0}(x - x_1)^{N_1} \cdots (x - x_n)^{N_n}$. Если $f \in \mathbb{C}^{m+1}[a, b]$, то погрешность имеет вид

$$r(x) = f(x) - H_m(x) = \frac{f^{(m+1)}(\xi)}{(m+1)!}\Omega(x). \tag{33}$$

Оценка погрешности многочлена Эрмита $|r(x)| \leqslant \dfrac{M_{m+1}}{(m+1)!}\,|\Omega(x)|$, где $M_{m+1} = \sup\limits_{[a,b]} \left|f^{(m+1)}(x)\right|$.

Замечание. Если узлы интерполяции x_0, x_1, x_2, …, x_n простые, то есть $n = m$, то интерполяционный многочлен Эрмита совпадает с многочленом Лагранжа L_n, и совпадает оценка погрешности. Если рассмотреть другую крайность: x_0 — единственный узел кратности $m+1$, то есть в точке x_0 заданы производные до m-го порядка включительно, то многочлен Эрмита совпадает с многочленом Тейлора $H_m(x) = P_m(x) = \sum\limits_{k=0}^{m} \dfrac{f^{(k)}(x_0)}{k!}(x - x_0)^k$.

4.4. Интерполирование сплайнами

Рассмотренные выше аппараты приближения оказываются неудовлетворительными в ряде прикладных задач. Например, алгебраические многочлены обладают рядом недостатков как аппарат приближения для функций с особенностями и для функций с небольшой гладкостью. Эти трудности можно преодолеть, если разбить основной отрезок на частичные отрезки и на каждом из таких отрезков аппроксимировать функцию своим многочленом. Однако в ряде задач дополнительно требуется, чтобы приближающая функция была достаточно гладкой. В этом случае следует позаботиться, чтобы

в смежных точках приближающие агрегаты склеивались достаточно гладко. Это, по существу, и приводит нас к понятию сплайна.

В общем случае, *сплайнами* называют функции, склеенные из различных кусков заданных стандартных функций. В вычислительной математике важнейшую роль среди сплайнов играют функции, склеенные из кусков алгебраических многочленов. Мы ограничимся лишь такими сплайнами. Стоит отметить еще, что поведение многочленов в окрестности какой-либо точки определяет их поведение в целом. Сплайны лишены и этого недостатка.

Пусть на конечном отрезке $[a, b] \subset \mathbb{R}$ задана сетка

$$\Delta_n : a = x_0 < x_1 < \ldots < x_n = b.$$

Определение. *Полиномиальным сплайном степени m дефекта k* $(1 \leqslant k \leqslant m)$, построенным на сетке Δ_n, называется функция, удовлетворяющая следующим условиям:
1) $\forall x \in [x_i, x_{i+1}] : S_{m,k}(x) = P_m(x)$;
2) $S_{m,k} \in \mathbb{C}^{(m-k)}[a, b]$, а производная $(m - k + 1)$-го порядка может иметь разрывы в точках x_i, которые называются *узлами сплайна*.

Если выполнены условия $S_{m,k}(x_i) = f(x_i),\ i = 0, 1, \ldots, n$, то сплайн называется *интерполяционным*.

Определение. *Интерполяционным кубическим сплайном*, построенным на сетке Δ_n для функции f, называется функция $S_3(x) = S_3(x, \Delta_n, f)$, удовлетворяющая условиям
1) $S_3(x) = P_3(x),\ x \in [x_i, x_{i+1}],\ i = 0, 1, \ldots, n-1$;
2) $S_3(x) \in \mathbb{C}^{(2)}[a, b]$;
3) выполняются условия интерполяции $S_3(x_i) = f(x_i),\ i = 0, 1, \ldots, n$;
4) выполняется одно из краевых условий:
(I) $S_3^{(i)}(a) = S_3^{(i)}(b),\ i = 1, 2$;
(II) $S_3'(a) = a_n,\ S_3'(b) = b_n$;
(III) $S_3''(a) = A_n,\ S_3''(b) = B_n$;
(IV) $S_3'''(x_1 - 0) = S_3'''(x_1 + 0),\ S_3'''(x_{n-1} - 0) = S_3'''(x_{n-1} + 0)$.

Условие (I) используется, если $f(x)$ — $(b-a)$-периодическая функция. В условии (II) полагают $a_n = f'(a)$, $b_n = f'(b)$, если производная функции f существует, иначе — используют разделенные разности первого порядка

$$a_n = f(x_0, x_1) = \frac{f(x_1) - f(a)}{x_1 - a},\ b_n = f(x_{n-1}, x_n) = \frac{f(b) - f(x_{n-1})}{b - x_{n-1}}.$$

Аналогично в условии (III) полагают $A_n = f''(a)$, $B_n = f''(b)$, если функция f имеет производные второго порядка, в противном случае, полагают равным разделенным разностям второго порядка $A_n = f(x_0, x_1, x_2)$, $B_n = f(x_{n-2}, x_{n-1}, x_n)$. Условие (IV) означает, что мы «ликвидируем» узлы x_1, x_{n-1}.

Кубический интерполяционный сплайн, удовлетворяющий одному из краевых условий (I)–(IV) существует и единственный.

Введем обозначения: $f_i = f(x_i)$, $M_i = S_3''(x_i)$, $h_i = x_{i+1} - x_i$, $i = 0, 1, \ldots, n$. Тогда для $\forall x \in [x_i, x_{i+1}]$ локальное представление сплайна имеет вид

$$S_3(x) = -\frac{M_i}{6h_i}(x - x_{i+1})^3 + \frac{M_{i+1}}{6h_i}(x - x_i)^3 + \\ + \left(\frac{f_{i+1} - f_i}{h_i} - \frac{h_i}{6}(M_{i+1} - M_i)\right)(x - x_i) + f_i - \frac{M_i h_i^2}{6}. \quad (34)$$

Для нахождения M_i имеем систему при $i = 1, 2, \ldots, n-1$

$$\frac{1}{6}h_{i-1}M_{i-1} + \frac{1}{3}(h_{i-1} + h_i)M_i + \frac{1}{6}h_i M_{i+1} = \frac{f_{i+1} - f_i}{h_i} - \frac{f_i - f_{i-1}}{h_{i-1}}, \quad (35)$$

к которой осталось добавить краевые условия:

в случае краевых условий (I) типа: $M_0 = M_n$, $M_{n+1} = M_1$, $h_n = h_0$;

в случае (II): $2M_0 + M_1 = \dfrac{6}{h_0}\left(\dfrac{f_1 - f_0}{h_0} - a_n\right)$,

$$M_{n-1} + 2M_n = \frac{6}{h_{n-1}}\left(b_n - \frac{f_n - f_{n-1}}{h_{n-1}}\right);$$

в случае (III): $M_0 = A_n$, $M_n = B_n$;

в случае (IV): $M_0 = \left(1 + \dfrac{h_0}{h_1}\right)M_1 - \dfrac{h_0}{h_1}M_2$,

$$M_n = \left(1 + \frac{h_{n-1}}{h_{n-2}}\right)M_{n-1} - \frac{h_{n-1}}{h_{n-2}}M_{n-2}.$$

Матрица полученной системы имеет трехдиагональный вид, причем в силу неравенства $\dfrac{1}{3}(h_i + h_{i-1}) > \dfrac{1}{6}h_{i-1} + \dfrac{1}{6}h_i$ имеем матрицу с доминирующей главной диагональю. А значит, определитель матрицы A не равен нулю, и система (35) имеет единственное решение.

Решая систему (35) методом прогонки (см. 1.5.), находим коэффициенты M_i. Подставляем их в формулу (34) для нахождения сплайна, получая его представление на каждом отрезке $[x_i, x_{i+1}]$.

Если $h_i = h = \text{const}$, то матрица A имеет простой вид

$$A = \frac{h}{6} \cdot \begin{pmatrix} 4 & 1 & 0 & 0 & \cdots & 0 \\ 1 & 4 & 1 & 0 & \cdots & 0 \\ 0 & 1 & 4 & 1 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & \cdots & \cdots & 0 & 1 & 4 \end{pmatrix}$$

4.5. Метод наименьших квадратов

При построении интерполяционных многочленов мы требовали совпадения многочлена с заданными значениями функции (точное совпадение). Однако на практике значения функции задаются как результаты экспериментов. Причем характерной особенностью таких задач является то, что эти данные — заведомо приближенные. Поэтому нет смысла требовать абсолютного совпадения функций $f(x)$ и $\varphi(x)$ в заданных точках. С другой стороны, исследователи стремятся к накоплению как можно большего количества данных, с тем чтобы, усредняя их каким-либо образом, получить требуемые параметры изучаемого явления наиболее точно. Но при увеличении количества данных растет степень интерполяционного многочлена, что не совсем логично с точки зрения самой задачи аппроксимации: ведь мы хотели получить функцию попроще. Поэтому есть смысл заранее ограничить степень многочлена, не связывая ее с количеством заданных значений и не требуя абсолютного совпадения значений с исходными данными. Тогда можно перейти к другой постановке задачи апроксимации.

Пусть дана таблица значений функции $f(x)$:

x	x_0	x_1	$\ldots$	x_n
y	y_0	y_1	$\ldots$	y_n

.

В качестве аппроксимирующей функции $\varphi(x)$ будем брать многочлен $Q_m(x)$ степени m вне зависимости от количества заданных узлов. Близость $\varphi(x)$ и $f(x)$ будем определять по величине — сумма

квадратов отклонений принимает наименьшее значение:

$$S = \sum_{i=0}^{n} \left(Q_m(x_i) - y_i\right)^2 \to \min\,. \tag{36}$$

Будем строить $Q_m(x) = a_0 + a_1 x + \cdots + a_m x^m$, $m \leqslant n$, такой, что сумма разностей в квадрате между значениями многочлена и табличными была бы минимальной. Если $m = n$, то минимум, равный нулю, достигается, очевидно, для интерполяционного многочлена Лагранжа, поэтому рассмотрим случай, когда $m < n$. Обозначим через

$$S_k = \sum_{i=0}^{n} x_i^k,\ \forall k = 0, 1, \ldots, 2m; \quad t_k = \sum_{i=0}^{n} y_i x_i^k,\ \forall k = 0, 1, \ldots, m,$$

где $S_0 = n+1$. Тогда имеем систему для нахождения коэффициентов многочлена a_i

$$\begin{cases} S_0 a_0 + S_1 a_1 + \cdots + S_m a_m = t_0, \\ S_1 a_0 + S_2 a_1 + \cdots + S_{m+1} a_m = t_1, \\ \ldots \\ S_m a_0 + S_{m+1} a_1 + \cdots + S_{2m} a_m = t_m. \end{cases} \tag{37}$$

Если узлы различны, то есть $x_i \neq x_j$ при $i \neq j$ и $m \leqslant n$, то определитель системы (37) отличен от нуля и, следовательно, эта система имеет единственное решение. Тогда многочлен $Q_m(x)$ с такими коэффициентами будет обладать минимальным квадратическим отклонением среди всех многочленов степени m.

Пример. Дана таблица результатов эксперимента:

x	0	1	2	3
y	1	2	2.5	3.5

.

