Филиал Московского Государственного Университета
имени М.В. Ломоносова в г. Баку

И. С. Григорьев

Числа с плавающей точкой

Баку 2021 год

УДК 004.421

И.С. Григорьев. Числа с плавающей точкой. — 79 с.

Данное пособие является обобщением опыта преподавания основ вычислительного программирования во втором (весеннем) семестре курса “Работа на ЭВМ и программирование” для студентов математиков. Предполагается, что студент успешно освоил материал первого семестра курса “Работа на ЭВМ и программирование” (простейшие алгоритмические задачи). Представленный в пособии материал соответствует 1-му курсу механико–математического факультета МГУ и апробирован в Филиале МГУ в г. Баку.

Поскольку основной целью данного пособия является обучение программированию и освоению понятий, связанных с вычислительной погрешностью, то помимо формулировок задач, в пособии даются необходимые определения, обсуждаются идеи построения алгоритмов, для некоторых задач приводятся решения или отдельные фрагменты программ.

Для студентов, преподавателей и широкого круга специалистов, интересующихся вычислительным программированием и методикой преподавания.

Рецензент:
д.ф.–м.н., проф. Г. М. Кобельков.

Пособие представлено в свободном доступе на странице автора
`http://mech.math.msu.su/~iliagri`
Рабочая версия для второго издания от 11 мая 2022 г..

ISBN 978-9952-8354-3-4

Оглавление

Введение

Данное пособие предназначено помочь студентам при изучении программирования. Предполагается, что студент успешно освоил решение простейших алгоритмических задач.

Вычисления с плавающей точкой и реализация вычислительных алгоритмов представляет собой важный и сложный раздел программирования; идеологически он принципиально отличается от обычного целочисленного программирования. Главная его особенность состоит в том, что вычисления с плавающей точкой производятся не точно, вычислительная ошибка накапливается. Многие вычислительные алгоритмы являются итерационными: искомый результат вычисляется как некоторое приближение, уточняемое от итерации к итерации, пока оно не удовлетворит заданному критерию точности. Основную проблему вычислительного программирования составляет теоретический и практический контроль за величиной вычислительной погрешности. Изложение теоретических основ и принципов построения эффективных вычислительных алгоритмов требует знания многих математических дисциплин и не входит в задачи данного пособия. Студентам предлагается реализовать некоторые вычислительные алгоритмы по их описанию и проверить их работоспособность на тестовых задачах.

Представленный в пособии материал соответствует [4, Гл. 2].

Издание второе. Исправлены замеченные опечатки, изменено и дополнено изложение ряда разделов.

Замечание. Часть приведённых в пособии результатов работы программ компиляторозависима (результат зависит от используемой операционной системы и компилятора). Ожидается, что, при отработке заданий с использованием компилятора `gcc` на Linux-компьютерах полученные числа будут в точности соответствовать приведённым в пособии.

Внимание! Если в программе используются математические функции, то компилятор `gcc` требует использования ключа `-lm` (подключение математической библиотеки).

Глава 1.

Представление чисел

Основным типом чисел с плавающей точкой в языке C/C++ является тип `double`. Существует ещё тип `float`, но он считается устаревшим. При действиях с числами типа `float` они преобразуются к числам типа `double`, вычисленное значение типа `double` при сохранении округляется до подходящего числа типа `float`. При написании книги предполагается, что тип `float` хранится в памяти с использованием 4 байт (32 бита), тип `double` — 8 байт (64 бита). Проверить это возможно с использованием функции `sizeof`:

```
#include <stdio.h>

int main(void)
{
printf( "Size of float=%lu\n",sizeof(float) );
printf( "Size of double=%lu\n",sizeof(double) );
return 0;
}
```

Тип `long double` стандартным может считаться или не считаться и использовать 10 байт (80 бит) или 16 байт (128 бит). Ранее тип `long double` использовал 10 байт (80 бит) и представлял собой тип данных, используемых для вычисления "на процессоре". При этом число типа `double` преобразовывалось к числу типа `long double`, вычисления проводились с десятибайтовым представлением, по окончании десятибайтовое представление округлялось до восьмибайтового. Такое преобразование использовалось для вычислений, более быстрых по сравнению с шестнадцатибайтными, но в то же время достаточно точных.

Если `long double` использует 10-байтовое представление, можно попробовать использовать тип `__float128` (с двумя подчёркиваниями). Использование такого типа в настоящее время традиционно конфликтует с библиотеками ввода-вывода. В качестве простейшей рекомендации, в случае если вывод 34-значного числа не принципиален, можно перед выводом округ-

лить значение до стандартного типа `double`.

Заметим, что возвращаемое функцией `sizeof(long double)` значение 12 байт (96 бит) или 16 байт (128 бит) не обозначает использования именно 12 или 16 байт полностью для хранения числа; это может означать хранение 10-байтного числа в памяти с выравниванием адресации. В этом случае старшие 2 (из 12) или 6 (из 16) байт не используются, что бы ни находилось в этих байтах, на значении хранимого числа это не отражается.