Построим многочлен первой степени, который наилучшим образом аппроксимирует данную табличную функцию. Геометрически это означает, что мы хотим на плоскости провести такую прямую, которая наиболее близко расположена к данным четырем точкам. $Q_1(x) = a_0 + a_1 x$; $m = 1$, $n = 3$, $S_0 = n + 1 = 4$, $S_1 = \sum\limits_{i=0}^{3} x_i = 6$,

$S_2 = \sum\limits_{i=0}^{3} x_i^2 = 14$, $t_0 = \sum\limits_{i=0}^{3} y_i = 9$, $t_1 = \sum\limits_{i=0}^{3} y_i x_i = 17.5$. Тогда система (37) для нашего примера примет вид $\begin{cases} 4a_0 + 6a_1 = 9, \\ 6a_0 + 14a_1 = 17.5. \end{cases}$ Решая ее, найдем $a_1 = 0.8$, $a_0 = 1.05$ и получим ответ $Q_1(x) = 0.8x + 1.05$.

5. Численное дифференцирование

Необходимость формул численного дифференцирования обусловлена следующим:

1. В прикладных задачах чаще всего неизвестно аналитическое выражение для заданной функции, а если даже и известно, выражение для производной может быть громоздким для дальнейшего применения.

2. Желательно иметь однотипные, простые вычислительные процессы для нахождения производных.

3. Одним из универсальных методов решения дифференциальных уравнений является разностный метод, где вместо функции непрерывного аргумента, заданного в постановке задачи, ищут сеточную функцию. При этом дифференциальные операторы, входящие в постановку задачи, заменяются приближенными разностными.

Задача численного дифференцирования состоит в приближенном вычислении производных функции $f(x)$ по заданной таблице значений.

x	x_0	x_1	$\dots$	x_{i-1}	x_i	x_{i+1}	$\dots$	x_n
$f(x)$	f_0	f_1	$\dots$	f_{i-1}	f_i	f_{i+1}	$\dots$	f_n

Требуется определить приближенные значения производных этой функции в какой-либо точке x, т. е. найти $f^{(k)}(x)$, $k = 1, 2, \dots$, причем x не обязательно совпадает с одним из значений заданных точек x_i.

5.1. Метод неопределенных коэффициентов

Удобным способом построения формул численного дифференцирования является метод неопределенных коэффициентов. Здесь в качестве приближенного значени производной $f^{(k)}(x)$ рассматривают

линейную комбинацию заданных значений функции в узлах сетки

$$f^{(k)}(x) \approx \sum_{i=0}^{n} a_i f(x_i), \tag{38}$$

где коэффициенты a_i подбирают из некоторых требований. Можно, например, потребовать, чтобы формула (38) была точна для многочленов как можно более высокой степени. Вместо многочленов можно рассматривать другие классы функций и накладывать соответствующие ограничения.

Возьмем в качестве f многочлен $f(x) = c_0 + c_1 x + c_2 x^2 + \ldots + c_m x^m = \sum_{j=0}^{m} c_j x^j$ и потребуем, чтобы для такого многочлена при некотором k левые и правые части приближенного равенства (38) совпадали.

Рассмотрим случай $k = 1$, т. е. получим приближенную формулу для первой производной. Система для нахождения коэффициентов a_i имеет вид:

$$\begin{cases} \sum\limits_{i=0}^{n} a_i = 0, \\ \sum\limits_{i=0}^{n} a_i x_i = 1, \\ \sum\limits_{i=0}^{n} a_i x_i^2 = 2x, \\ \ldots \\ \sum\limits_{i=0}^{n} a_i x_i^m = m x^{m-1}. \end{cases} \tag{39}$$

Система (39) имеет единственное решение, если $m = n$ и все точки x_i различны. Решив эту систему и подставляя найденные a_i в (38), получим формулу для приближенного вычисления первой производной и эта формула будет точна для всех многочленов степени $m = n$. Например, если даны три точки x_0, x_1, x_2, т. е. $n = 2$, то формула (38) будет верна для всех многочленов степени $m = n = 2$.

Рассмотрим случай $k = 2$. Аналогично получается система с той же матрицей, что и в (39), но другим столбцом свободных членов

$$\sum_{i=0}^{n} a_i = 0, \ \sum_{i=0}^{n} a_i x_i = 0, \ \sum_{i=0}^{n} a_i x_i^2 = 2, \ \ldots, \ \sum_{i=0}^{n} a_i x_i^m = m(m-1)x^{m-2}.$$

Решение такой системы даст коэффициенты приближенной формулы (38) для второй производной в точке x. Аналогичным образом можно найти производные и других порядков.

Частные случаи

1. При $k = 1$ формула дает точное значение первой производной в любой точке x для всех многочленов первой степени ($n = 1$) по любым двум значениям $f(x_{i+1})$ и $f(x_i)$:

$$f'(x) \approx \frac{f(x_{i+1}) - f(x_i)}{h}, \text{ где } h = x_{i+1} - x_i,\ x \in [x_i, x_{i+1}]. \qquad (40)$$

Приближенная формула (40) имеет погрешность порядка h.

2. При $k = 1$, $m = n = 2$ приближенная формула

$$f'(x_i) \approx \frac{f(x_{i+1}) - f(x_{i-1})}{2h}, \text{ где } h = x_{i+1} - x_i = x_i - x_{i-1}, \qquad (41)$$

дает точное значение производной для всех многочленов до второй степени включительно. Приближенная формула (41) имеет погрешность порядка h^2, т. е. она лучше, чем (40).

⋆**Упр.** Проверьте, что формула (41) точна для многочленов до второй степени включительно.

Заметим, что мы нашли приближенную формулу для первой производной в центральной точке $x = x_i$ по трем значениям функции в точках x_{i-1}, x_i и x_{i+1}, симметрично расположенных относительно x_i. Однако точно так же можно найти формулу и для любой другой точки x. При этом коэффициенты формулы будут теперь зависеть от x (т. к. правая часть системы зависит от x), и она будет не столь красива как (41). Но она так же будет точна для всех многочленов второй степени.

3. При $k = 2$, $m = n = 2$ приближенная формула для второй производной по трем значениям функции в точках x_{i-1}, x_i и x_{i+1}, $h = x_{i+1} - x_i = x_i - x_{i-1}$, имеет вид

$$f''(x) \approx \frac{f(x_{i-1}) - 2f(x_i) + f(x_{i+1})}{h^2},\ x \in [x_{i-1}, x_{i+1}]. \qquad (42)$$

Формула (42) для аппроксимации второй производной имеет погрешность порядка h^2 и точна для всех многочленов до второй степени включительно для любого $x \in [x_{i-1}, x_{i+1}]$. В центральной точке эта формула точна и для всех многочленов третьей степени.

⋆**Упр.** Проверьте точность формулы (42).

Замечание. Увеличивая количество используемых точек, можно увеличить степень многочлена для которого формула будет точна. Например, если взять пять равномерно расположенных с шагом h точек x_{i-2}, x_{i-1}, x_i, x_{i+1}, x_{i+2}, то для приближенного вычисления первой производной в точке $x = x_i$ можно получить формулу

$$f'(x_i) \approx \frac{f(x_{i-2}) - 8f(x_{i-1}) + 8f(x_{i+1}) - f(x_{i+2})}{12h}, \qquad (43)$$

которая точна для всех многочленов четвертой степени.

⋆**Упр.** Докажите формулу (43), проверьте ее точность и найдите оценку ее погрешности.

Если выбрать вместо обычных многочленов другой класс функций и накладывать соответствующие ограничения, то получим другие приближенные формулы.

5.2. Интерполяционный метод

Идея метода проста: по заданной таблице значений функции на отрезке $[a, b]$ строят интерполяционный многочлен Лагранжа $L_n(x) \approx f(x)$ и полагают $f^{(k)}(x) \approx L_n^{(k)}(x)$. Рассмотрим неравномерную сетку $a = x_0 < x_1 < x_2 < \ldots < x_n = b$ и обозначим $h_i = x_i - x_{i-1}$ $(i = 1, \ldots, n)$, $f_i = f(x_i)$ $(i = 0, 1, \ldots, n)$.

Используя многочлен Лагранжа первой степени по двум значениям функции в точках x_i и x_{i-1}, получим $f'(x) \approx \dfrac{f_i - f_{i-1}}{h_i}$, что совпадает с приближенной формулой (40).

Используя интерполяционный многочлен Лагранжа второй степени по трем узлам x_{i-1}, x_i, x_{i+1}, получим приближенное значение первой производной на неравномерной сетке

$$f'(x) \approx f_{i-1}\frac{2x - x_i - x_{i+1}}{h_i(h_i + h_{i+1})} - f_i\frac{2x - x_{i+1} - x_{i-1}}{h_i h_{i+1}} + \\ + f_{i+1}\frac{2x - x_{i-1} - x_i}{(h_i + h_{i+1})h_{i+1}}, \quad x \in [x_{i-1}, x_{i+1}].$$

На равномерной сетке, например, для точки $x = x_i$ $(h_i = h_{i+1} = h)$, получаем центральную разностную формулу (41).

Вторая производная будет иметь приближенную формулу

$$f''(x) \approx \frac{1}{\overline{h}_i}\left(\frac{f_{i+1}-f_i}{h_{i+1}} - \frac{f_i - f_{i-1}}{h_i}\right), \quad x \in [x_{i-1}, x_{i+1}], \tag{44}$$

где $\overline{h}_i = \dfrac{h_{i+1}+h_i}{2}$. На равномерной сетке данное выражение для приближенного вычисления второй производной совпадает с (42). На неравномерной сетке ($h_i \neq h_{i+1}$) приближенная формула (44) имеет только первый порядок аппроксимации, на равномерной сетке только в точке $x = x_i$ будет второй порядок аппроксимации.

5.3. Численное дифференцирование при помощи интерполяционных сплайнов

Аналогично интерполяционному методу производные функции $f(x)$ заменяем производными интерполяционного сплайна.

Пусть на $[a, b]$ задана сетка Δ: $a = x_0 < x_1 < \ldots < x_n = b$. Рассмотрим на примере кубического сплайна $S_3(x)$, построенного по значениям $f_i = f(x_i)$, $i = 0, 1, \ldots, n$. $S_3(x)$ на каждом промежутке $x \in [x_i, x_{i+1}]$ представим в виде (34).

Обозначив $t = \dfrac{x - x_i}{h_i}$, имеем следующие формулы численного дифференцирования при $i = 0, 1, \ldots, n-1$

$$f'(x) \approx \frac{f_{i+1}-f_i}{h_i} - \frac{h_i}{6}\left[(2 - 6t + 3t^2)M_i + (1 - 3t^2)M_{i+1}\right],$$

$$f''(x) \approx M_i(1-t) + M_{i+1}t, \quad f'''(x) \approx \frac{M_{i+1} - M_i}{h_i}.$$

5.4. Метод Рунге–Ромберга

Пусть решается задача определения Y и есть алгоритм нахождения приближенного значения в зависимости от параметра h такой, что $\|Y - Y_h\| = O\left(\|h\|^m\right)$, т. е. остаточный член приближенной формулы Y_h

$$Y - Y_h = R \cdot h^m + O\left(h^{m+1}\right). \tag{45}$$

Выполним теперь вычисления по той же приближенной формуле, но с другим шагом (параметром сетки) rh:

$$Y - Y_{rh} = R \cdot (rh)^m + O\left(h^{m+1}\right), \tag{46}$$

где r — константа. Вычтем почленно из (45) формулу (46) и преобразуем, выразив $R \cdot h^m$:

$$R \cdot h^m = \frac{Y_{rh} - Y_h}{r^m - 1} + O\left(h^{m+1}\right). \tag{47}$$

Формулу (47) называют *первой формулой Рунге*. Первое слагаемое справа есть главный член погрешности, который позволяет оценить погрешность уже найденного значения Y_h. Подставим его в (45) и выразим Y:

$$Y = Y_h + \frac{Y_h - Y_{rh}}{r^m - 1} + O(h^{m+1}). \tag{48}$$

Получили *вторую формулу Рунге* для нахождения Y с более высокой точностью.

В частном случае $r = \dfrac{1}{2}$ формула принимает вид

$$Y = Y_{\frac{h}{2}} + \frac{Y_{\frac{h}{2}} - Y_h}{2^m - 1} + O\left(h^{m+1}\right). \tag{49}$$

Если мы хотим получить значение Y с заданной точностью ε, то вычисляем значения с шагами h и $\dfrac{h}{2}$ и проверяем выполнение неравенства:

$$|\delta| < \varepsilon, \text{ где } \delta = \frac{Y_{\frac{h}{2}} - Y_h}{2^m - 1}.$$

Если неравенство выполняется, то счет можно прекращать, взяв в качестве приближенного значения, найденное с заданной точностью $Y \approx Y_{\frac{h}{2}} + \delta$. Если же неравенство не выполняется, т. е. $\delta \geqslant \varepsilon$, то снова уменьшаем шаг в два раза, вычисляем новое значение и сравниваем два последних приближения и т. д.

Отсюда видно, что для получения высокой точности не обязательно проводить вычисления по формулам высокого порядка. Достаточно провести вычисления по простым формулам низкой точности на разных сетках и уточнить результат методом Рунге–Ромберга.

Замечание. При реализации метода, кроме ошибок связанных с величиной h, на результат оказывает влияние и округление.

Рассмотрим формулу второго порядка точности для приближенного вычисления производной в точке x_i с использованием соседних

узлов x_{i-1} и x_{i+1}

$$f'(x_i) = \frac{f(x_{i+1}) - f(x_{i-1})}{2h} + O(h^2) \text{ или } f'_h = \frac{f_{i+1} - f_{i-1}}{2h} + O(h^2).$$

А теперь запишем ту же формулу, используя узлы x_{i-2} и x_{i+2}

$$f'(x_i) = \frac{f(x_{i+2}) - f(x_{i-2})}{4h} + O(h^2) \text{ или } f'_{2h} = \frac{f_{i+2} - f_{i-2}}{4h} + O(h^2).$$

Имеем $p = 2$, коэффициент увеличения шага $r = 2$, поэтому по формуле Рунге–Ромберга получим

$$f'(x_i) = \frac{f(x_{i+1}) - f(x_{i-1})}{2h} + \\ + \frac{1}{3}\left[\frac{f(x_{i+1}) - f(x_{i-1})}{2h} - \frac{f(x_{i+2}) - f(x_{i-2})}{4h}\right] + O(h^3) \text{ или}$$

$$f'(x_i) = \frac{f(x_{i-2}) - 8f(x_{i-1}) + 8f(x_{i+1}) - f(x_{i+2})}{12h} + O(h^3).$$

Эта формула уже имеет третий порядок точности и она точна для всех многочленов до четвертой степени включительно.

Привлечение большого числа узлов приводит к громоздким формулам. Удобнее применять метод Рунге–Ромберга, проводя вычисления по простым формулам низкой точности на разных сетках.

5.5. Некорректность задачи численного дифференцирования

Некорректность обусловлена следующим:

1. Совпадение значений двух функций даже в сколь угодно большом количестве точек не гарантирует совпадение их производных ни в одной точке.

Пример. Рассмотрим функцию $y = \sin C\pi x$, где C — некоторое целое число. Пусть $x_i = i$, $i = 0, 1, \ldots, n$ — узлы, $y(x_i) = 0$. Тогда интерполяционный многочлен Лагранжа $L_n(x) \equiv 0$, $L'_n(x) \equiv 0$. Но у исходной функции $y'(x) = C\pi \cos C\pi x$, и при целых C в узлах сетки $|y'(x_i)| = |C\pi|$, которое может быть сколь угодно большим. Таким образом, близость функций не гарантирует близость их производных.

2. С другой стороны, заданные значения $f(x_i)$ сами определены приближенно. При использовании ЭВМ мы имеем дело с машинной арифметикой, где неизбежны ошибки округления. Погрешность, возникающая при вычислении разностных отношений, намного превосходит погрешность в задании значений функции $f(x)$ и даже может неограниченно возрастать при стремлении шага сетки h к нулю.

Пример. Рассмотрим функцию $f(x) = e^x$. Вычислим приближенные значения первой производной этой функции в точке $x = 1$, точное значение которой равно $e = 2.7182818285\ldots$. Используем формулу $f'(x_i) \approx \dfrac{f_{i+1} - f_i}{h}$ при различных шагах $h_j = 10^{-j}$. Все вычисления с 9 десятичными знаками помещены в таблицу.

h	$f(1+h)$	$f(1+h) - f(1)$	$\dfrac{f(1+h) - f(1)}{h}$	погрешность
0.1	3.004166024	0.285884196	2.858841960	0.140560132
0.01	2.745601015	0.027319187	2.731918700	0.013636872
0.001	2.721001470	0.002719642	2.719642000	0.001360172
10^{-4}	2.718553670	0.000271842	2.718420000	0.000138172
10^{-5}	2.718309011	0.000027183	2.718300000	0.000018172
10^{-6}	2.718284547	0.000002719	2.719000000	0.000718172
10^{-7}	2.718282100	0.000000272	2.720000000	0.001718172
10^{-8}	2.718281856	0.000000028	2.800000000	0.081718172
10^{-9}	2.718281831	0.000000003	3.000000000	0.281718172
10^{-10}	2.718281829	0.000000001	10.000000000	7.281718172

Из таблицы видим, что если взять $h < 10^{-5}$, то погрешность уже начинает расти.

Это не означает, что нельзя пользоваться формулами численного дифференцирования. Надо только следить, чтобы погрешность округления имела тот же порядок, что и погрешность аппроксимации. Работая с грубыми округлениями или с данными с большими погрешностями, нелогично и бесполезно пытаться получить хорошее приближение за счет малых h.

6. Лабораторные работы

6.1. Решение СЛАУ. Собственные значения и векторы

Решите систему линейных алгебраических уравнений, используя
1) метод Гаусса или его модификацию;
2) метод Жордана–Гаусса;
3) метод LU–разложения;
4) метод квадратных корней;
5) метод ортогонализации;
6) метод прогонки;
7) метод простых итераций;
8) метод Якоби;
9) метод Зейделя;
10) по возможности сравните результаты и оцените погрешность, или найдите невязку для каждого метода;
11) вычислите определитель матрицы двумя способами;
12) найдите обратную матрицу, оцените погрешность;
13) решите СЛАУ в комплексном пространстве;
14) найдите максимальное и минимальное по модулю собственные значения и соответствующие им собственные векторы;
15) найдите все собственные значения и соответствующие собственные векторы.

Варианты заданий

Даны матрица системы A и вектор правой части b:

$$A = \begin{pmatrix} j \cdot m & 0.5 \cdot j & 0 & 0.2 \cdot l & 0 \\ 0.5 \cdot j & j & 0.3 \cdot j & 0 & 0.1 \cdot l \\ 0 & 0.3 \cdot j & 10 & -0.3 \cdot j & 0.5 \cdot l \\ 0.2 \cdot k & 0 & -0.3 \cdot j & j & -0.1 \cdot j \\ 0 & 0.1 \cdot k & 0.5 \cdot k & -0.1 \cdot j & j \cdot m \end{pmatrix},$$

$$b = \begin{pmatrix} -j + 0.05 \cdot j^2 \\ -0.8 \cdot j + 0.1 \cdot j^2 - 0.02 \cdot l \cdot j \\ -10 + 0.03 \cdot j^2 - 0.1 \cdot l \cdot j \\ -0.2 \cdot k + 0.3 \cdot j + 0.02 \cdot j^2 \\ 0.01 \cdot k \cdot j - 0.5 \cdot k - 0.2 \cdot j^2 \end{pmatrix},$$

где $j = 1.5 + 0.1 \cdot NN$, NN — номер варианта.

Задания **1–3, 5, 7, 8, 11, 12** выполните при $k = j$, $l = k - 0.1$, задания **4, 9** — при $k = l = NN$, задание **6** — при $k = l = 0$. В заданиях **1–12** следует полагать $m = 1$.

Для задания **13** матрица СЛАУ A и вектор b имеют вид:

$$A = \begin{pmatrix} 1 - j \cdot i & 0 & -j \cdot i \\ -j - 2 \cdot i & -j \cdot i & 2 + j \cdot i \\ i & 2 & j \end{pmatrix}, \quad b = \begin{pmatrix} 1 + j - 3j \cdot i \\ 3j + 4 + 2j \cdot i \\ 2j + (j - 1) \cdot i \end{pmatrix},$$

где i — мнимая единица, $j = 1.5 + 0.1 \cdot NN$, NN — номер варианта.

В заданиях **14, 15** для реализации методов нахождения собственных значений симметричных положительно определенных матриц используйте матрицу из задания **4**, где $m = 10$, в противном случае — матрицу из задания **1**.

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

Реализуйте метод решения задачи на ЭВМ, используя системы компьютерной математики (Maple, Wolfram Mathematica, Mathcad и пр.) или языки программирования (по выбору студента).

Выведите на экран полученный результат, вычислите невязку или выполните проверку, используя встроенные функции; для итерационных методов дополнительно выведите последовательные приближения, число итераций, заданную точность.

Для успешной сдачи задания преподаватель может потребовать от студента:

1) сформулировать постановку задачи;

2) определить входные, выходные данные, их тип;

3) дать основные определения, относящиеся к изучаемому разделу;

4) определить тип задачи, к которой применим рассматриваемый метод;

5) сформулировать и проверить условия сходимости используемого метода, сделать соответствующие выводы;

6) изложить суть используемого метода;

7) пояснить алгоритм своей программы;

8) дать описание встроенных функций и подключаемых библиотек, используемых в программе.

6.2. Решение нелинейных уравнений и систем

Решите нелинейное уравнение, используя следующие методы:

1) метод половинного деления;
2) метод Ньютона–Рафсона (касательных);
3) метод секущих;
4) комбинированный метод;
5) метод простых итераций.

вариант	уравнение	отрезок
1	$e^x + x^2 - 2 = 0$	$[-4; -1.2]$
2	$e^{-x} - x + 2 = 0$	$[1.5; 5]$
3	$2\sin x + x + 0.4 = 0$	$[-1.5; 0]$
4	$2x - 4\cos x - 0.6 = 0$	$[-0.5; 1.5]$
5	$e^{-x} - x - 2 = 0$	$[-3; 0]$
6	$-\ln x + \frac{1}{x} = 0$	$[1.5; 4]$
7	$x^2 - 2\sin x + 0.7 = 0$	$[0.8; 3]$
8	$\cos x + x^2 - 5 = 0$	$[2; 3]$
9	$-e^{-x} + 6x^2 - 5 = 0$	$[0; 2]$
10	$x - 2\ln x - 2 = 0$	$[3; 6]$
11	$-\ln x + 0.2x^2 - 1 = 0$	$[2; 4]$
12	$-\ln x + \frac{1}{x^2} + 1 = 0$	$[2.5; 5]$
13	$5\,\mathrm{tg}\, x - \frac{10}{x^2} - 3 = 0$	$[-3; -1.7]$
14	$\frac{5x}{1+x^2} - x = 0$	$[1.8; 3]$
15	$\mathrm{arctg}(x) + x^2 - 2 = 0$	$[1; 3]$
16	$-\frac{5x}{1+x^2} + 2\cos(2x) - 1 = 0$	$[-1.5; -1]$
17	$-\arccos(x-1) + x^3 - 4 = 0$	$[0.5; 1.9]$
18	$\arcsin(x-1) + x - 2 = 0$	$[1.1; 1.9]$
19	$-\mathrm{arctg}(x-1) - x + 2 = 0$	$[1.1; 3]$
20	$-\arccos(x-2) - 2x + 6 = 0$	$[2.05; 2.8]$

вариант	уравнение	отрезок
21	$3\arcsin x + x^2 - 0.1 = 0$	$[-0.4; 0.5]$
22	$-\lg x + x - 2 = 0$	$[1; 3]$
23	$-3\lg x + x^2 - 2 = 0$	$[1; 5]$
24	$2^x - 2x - 0.7 = 0$	$[0; 1.5]$
25	$3^x - 7\sin x = 0$	$[1.5; 2.4]$

Решите систему нелинейных уравнений

$$\begin{cases} \sin(ax + b) + cy + d = 0, \\ \cos(ey + f) + gx + h = 0, \end{cases}$$

используя
6) метод Ньютона;
7) метод спуска;
8) сравните точности полученных решений для уравнений и систем, оцените погрешность для каждого метода или найдите невязку.

Коэффициенты a, b, c, d, e, f, g, h для системы указаны в таблице, i — номер соответствующего варианта.

i	a	b	c	d	e	f	g	h
1	1	$\pi/2 - 1$	3	-1.5	1	π	2	-4
2	1	2	-1	-0.2	2	0	1	1
3	2	$\pi/2$	1	-3	1	$\pi/2 - 1$	1	0.5
4	1	-2	1	-0.7	1	$\pi/2$	2	-0.6
5	1	$\pi/2 - 1$	1	-0.5	1	π	1	-3
6	1	0	2	-2	1	-1	1	-0.7
7	1	0.5	-1	-1	1	-2	1	0
8	1	2	-1	-1.5	1	-2	1	-0.5
9	1	$\pi/2 + 0.5$	-1	-2	-1	$\pi/2$	-2	-1
10	1	$\pi/2 - 1$	1	-0.7	-1	$\pi/2$	2	-2
11	-1	$\pi/2 + 2$	1	0	-1	$\pi/2 - 0.5$	-1	-1
12	1	0	-2	-1.6	1	0.5	1	-0.8

13	1	0.5	-1	-1.2	1	-2	1	0
14	1	$\pi/2-1$	1	-0.5	1	π	1	-3
15	1	0	2	-2	1	-1	1	-0.7
16	1	$\pi/2$	1	-1.5	1	$\pi/2-0.5$	2	-1
17	1	$3\pi/2+1$	2	0	-1	$\pi/2$	1	0.4
18	1	0	1	0.4	1	$\pi+1$	2	0
19	1	$\pi/2-2$	1	-0.5	-1	$\pi/2-2$	-1	-1.5
20	1	$\pi/2+0.5$	1	-1	-1	$\pi/2$	-2	-2
21	1	$\pi/2-1$	0	-2	-1	$\pi/2-1$	-1	1
22	1	$\pi/2-1$	1	-0.8	1	π	1	2
23	1	-1	1	-0.8	1	π	1	2
24	1	-1	2	0.5	1	2	1	-2
25	1	π	-1	-1.5	1	-2	1	-0.5

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

Выведите на экран полученный результат, вычислите невязку или выполните проверку, используя встроенные функции; для итерационного метода дополнительно выведите последовательные приближения, число итераций, заданную точность. Точность вывода значений функций должна соответствовать заданной точности вычислений (если $\varepsilon = 0.001$, то выводимое значение следует округлить до трех знаков после запятой).

Для успешной сдачи задания преподаватель может потребовать от студента:
1) сформулировать постановку задачи;
2) определить входные, выходные данные, их тип;
3) дать основные определения, относящиеся к изучаемому разделу;
4) сформулировать и проверить условия сходимости используемого метода, сделать соответствующие выводы;
5) изложить суть используемого метода;
6) пояснить алгоритм своей программы;
7) дать описание встроенных функций и подключаемых библиотек, используемых в программе.

6.3. Теория приближения и аппроксимация функций

1. Постройте интерполяционный многочлен Лагранжа для функции, заданной таблично:

1)	x	-3	-2	-1	0	1	2
y	-180	-19	2	3	8	65

2)	x	-1	0	1	2	3
y	11	7	7	17	67

3)	x	-3	-2	-1	0	1	2
y	-47	13	7	1	13	133

4)	x	-2	0	1	2	3
y	67	9	4	-1	17

5)	x	-3	-2	0	1	2
y	64	14	4	8	34

6)	x	-2	-1	0	1	2	3
y	12	6	-4	2	8	-10

7)	x	-2	-1	0	1	2
y	-7	-1	9	11	17

8)	x	-1	0	1	2	3	4
y	-243	-32	-1	0	1	32

9)	x	-4	-2	-1	0	1	2
y	-510	2	3	2	5	66

10)	x	-1	0	1	2	3
y	17	7	5	11	49

11)	x	-2	-1	0	1	2	3
y	6	-2	0	9	-4	10

12)	x	-2	-1	0	1	2
y	68	21	10	5	0

13)	x	-2	-1	0	1	2
y	17	0	5	8	9

14)	x	-3	-2	-1	0	1	2
y	-26	11	0	1	2	39

15)	x	-2	-1	0	1	2
y	88	19	4	-5	16

16)	x	-2	-1	0	1	3
y	86	17	2	-7	161

17)	x	-2	-1	0	1	2	3
y	6	18	6	0	20	246

18)	x	-1	0	1	2	3
y	22	8	4	8	24

19)	x	-3	-2	0	2	3
y	7	11	-3	9	-5

20)	x	-2	-1	0	1	2
y	68	21	10	5	0

2. Постройте интерполяционный многочлен Лагранжа для заданной функции $f(x)$ с заданными узлами. Оцените погрешность интерполяции.

вариант	функция $f(x)$	узлы x_i
1	$\sin x$	$0,\ \pi/6,\ \pi/4,\ \pi/2$
2	$\sin^2 x$	$\pi/6,\ \pi/4,\ \pi/3,\ \pi/2$
3	$\cos^2 x$	$0,\ \pi/6,\ \pi/4,\ \pi/3$
4	$\operatorname{ctg} x$	$\pi/6,\ \pi/4,\ \pi/3,\ \pi/2$
5	$\operatorname{tg} x$	$0,\ \pi/6,\ \pi/4,\ \pi/3$
6	$\cos x$	$0,\ \pi/6,\ \pi/4,\ \pi/3$
7	$2\sin x$	$\pi/6,\ \pi/4,\ \pi/3,\ \pi/2$

вариант	функция $f(x)$	узлы x_i
8	5^x	-1, 0, 1, 2
9	5^{-x}	-2, -1, 0, 1
10	4^{-x}	-2, -1, 0, 1
11	4^x	-1, 0, 1, 2
12	x^{-1}	0.25, 0.5, 1, 2
13	2^{-x}	-3, -2, -1, 0
14	2^x	-1, 0, 1, 2
15	3^x	-1, 1, 2, 3
16	3^{-x}	-2, -1, 0, 1
17	$\sin 2x$	0, $\pi/12$, $\pi/8$, $\pi/6$
18	3^{2-x}	-2, -1, 0, 1
19	2^{x-2}	-2, 1, 3, 4
20	2^{3-2x}	-2, -1, 0, 1, 2

3, 4. Постройте интерполяционный многочлен Ньютона для функций из заданий **1, 2** соответственно.

5. Постройте интерполяционный многочлен Эрмита:

1)

x	-1	0	1	2
y	9	8	5	0
y'	-24	0	-12	504
y''	230	-4	-58	

2)

x	-2	-1	0	1
y	8	-7	-8	-5
y'		17	0	1
y''		-176	8	-32

3)

x	-1	0	1	2
y	0	-5	-4	3
y'	-26	0	-2	
y''		4	-24	

4)

x	-2	-1	0	1
y	8	0	-4	2
y'	244	-13	0	31
y''		46	6	

5)

x	-1	0	1	2
y	-12	-10	-10	18
y'	43	0	-25	
y''	-298	14	-154	

6)

x	-2	-1	0	1
y	-48	34	16	30
y'		-44	0	20
y''		86	32	-22

7)

x	-1	0	1	2
y	191	200	-209	-16
y'	210	-200	-610	4080
y''	-330	-420	-330	

8)

x	-2	-1	0	1
y	-14	37	14	43
y'		-37	0	79
y''		-26	50	

9)

x	-1	0	1	2
y	-106	-100	-104	204
y'	31	0	-13	1720
y''		-4	-22	

10)

x	-1	0	1	2
y	2	1	-4	449
y'	-13	0	-17	2696
y''	138	0	-30	

11)

x	-2	-1	0	1
y	-72	1	-2	9
y'		-30	3	54
y''		138	0	282

12)

x	-2	-1	0	1
y	-5	7	7	1
y'		-13	0	-25
y''		90	-6	

13)

x	-1	0	1	2
y	4	8	6	514
y'	15	0	3	
y''	-78	-6	66	

14)

x	-1	0	1	2
y	-2	1	0	481
y'	17	0	1	2240
y''	-96		48	

15)

x	-1	0	1	2
y	2	1	2	257
y'	-8	0	8	1024
y''			56	

16)

x	0	1	2
y	2	3	130
y'	0	7	448
y''	0	42	

17)

x	-2	-1	0	1
y	3	0	-5	-6
y'		-13	0	-15
y''		100	4	-92

18)

x	-1	0	1	2
y	-5	-6	-3	10
y'	17	0	1	
y''	-176	8	-32	

19)

x	-1	0	1	2
y	11	10	7	2
y'	-24	0	-12	504
y''	230	-4	-58	

20)

x	-1	0	1	2
y	-5	-10	-9	-2
y'	-26	0	-2	
y''		4	-24	

6, 7. Постройте интерполяционный кубический сплайн для функций из заданий **1, 2** соответственно.

8. Используя метод наименьших квадратов, постройте прямую линейной регрессии, аппроксимирующую функцию, заданную таблично:

1)

x	-3	-2	-1	0	1	2
y	-4	-3	-3	-1	1	0

2)

x	-2	-1	0	1	2	3
y	12	6	4	2	8	10

3)

x	-3	-2	-1	0	1	2
y	-1.86	-1.91	-1.28	-1.31	-1.84	-1.65

4)

x	-2	-1	0	1	2	3
y	1	-1	2	1	-1	3

5)

x	-3	-2	-1	0	1	2
y	2	4	5	3	2	4

6)

x	-3	-1	1	3	4
y	-5	-1	-2	0	2

7)

x	-3	-1	0	1	3
y	-1	1	-2	0	1

8)

x	-2	-1	1	2	3
y	2	3	1	1	2

9)

x	-2	-1	0	1	2	3
y	-3	-1	2	1	1	3

10)

x	0	1	2	3	4	5	6
y	2.131	2.253	2.259	2.197	2.113	2.124	2.318

11)

x	0	1	2	3	4	5	6
y	-3.31	-3.53	-3.25	-3.19	-3.11	-3.24	-3.38

12)

x	0	1	2	3	4	5	6
y	6.312	6.531	6.654	6.918	7.117	7.146	7.289

13)

x	0	1	2	3	4	5	6
y	3.312	3.536	3.258	3.191	3.113	3.149	3.185

14)

x	-1	0	1	2	3
y	7.4	7.3	7.2	6.9	7.1

15)

x	-3	-2	-1	1	2
y	5.5	5.3	5.2	5.1	5.4

16)

x	-2	-1	1	2	3	4	5
y	3.121	3.314	3.276	3.198	3.155	3.142	3.211

17)

x	1	2	3	4	5	6	7	8
y	1.13	1.32	1.22	0.99	1.11	1.42	1.53	1.44

18)

x	-1	0	1	2	3	4	5	6
y	2.122	2.331	2.249	2.901	2.456	2.675	2.721	2.698

19)

x	-1	0	1	2	3	4	5	6
y	4.198	4.312	4.241	3.976	4.123	4.076	4.342	3.999

20)

x	-3	-2	-1	0	1	2	3	4	5
y	4.2	3.7	3.2	3.2	3.8	4.0	4.5	4.7	4.6

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

Выведите на экран полученный результат, выполните проверку, используя встроенные функции. Если постановка задачи содержит явно заданную функцию, постройте график интерполируемой и интерполирующей функции в одной координатной плоскости, сделайте выводы.

Для успешной сдачи задания преподаватель может потребовать от студента:

1) сформулировать постановку задачи;
2) определить входные, выходные данные, их тип;
3) дать основные определения, относящиеся к изучаемому разделу;
4) изложить суть используемого метода;
5) пояснить алгоритм своей программы;
6) дать описание встроенных функций и подключаемых библиотек, используемых в программе.

6.4. Численное дифференцирование

Дана функция

$$f(x) = \frac{NN}{2}\sqrt[k+4]{e^{-x}}\cos\left(\frac{x}{k+NN}\right) + \frac{\pi NN}{2},$$

где NN — номер Вашего варианта, $k = NN \pmod 2$.

1. Создайте таблицу с приближенными значениями производной функции $f'(x)$, используя формулы первого порядка точности.

2. Создайте таблицу с приближенными значениями первой производной функции $f'(x)$, используя формулы второго порядка точности.

3. Создайте таблицу приближенных значений производных функции $f''(x)$, вычисляемых по формулам второго порядка точности.

4. Используя интерполяционный метод, вычислите производные первого и второго порядков.

5. Вычислите производные первого и второго порядков при помощи сплайнов.

6. Сравните полученные результаты. Оцените погрешность вычислений.

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

Пусть $P := [-5; 5]$. Произвольным образом выберите отрезок $[a, b]$ на множестве P длиной 1. При выполнении заданий **1–3, 5** равномерно разбейте отрезок $[a, b]$ на N частей, в задании **4** требуется использование неравномерной сетки, полученной путем генерации каждого следующего значения узла сетки случайным образом. Число разбиений N задайте произвольно, но не менее 10.

Вычислите требуемые значения производных и сравните полученные результаты со значениями производной в точках, найденной при непосредственном дифференцировании. Результат представьте графически, либо в виде таблицы.

Для успешной сдачи задания преподаватель может потребовать от студента:

1) сформулировать постановку задачи;

2) определить входные, выходные данные, их тип;

3) дать основные определения, относящиеся к изучаемому разделу;

4) изложить суть используемого метода;

5) пояснить алгоритм своей программы;
6) дать описание встроенных функций и подключаемых библиотек, используемых в программе.

Список рекомендуемой литературы

1. Бахвалов Н. С., Лапин А. В., Чижонков Е. В. Численные методы в задачах и упражнениях. М.: Лаборатория Базовых Знаний, 2016. 355 с.
2. Бахвалов Н. С., Жидков Н. П., Кобельков Г. М. Численные методы. М.: Лаборатория Базовых Знаний, 2011. 640 с.
3. Бердышев В. И., Субботин Ю. Н. Численные методы приближения функций. Свердловск: Среднеуральское книжное изд-во, 1979. 120 с.
4. Вабищевич П. Н. Численные методы. Вычислительный практикум. URSS, 2016. 320 с.
5. Вержбицкий В. М. Численные методы (линейная алгебра и нелинейные уравнения). М.: ОНИКС 21 век, 2005. 432 с.
6. Вержбицкий В. М. Численные методы (математический анализ и обыкновенные дифференциальные уравнения). М.: ОНИКС 21 век, 2005. 400 с.
7. Даугавет И. К. Теория приближенных методов. Линейные уравнения. 2-е изд., перераб. и доп. СПб.: БХВ-Петербург, 2006. 288 с.
8. Демидович Б. П., Марон И. А. Основы вычислительной математики. СПб.: Лань, 2009. 672 с.
9. Демидович Б. П., Марон И. А., Шувалова Э. З. Численные методы анализа. Приближение функций, дифференциальные и интегральные уравнения. СПб.: Лань, 2010. 400 с.
10. Дьяконов В. П., MAPLE 10/11/12/13/14 в математических расчетах. М.: ДМК Пресс, 2014. 800 с.
11. Киреев В. И., Пантелеев А. В. Численные методы в примерах и задачах. СПб.: Лань, 2015. 448 с.
12. Латыпова Н. В. Теория приближения. Ижевск: Изд-во «Удм. ун-т», 2002. 97 с.
13. Латыпова Н. В. Приложения сплайнов. Ижевск: Изд-во «Удм. ун-т», 2005. 52 с.
14. Махмутов М. М. Лекции по численным методам. М.–Ижевск: НИЦ «Регулярная и хаотическая днамика», 2007. 238 с.
15. Самарский А. А. Введение в численные методы. СПб.: Лань, 2009. 288 с.
16. Фаддеев Д. К., Фаддеева В. И. Вычислительные методы линейной алгебры. СПб.: Лань, 2002. 736 с.

17. http://www.exponenta.ru/journal
18. http://num-anal.srcc.msu.su
19. YouTube-канал сообщества русскоязычной поддержки Wolfram Mathematica
    https://www.youtube.com/channel/UC7JeG87kenaPEjEJBA6tmKw
20. Онлайн-центр документации языка Wolfram Language
    https://reference.wolfram.com/language/

Рекомендации по выполнению лабораторных работ

Wolfram — весьма общий мультипарадигмальный язык программирования, разработанный компанией Wolfram Research, который служит основным связующим языком для системы Mathematica. Он был спроектирован как максимально универсальный язык, с акцентом на символьные вычисления, функциональное и логическое программирование.

Язык является достаточно большим и касается многих, часто специализированных, сфер знаний. Например, он обладает встроенными функциями для создания и приведения в действие машины Тьюринга, создания графики и аудио, анализа трехмерных моделей и решения дифференциальных уравнений. В частности, имеется большое количество функций, в которых уже реализованы некоторые алгоритмы, рассматриваемые в данном пособии. Возникает естественные вопрос: зачем самостоятельно реализовывать те или иные алгоритмы, если они уже есть в языке? Но чтобы рационально использовать встроенные функции, их нужно понимать. Во-первых, функции и библиотеки могут иметь побочные эффекты или поведение, которые сложно предусмотреть (предвидеть) без понимания алгоритмов. Получив баг в таком случае, можно долго и упорно пытаться его поймать и исправить, когда этого можно было избежать. Во-вторых, различные инструменты и библиотеки часто нужно «настраивать» — указывать какие алгоритмы, структуры данных и технологии использовать внутри. Без элементарных знаний придется либо пользоваться значениями по умолчанию (порой неэффективными именно для вашей задачи), либо выбирать наугад. В-третьих, есть множество задач, которые нельзя решить простым вызовом API библиотеки или фреймворка.

Основные особенности синтаксиса:
1) все встроенные функции пишутся с заглавной буквы и состоят из одного (Plot, Solve) или нескольких (FindRoot, RowReduce, LinearSolve) английских слов, или общепринятого международного сокращения (GCD — наибольший общий делитель, greatest common divisor);
2) аргументы указываются в квадратных скобках и разделяются запятыми (Solve[5x+6==-5,x]);

3) необязательные аргументы (опции) имеют вид `НазваниеОпции` → `Значение` (Solve[5x+6==7,x,Modulus->8]);
4) для часто используемых функций используются краткие обозначения: D — оператор дифференцирования, N — приведение к вещественному типу, и т. д.

Следует различать = (оператор Set), := (оператор SetDelayed), == (оператор Equal) и === (оператор SameQ).

Основным универсальным типом данных является список (оператор List), который визуально представлен фигурными скобками ({a,b,c} тоже самое, что и List[a,b,c]). Векторам соответствуют одномерные списки, матрицам — двумерные (списки списков), запись и хранение матриц производится построчно. Например, матрице $\begin{pmatrix}1 & 2 & 3\\ 4 & 5 & 6\\ 7 & 8 & 9\end{pmatrix}$ соответствует список {{1,2,3},{4,5,6},{7,8,9}}. Многие аргументы команд имеют вид списков. Например, для указания диапазона, на котором будет строится график, используется список, состоящий из трех элементов: имя переменной, левая граница, правая граница (Plot[Cos[s],{s,-π,6-E}]). Если в качестве аргумента некоторым функциям (имеющим атрибут Listable) передать список, то на выходе получится список из значений функции на каждом элементе списка (Sin[{0,π/2,π}] вернет список {0,1,0}). Это избавляет от необходимости писать циклы, делая код компактнее и прозрачнее. Такую же функциональность для пользовательских функций (которым не был назначен атрибут Listable) можно получить, используя функцию высшего порядка — Map, или в краткой форме /@: запись f/@list вернет значения функции f на элементах списка list.

Универсальным генератором списков по заданному выражению является функция Table. Ее синтаксис выглядит следующим образом: Table[expr, iter], где expr — выражение, подлежащее вычислению, iter — итератор, имеющий форму списка {переменная, нижняя_граница, верхняя_граница, шаг_изменения}. Если не указывается шаг, то он считается равным 1. Если не указана и нижняя граница, то изменение итератора начинается с 1. Например,
Table[Prime[k],{k,10}], или что то же самое, что и
Table[Prime[k],{k,1,10,1}] — список из первых десяти простых чисел.

Итераторов может быть несколько. Тогда они разделяются запятыми, самый внутренний итератор соответствует самому внешнему

уровню получающегося многомерного списка:

Table[i*j,{i,2,9},{j,2,9}] — таблица умножения.

Table[Binomial[n,m],{n,0,7},{m,0,n}] — треугольник Паскаля.

Обратите внимание, что границы значений второго итератора зависят от первого, поэтому на выходе получается структура, именуемая ragged array (дословно «неровный»), т. е. массив, который может содержать разное количество элементов на одном и том же уровне.

Имеется еще одна возможность «заставить» итератор принимать значения произвольного списка, которая используется, например, при построении интерполяционного многочлена Эрмита:

```
dataset={{-2,3},{-1,0,-13,100},{0,-5,0,-4},{1,-6,-15,-92}};
Table[h[i][set[[1]]]==set[[i+2]]],{set,dataset},{i,0,Length[set]-2,1}]
```

Здесь перебор ведется по списку dataset, каждый элемент которого есть список из аргумента, значения функции на этом аргументе, значения первой производной, и т. д., при этом для разных узлов количество значений может быть разное. Для доступа к элементу списка по его номеру используется функция Part, которая в краткой форме принимает вид двойных квадратных скобок. lst[[i]] — i-й элемент списка lst, lst[[i,j]] — j-й элемент i-го элемента, например, для матриц это означает элемент i-й строки и j-го столбца.

Аналогичным образом работают итераторы и в операторах процедурного стиля, например, в циклах Do. Эта конструкция позволяет производить однотипные действия, связанные с изменением счетчиков индексов. Синтаксис следующий: Do[body, iter], где body — одна или несколько операций (в этом случае они разделяются точкой с запятой, что является краткой формой оператора CompoundExpression), iter — итератор в одной из вышеописанных форм. Например, в степенном методе за одну итерацию производятся три действия: 1) умножается вектор на матрицу; 2) вычисляется очередное приближение собственного значения; 3) полученный на шаге 1 вектор нормируется для следующей итерации. Соответствующий фрагмент кода выглядит следующим образом:

```
Do[y1=mA.y0;
λ1=mA.y1.y1/y1.y1;
y0=y1/Max[Abs[y1]];
,{k,1,kmax,1}]
```

Здесь отчетливо проявляется особенность языка Wolfram, позволяющая избегать дополнительных циклов, так как в третьей строчке

операция / (деления) применяется к списку y1 целиком для получения нового списка y0, все элементы которого есть элементы списка y1, поделенные на одно и то же число — максимальный по модулю элемент списка y1.

Один из самых универсальных циклов While применяется в случае, когда количество итераций заранее неизвестно, но известны условия, при выполнении которых можно считать результат достигнутым. Синтаксис While имеет вид: While[cond, body], где body — одна или несколько операций, разделенных точкой с запятой (тело цикла), cond — условия, при истинности которых выполняется тело цикла. Как только условия cond становятся ложными, цикл прекращает свою работу. Типичная схема работы с неопределенным количеством итераций выглядит следующим образом: задаются переменная found со начальным значением False и ограничение на количество итераций kmax (скажем, 100 или 1000), для того чтобы избежать бесконечного выполнения в случае ошибочной реализации или очень медленной сходимости метода. Счетчик итераций k устанавливается равным 0. В теле цикла осуществляется увеличение на 1 счетчика итераций и вычисление условий остановки. Затем формируется условие Not[found]&&k<kmax, которое гарантирует конечную работу цикла до тех пор, пока либо условие остановки не будет выполнено, либо пока мы не превысим максимального количества итераций. Подобная конструкция применяется почти во всех представленных реализациях (например, в степенном методе, в методе поиска корней уравнений дихотомией, в методах секущих и градиентного спуска).

Отдельного обсуждения заслуживает условный оператор If, который может принимать от двух до четырех аргументов, первым из которых всегда является логическое выражение. Второй аргумент представляет собой набор действий, который выполняется при истинности логического выражения, указанного в первом аргументе. Третий аргумент — действия, которые будут выполнены при ложности условия, указанного в первом аргументе. Редко используемый четвертый аргумент применяется для указания действий в случаях, когда условия в первом аргументе не могут быть преобразованы ни к True, ни к False. Такая ситуация может возникнуть при использовании неинициализированных переменных или в случаях, когда в условиях вместо ожидаемых скалярных типов (числа, строки, списки определенной структуры) появляются другие значения (напри-

мер, как результат ошибки, возникшей перед выполнением условного оператора If). Например, если значение переменной x не определено, то конструкция y=If[x>0,x++,x- -,42] вернет 42, так как невозможно определить, является ли выражение x положительным, в то время как x=17;y=If[x>0,x++,x- -,42];x ожидаемо увеличит значение x с 17 до 18 (переменная y будет равна 17, так как в нее запишется значение x до увеличения на единицу в полном соответствии с семантикой оператора инкремента ++).

Задание пользовательских функций опирается на работу с шаблонами (_, или Blank[] в полной форме) и операторами Set (=) и SetDelayed (:=). Указание шаблона после имени формального параметра означает возможность при дальнейшей работе с функцией подставлять вместо аргумента произвольное допустимое выражение системы Mathematica. Сравните:

f[x]=x^2;
g[x_]=x^2;
{f[x],f[y],f[5]}
{g[x],g[y],g[5]}
{x^2, f[y], f[5]}
{x^2, y^2, 25}

В случае определения функции f без шаблона на выходе получается список {x^2, f[y], f[5]}, где только при полном совпадении реального и формального параметров произошли вычисления, ассоциированные с определением функции f. В случае же с функцией g возведение в квадрат было произведено для всех аргументов — {x^2, y^2, 25}.

Различия между = и := заключаются в порядке выполнения правых частей определений функций. Оператор Set сначала вычисляет значение правой части, а затем при последующих вызовах использует полученное при определении функции значение. Оператор SetDelayed сначала запоминает определение правой части без непосредственного вычисления и каждый раз при последующих вызовах подставляет переданные значения аргументов для расчета. Проще всего понять различие на следующем примере:

f[x_]=RandomInteger[1,100]
g[x_]:=RandomInteger[1,100]
{f[1],f[1],f[x]}
{g[1],g[1],g[x]}

```
99
{99,99,99}
{39,1,6}
```

В первом случае сначала было вычислено значение — псевдослучайное целое число из диапазона от 1 до 100, которое стало значением функции для любого аргумента благодаря использованию шаблона x_. Поэтому вызов функции f на любом аргументе возвращает одно и то же число (99 в данном примере). Во втором случае вычисление псевдослучайного целого числа производилось при каждом вызове функции g, поэтому даже на одном и том же аргументе получились различные значения (39 и 1), ведь процесс генерации запускался заново для каждого упоминания вызова функции g.

Такое различие в поведении операторов Set и SetDelayed при определении функций позволяет в одну строку реализовать подход, именуемый *мемоизацией*, т. е. запоминание вычисленных значений. Общий вид таких определений:

```
f[x_]:=f[x]=expr;
```

Как это работает? При вычислении на новом значении аргумента (для которого ранее значение функции не вычислялось) Mathematica использует определение, стоящее справа от :=, т. е. запись f[x]=expr целиком, при этом вычисленное значение согласно семантике Set запоминается для выражения f[x] (так как x здесь уже без шаблона, это становится дополнительным определением функции на конкретном аргументе). При повторном вызове функции на том же самом аргументе никаких дополнительных вычислений не производится, а берется значение, полученное в предыдущих вычислениях.

Например, для построения интерполяционного многочлена Ньютона задается рекурсивное определение конечных разностей, а затем ставится на счет вычисление самой старшей конечной разности:

```
f[x_List,0,i_Integer]:=f[x[[i]]]
f[x_List,k_Integer,i_Integer]:=
f[x,k,i]=(f[x,k-1,i+1]-f[x,k-1,i])/(x[[i+k]]-x[[i]])
Newton[t_]=Sum[f[x,i,n-i]*Product[(t-x[[j]])
,{j,n,n-i+1,-1}],{i,0,n-1}]//Expand
```

Так как аргумент n-i в вызове функции f[x,i,n-i] при изменении итератора i от 0 до n-1 уменьшается от n (наибольшего возможного значения) до 1, то при подсчете первого слагаемого суммы сначала разворачивается рекурсивное определение. Оно приводит к вы-

числению и запоминанию всех промежуточных конечных разностей, которые затем используются (уже без повторных вычислений) при формировании остальных слагаемых.

При выполнении лабораторных работ рекомендуется придерживаться следующих шагов:

1) получить задание согласно номеру варианта;

2) составить математическую модель решения задачи, предварительно ознакомившись с теоретическим материалом по данной тематике;

3) разработать алгоритм решения задачи в виде блок-схемы, хотя бы умозрительной; определить, что является входными значениями, и что ожидается на выходе;

4) написать текст программы в выбранной среде программирования;

5) выполнить и отладить программу;

6) произвести сравнение полученных результатов с результатом имеющихся в среде программирования встроенных функций, реализующих тот же самый алгоритм (при их наличии), или оценить погрешность найденнного решения;

7) сделать вывод о достижении намеченных целей.

Для выполнения пункта 6 в Mathematica могут оказаться полезными следующие функции:

– Solve, SolveAlways, LinearSolve при решении систем линейных уравнений;

– CholeskyDecomposition, JordanDecomposition, LUDecomposition, QRDecomposition при преобразованиях матриц;

– FindRoot при численном поиске корней нелинейных уравнений;

– Interpolation, InterpolatingPolynomial при построении интерполяционных многочленов;

– а также хорошо структурированная документация по языку Wolfram для поиска необходимых функций, где схожие по тематике функции собраны в одном разделе и имеют перекрестные ссылки друг на друга в разделе See Also (см. также).

Ниже представлены примеры реализации некоторых численных методов в пакете `Mathematica`.

1. Метод Гаусса с выбором главного элемента

In[1]:=
```
Clear["Global`*"]
mA=matrixA=RandomReal[{-10,10},{5,5}];
{nrows,mcolumns}=Dimensions[matrixA];
vb=vectorb=RandomReal[{-10,10},5];
ε=10^-10;
Row[{MatrixForm[mA],MatrixForm[vb]}]
Do[(* поиск главного элемента
      для текущей строки rowi *)
maxvalue=Abs@mA[[rowi,rowi]];
maxrow=rowi;
Do[currentvalue=Abs@mA[[rowk,rowi]];
If[currentvalue>maxvalue,
      maxvalue=currentvalue;maxrow=rowk];
,{rowk,rowi+1,nrows,1}
];
If[maxvalue<ε,
Print["Ведущий элемент слишком мал"];
Break[]];
(* обмен текущей строки на строку,
      содержащую главный элемент *)
mA[[{rowi,maxrow}]]=mA[[{maxrow,rowi}]];
vb[[{rowi,maxrow}]]=vb[[{maxrow,rowi}]];
(* пересчет коэффициентов матрицы системы
      и свободного члена *)
Do[vb[[k]]=vb[[k]]-
vb[[rowi]]*mA[[k,rowi]]/mA[[rowi,rowi]];
mA[[k]]=mA[[k]]-
mA[[rowi]]*mA[[k,rowi]]/mA[[rowi,rowi]];
,{k,rowi+1,nrows,1}];
,{rowi,1,nrows-1,1}];
If[Abs[mA[[nrows,nrows]]]<ε,
Print["Ведущий элемент слишком мал"]];
Row[{MatrixForm[mA],MatrixForm[vb]}]
(* обратный ход *)
vx=ConstantArray[0,nrows];
vx[[nrows]]=vb[[nrows]]/mA[[nrows,nrows]];
Do[vx[[k]]=(vb[[k]]-
```

```
        Sum[mA[[k,j]]vx[[j]]
              ,{j,k+1,nrows,1}])/mA[[k,k]]
        ,{k,nrows-1,1,-1}];
        mA.vx-vb
        matrixA.vx-vectorb
        (* сравнение со встроенным решателем *)
        vectorx=LinearSolve[matrixA,vectorb]
        matrixA.vectorx-vectorb
        mA.vectorx-vb
        vectorx-vx
```

(* исходные матрица системы и вектор правой части *)

$$\begin{pmatrix} 3.93229 & -7.91258 & 4.55594 & 7.88554 & -0.675419 \\ 4.10853 & 8.02245 & 3.08492 & 1.25662 & -1.44944 \\ -0.0215039 & 5.16696 & -6.813 & 5.647 & -7.73996 \\ 0.128485 & 0.971498 & -4.56485 & 1.84062 & -4.84594 \\ -9.54541 & -6.93898 & 8.81325 & -6.11977 & 8.4228 \end{pmatrix} \begin{pmatrix} 5.72731 \\ -1.36627 \\ 6.99154 \\ -0.0960034 \\ -1.53292 \end{pmatrix}$$

(* матрица и вектор после преобразований Гаусса *)

$$\begin{pmatrix} -9.54541 & -6.93898 & 8.81325 & -6.11977 & 8.4228 \\ 0. & -10.7711 & 8.18661 & 5.36446 & 2.7944 \\ 0. & 0. & 10.7058 & 1.13057 & 3.48236 \\ 0. & 0. & 0. & 8.54753 & -5.47309 \\ 0. & 0. & 0. & 0. & -1.61421 \end{pmatrix} \begin{pmatrix} -1.53292 \\ 5.09581 \\ 0.356353 \\ 9.54319 \\ -2.4723 \end{pmatrix}$$

(* векторы невязки для исходной и преобразованной систем *)

$$\{0.,\ 8.88178 \times 10^{-16},\ 0.,\ 1.77636 \times 10^{-15},\ 0.\}$$

$$\{-8.88178 \times 10^{-16},\ -1.33227 \times 10^{-15},\ 0.,\ 0.,\ 0.\}$$

(* вектор неизвестных, полученный встроеннным решателем, и соответствующие невязки *)

$$\{-0.791204,\ 0.447048,\ -0.686378,\ 2.09718,\ 1.53159\}$$

$$\{-8.88178 \times 10^{-16},\ -2.22045 \times 10^{-15},\ 0.,\ -8.88178 \times 10^{-16},\ 0.\}$$

$$\{0.,\ -8.88178 \times 10^{-16},\ -8.88178 \times 10^{-16},\ -1.77636 \times 10^{-15},\ 0.\}$$

(* сравнение двух найденнных решений *)

$$\{3.33067 \times 10^{-16},\ -1.66533 \times 10^{-16},\ 0.,\ -4.44089 \times 10^{-16},\ 0.\}$$

2. Степенной метод

In[2]:=
```
Clear["Global`*"]
mA={{3.819,0.572,-0.354,-0.606,0.856}
,{-0.440,4.320,0.509,-1.909,0.280}
,{-4.017,0.746,4.650,-1.154,-2.037}
,{-0.209,-0.440,0.207,5.110,-1.163}
,{0.280,-0.014,-0.137,-0.070,4.571}};
y0=RandomReal[{-1,1},5];
kmax=100;
Do[y1=mA.y0;
λ1=mA.y1.y1/y1.y1;
y0=y1/Max[Abs[y1]];
,{k,1,kmax,1}
]
λ1
mA.y1-λ1 y1
(* второе собственное значение *)
λ2=(mA.mA.y0-λ1 mA.y0)/(mA.y0-λ1 y0)
Mean[λ2]
y2=mA.y0-λ1 y0
mA.y2-λ2 y2
(* наименьшее собственное значение,
      знакоопределенная матрица *)
mB=mA-λ1 IdentityMatrix[5];
y0=RandomReal[{-1,1},5];
Do[
y1=mB.y0;
μ1=mB.y1.y1/y1.y1;
y0=y1/Max[Abs[y1]];
,{k,1,kmax,1}
]
μ1
mB.y1-μ1 y1
λn=λ1+μ1
mA.y1-λn y1
(* наименьшее собственное значение,
обратная матрица *)
mInvA=Inverse[mA];
```

```
y0=RandomReal[{-1,1},5];
Do[
y1=mInvA.y0;
μ1=mInvA.y1.y1/y1.y1;
y0=y1/Max[Abs[y1]];
,{k,1,kmax,1}]
λn=1/μ1
mA.y1-λn y1
y1
MatrixForm[mA]
N[Eigenvalues[mA]]
```

(* максимальное по модулю собственное значение *)

```
Out[6]=    6.106191570961087`
```

(* второе по модулю собственное значение *)

```
Out[9]=    5.37787402037977`
```

(* наименьшее по модулю собственное значение *)

```
Out[17]=   2.7444423474839965`
```

```
Out[22]=   2.7444423474839965`
```

(* исходная матрица *)

```
Out[25]//MatrixForm=
             3.819   0.572 -0.354 -0.606   0.856
            -0.44    4.32    0.509 -1.909   0.28
            -4.017   0.746   4.65  -1.154 -2.037
            -0.209  -0.44    0.207  5.11  -1.163
             0.28   -0.014 -0.137 -0.07    4.571
```

(* собственные значения, полученные встроенными функциями *)

```
Out[26]=  {6.10619,5.34389,4.19854,4.07693,2.74444}
```

3. Дихотомия

In[162]:=
```
Clear["Global`*"]
left=a=-2.0; (* левая граница *)
right=b=3.0; (* правая граница *)
ε=10^-5; (* требуемая точность *)
(* исследуемая функция *)
f[x_]=E^(-x) Sin[x+8]+Cos[x/3-7]
found=False; (* условие останова *)
(* максимальное количество итераций *)
nmax=40;
n=0; (* счетчик итераций *)
While[Not[found]&&n<nmax,
n=n+1;
c=(a+b)/2;
If[
Sign[f[a]]*Sign[f[c]]<0
,b=c
,If[
Sign[f[c]]*Sign[f[b]]<0
,a=c
,found=True (* нашли точное значение *)
]
];
If[((b-a)<ε)||(Abs[f[c]]<ε)
,found=True
(* требуемая точность достигнута *)];]
{a,b,c,f[c],n}
 (* визуализируем результат *)
Plot[f[x],{x,left,right}
,Epilog→
{Red,PointSize[Large],Point[{c,f[c]}]}]
```

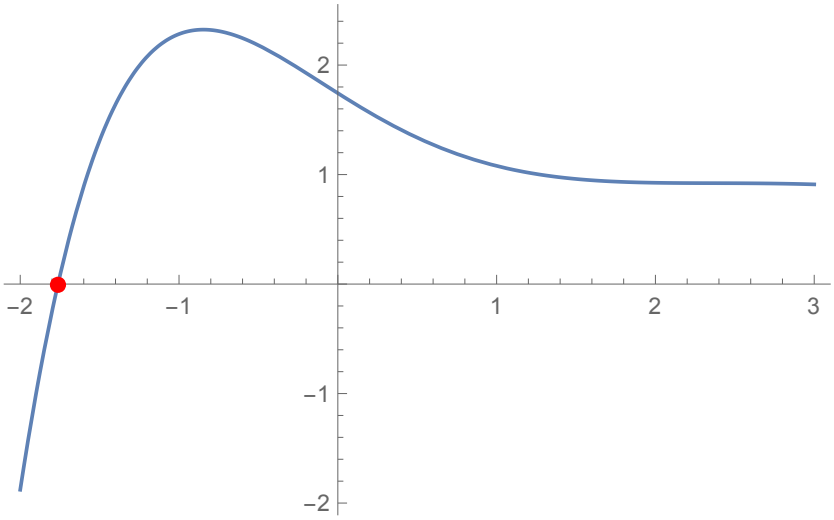
(* исследуемая функция *)

$$\text{Cos}\left[7-\frac{x}{3}\right]+e^{-x}\,\text{Sin}[8+x]$$

(* a, b, c, невязка, количество итераций *)

$$\{-1.76207,\ -1.76206,\ -1.76207,\ -7.45318\times10^{-6},\ 19\}$$

(* визуализация найденного решения *)



4. Метод секущих

In[190]:=
```
Clear["Global`*"]
a=0.15;
b=1.5;
f[x_]=x^6-Exp[-x^2]Sin[3x]
nmax=100;
ε=10^-2;
n=0;
found=False;
x0=a;
While[Not[found]&&n≤nmax,
n=n+1;
x1=x0-f[x0](b-x0)/(f[b]-f[x0]);
If[Abs[f[x1]]<ε,found=True,x0=x1]
]
{x1,f[x1],n}
Plot[{f[x]},{x,a,b},PlotRange→{-1,1}
,Epilog→
```

```
{Red,PointSize[Large],Point[{x1,f[x1]}]}]
```

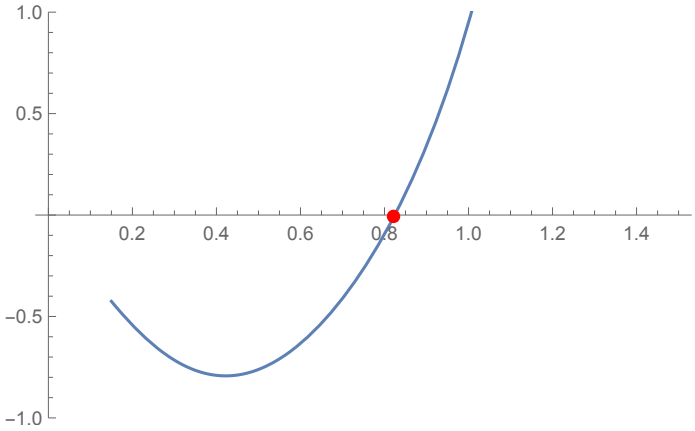
(* исследуемая функция *)

$$x^6 - e^{-x^2}\,\mathrm{Sin}[3\,x]$$

(* найденное решение, невязка, количество итераций *)

$$\{0.822242,\ -0.00874601,\ 25\}$$

(* визуализация найденного решения *)



5. Метод спуска

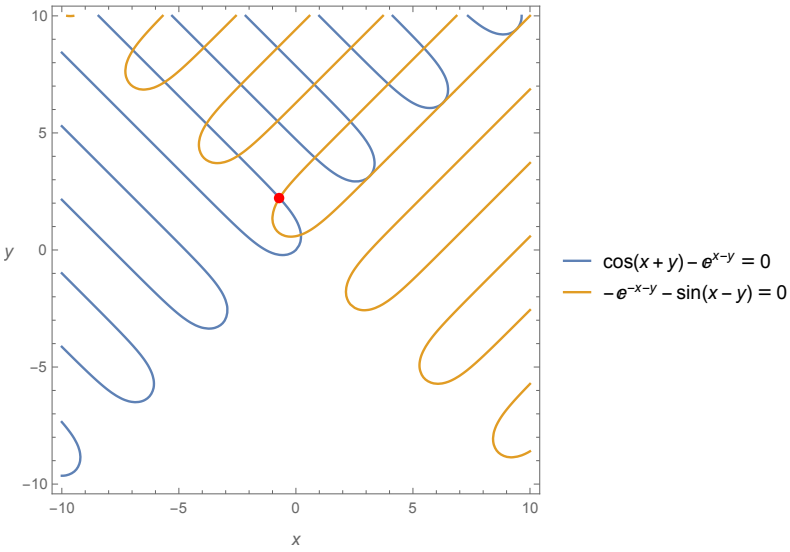
In[191]:=
```
ClearAll["Global`*"]
f[{x_,y_}]=Cos[x+y]-E^(x-y);
g[{x_,y_}]=Sin[y-x]-E^(-x-y);
Ψ[{x_,y_}]=f[{x,y}]^2+g[{x,y}]^2;
gr[{x_,y_}]=Grad[Ψ[{x,y}],{x,y}];
nmax=100;
ε=10^-3;
x0={-1.0,2.0};
found=False;
n=0;
While[Not[found]&&n≤nmax,
```

```
n=n+1;
α=1.0;
x1=x0-α gr[x0];
While[Ψ[x1]≥Ψ[x0],
α=α/2;
x1=x0-α gr[x0];];
If[√Ψ[x1]≤ε||Norm[gr[x1]]<ε
,found=True,x0=x1];]
{n,x1,Ψ[x1],f[x1],f[x0],g[x1],g[x0]}
ContourPlot[{f[{x,y}]==0,g[{x,y}]==0}
,{x,-10,10},{y,-10,10}
,Epilog→{Red,PointSize[Large],Point[x1]}
,PlotLegends→"Expressions"
,FrameLabel→
(Style[#,12,Italic]&/@{"x","y"})
,RotateLabel→False
]
Show[
Plot3D[{f[{x,y}],g[{x,y}]}
,{x,-2.5,2.5},{y,-2.5,2.5}
,PlotRange→{-2.5,2.5}
,PlotLegends→{f[{x,y}],g[{x,y}]}
,AxesLabel→
(Style[#,12,Italic]&/@{"x","y","z"})],
Graphics3D[{
    {Red,Sphere[Flatten[{x1,0}],0.1]}
,{Darker@Green,Opacity[0.7],
InfinitePlane[{0,0,0},{{1,0,0},{0,1,0}}]]}
}]]
```

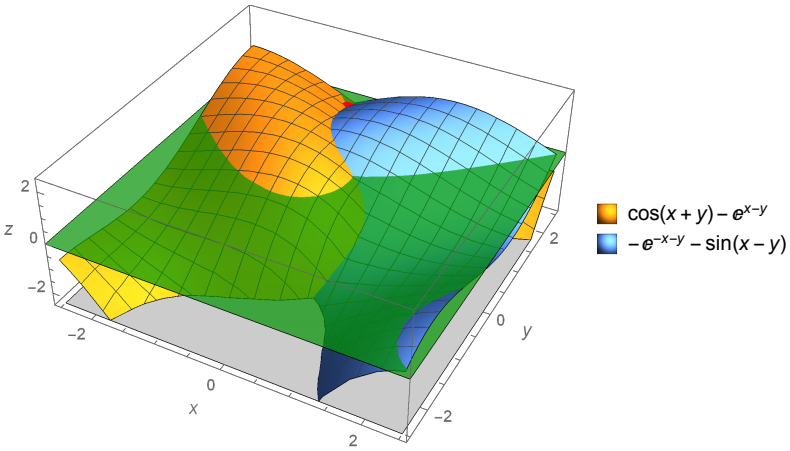
(* количество итераций, найденное решение,
невязки для функций $\Psi(x,y)$, $f(x,y)$, $g(x,y)$ *)

```
Out[217]=  {6,{-0.70232,2.21872},7.77769,
        0.000490463,-0.00270272,-0.000732949,
        0.00180705}
```

(* визуализация решения на плоскости *)



(* визуализация решения в пространстве *)



6. Интерполяционный многочлен Ньютона

In[237]:=
```
f[t_]=Sin[t]
x={0,π/6,π/4,π/2}
y=f/@x
n=Length[x]
f[x_List,0,i_Integer]:=f[x[[i]]]
f[x_List,k_Integer,i_Integer]:=
f[x,k,i]=(f[x,k-1,i+1]-f[x,k-1,i])/
(x[[i+k]]-x[[i]])
Newton[t_]=Sum[f[x,i,n-i]*
Product[(t-x[[j]])
,{j,n,n-i+1,-1}]
,{i,0,n-1}]//Expand
Expand[InterpolatingPolynomial[
Transpose[{x,y}],t]]==Newton[t]
```

(* интерполируемая функция *)

Out[237]= `Sin[t]`

(* узлы интерполяции *)

Out[238]= $\{0, \frac{\pi}{6}, \frac{\pi}{4}, \frac{\pi}{2}\}$

(* значения функции в узлах интерполяции *)

Out[239]= $\{0, \frac{1}{2}, \frac{1}{\sqrt{2}}, 1\}$

(* количество узлов интерполяции *)

Out[240]= 4

(* интерполяционный многочлен Ньютона *)

Out[243]= $\frac{29\ t}{2\ \pi} - \frac{8\ \sqrt{2}\ t}{\pi} - \frac{91\ t^2}{\pi^2} + \frac{64\ \sqrt{2}\ t^2}{\pi^2} + \frac{132\ t^3}{\pi^3} - \frac{96\ \sqrt{2}\ t^3}{\pi^3}$

(* сравнение со встроенным решателем *)

Out[244]= `True`

7. Интерполяционный многочлен Эрмита

In[245]:=
```
(*узлы с заданными значениями
функции и ее производных*)
dataset={{-2,3},{-1,0,-13,100},
{0,-5,0,4},{1,-6,-15,-92}};
degrees=Length/@dataset-1;
(*степень многочлена*)
totalDegree=Total@degrees-1;
(*общий вид многочлена*)
h[0][t_]=Total@Table[a[i]*t^i,
      {i,0,totalDegree,1}]
(*производные*)
Do[h[i][t_]=D[h[i-1][t],t],{i,1,Max[degrees]-1}]
(*система уравнений для определения
коэффициентов многочлена*)
system=Flatten@Table[
h[i][set[[1]]]==set[[i+2]]
,{set,dataset}(*итератор по узлам*)
,{i,0,Length[set]-2,1}
(*итератор по производным*)]
A=Array[a,totalDegree+1,0];(*список переменных*)
sol=Solve[system,A](*первый подход*)
H[t_]=h[0][t]/.First@sol(*подстановка*)
H[t]==Expand@
      InterpolatingPolynomial[dataset,t]
      (*второй подход*)
{b,mA}=CoefficientArrays[system,A]
coeffs=LinearSolve[mA,-b]
H2[t_]=coeffs.t^Range[0,totalDegree]
H2[t]==H[t]
```

(* интерполяционный многочлен Эрмита *)

Out[245]= $-2+3\,t+10\,t^3-5\,t^4-25\,t^5+17\,t^6+20\,t^7-5\,t^8-4\,t^9$

(* сравнение со встроенным решателем *)

Out[246]= True

8. Приближенные значения первой производной по формулам второго порядка точности

In[259]:=
```
Clear["Global`*"]
f[x_]:=∛(e^x)Cos[0.9 x+π/6]+Surd[x^4,3]
{xmin,xmax}={-1,1};
h=0.2;
xvalues=Range[xmin,xmax,h];
yvalues=f/@xvalues;
derivatives=1/(2h)*
ListConvolve[{1,0,-1},yvalues];
TableForm[{xvalues[[2;;-2]],derivatives}
,TableHeadings→{{"x","f'(x)"}, None}
,TableAlignments→{Center, Bottom}]
derf[x_]:=D[f[t],t]/.t->x
Plot[derf[x],{x,xmin,xmax},
Epilog→
{Red,PointSize[Large],
Point[Transpose[
{xvalues[[2;;-2]],derivatives}]]}]
```

Out[266]//TableForm=

x	-0.8	-0.6	-0.4	-0.2	0.
f'(x)	-0.85	-0.84	-0.82	-0.73	-0.16

Учебное издание

Банников Александр Сергеевич
Ким Инна Геральдовна
Латыпова Наталья Владимировна

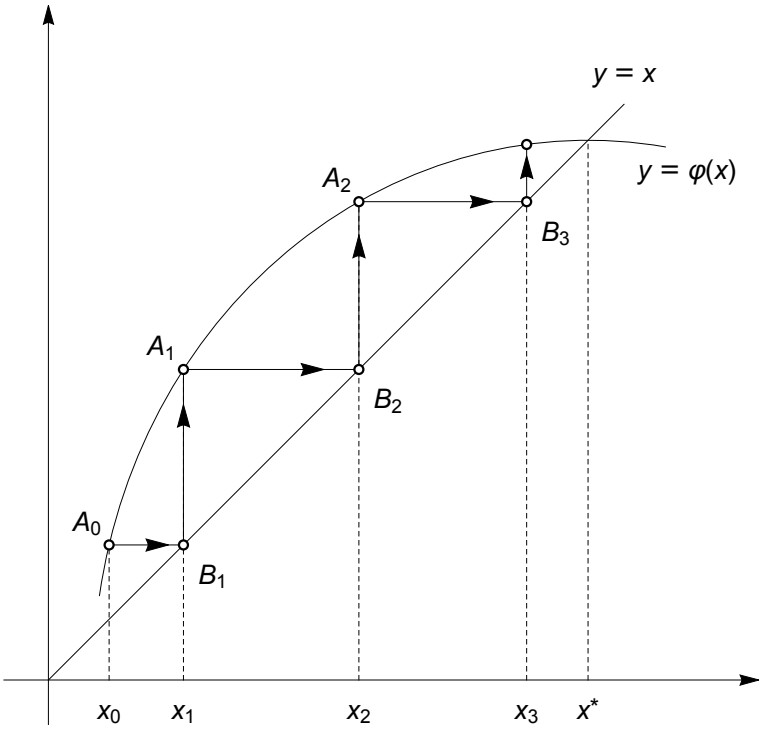
ЧИСЛЕННЫЕ МЕТОДЫ

Учебное пособие
Часть 1

Авторская редакция

Подписано в печать 15.09.18. Формат 60x84 1/16.
Усл. печ. л. 4,65. Уч.-изд.л. *,**.
Тираж 50 экз. Заказ №

Издательский центр «Удмуртский университет»
426034, г. Ижевск, ул. Университетская, д. 1, корп. 4
Тел./факс: +7(3412) 500–295 E-mail: editorial@udsu.ru
Типография Издательского центра «Удмуртский университет»
426034, г. Ижевск, ул. Университетская, д. 1, корп. 2



ISBN 978-5-4312-0643-6