Задание 1.1. *Проверьте сколько памяти выделяет используемый Вами компилятор для хранения чисел типа* `float`, `double`, `long double`. *Допустим ли тип данных* `__float128`*?*

В первой главе рассматривается представление чисел с плавающей точкой и основные понятия, связанные с этим представлением: **шаг сетки**, **машинный эпсилон**, эффект потери точности.

Не так давно появившаяся posit–арифметика в настоящее время является предметом дискуссий и в пособии не рассматривается.

1.1. Хранение чисел с плавающей точкой

Числа с плавающей точкой (floating point) — формат представления действительных чисел, в котором число хранится в форме мантиссы и показателя степени. При этом число с плавающей точкой имеет фиксированную относительную точность и изменяющуюся абсолютную. Представление числа в форме с плавающей точкой может рассматриваться как компьютерная реализация экспоненциальной записи чисел.

Число с плавающей точкой состоит из:

- Знака числа (знака мантиссы).
- Мантиссы, выражающей значение числа без учета порядка.
- Порядка (целого числа со знаком или без знака).

Хранение осуществляется в нормализованной форме, исключающей неоднозначную запись чисел. Зарезервировано некоторое число специальных значений: ноль, `NaN` (Not a Number, не

число) и $\pm$`INF` (Infinity, бесконечность), получающихся в результате операций деления на ноль или при превышении числового диапазона. Также специализированными допустимо считать денормализованные числа.

Для хранения чисел с плавающей точкой разработаны специальные стандарты (см. стандарт `IEEE 754`). Подробнее об этом можно посмотреть [11, § 17].

Чтобы разобраться с битовым представлением чисел с плавающей точкой, воспользуемся функцией

```
void printBits(void *a)
{
  int i; unsigned long x;
  x=*(unsigned long *)a;
  for( i=63; i>=0; i--) {
    if((x & ((unsigned long)1 << i)) != 0) printf("1");
    else printf("0");
    if( i==63 ) printf(" ");
    if( i==62 ) printf(" ");
    if( i==52 ) printf(" ");
  }
}
```

Следует обратить внимание, что аргументом этой функции является неспецифицированный адрес (`void*`). При этом учитывается, что любой специфицированный адрес бесконфликтно преобразуется к такому неспецифицированному адресу. Значение, лежащее по полученному адресу, рассматривается как восьмибитное беззнаковое число. Предполагается, что размеры беззнакового длинного целого и числа с плавающей точкой двойной точности совпадают и равны 8 байт. Разумеется, если в используемом Вами компиляторе эти размеры другие

$$\texttt{sizeof(double)} \neq 8 \text{ и/или } \texttt{sizeof(unsigned long)} \neq 8,$$

то функцию следует переписать.

Полезное упражнение — самостоятельное определение "какой бит за что отвечает" в числах типа `double`.

Задание 1.2. *Проверить, чем отличаются битовые представления чисел* x, $-x$, $x/2$, $x/4$, $x/8$, $x+1$, $x+2$, ...

Задание 1.3. *Написать функцию печати битового представления для* `long double`, `__float128`.

1.2. Минимальное положительное машинно–представимое число, шаг сетки и машинный эпсилон

1.2.1. Минимальное положительное число

С использованием функции `printBits` предыдущего пункта можно увидеть, как изменяется число — степень двойки. Последнее показанное на экране является минимальным положительным (денормализованным) машинно–представимым числом.

```
int main(void)
{
  double a=16.0; int i=4;
    do
    {
    printf("%d: %le=\t",i,a);
    printBits(&a); printf("\n");
    a/=2.0; i--;
    }
    while( a>0 );
  return 0;
}
```

1.2.2. Шаг сетки и машинный эпсилон

Шагом сетки называется разность между двумя соседними машинно–представимыми числами, машинным эпсилон — разность между минимальным машинно–представимым числом большим единицы и единицей:

$$\varepsilon(x) = \min_{y>x}(y - x), \quad \varepsilon = \min_{y>1}(y - 1).$$

```
int main(void)
{
  double x=1.0, b; int i=0;
    do
    {
    printf("%d: %le\n",i,x);
    printBits(&x); printf("\n");
```

```
    b=1+x;
    printBits(&b); printf("\n");
    x/=2.0; i--;
    }
    while( 1.0+x>1.0 );
    b=1+x;
    printBits(&b); printf("\n");
  return 0;
}
```

В этой программе кроме степени двойки, её приближённого значения в десятичной записи и битового представления шага сетки выводится ещё и битовое представление суммы уменьшаемого числа `x` с единицей, а последней строкой приводится битовое представление числа 1. Следует обратить внимание, что последнее представленное число `2.220446e-16` действительно является разностью минимального из чисел больших единицы и единицы.

Задание 1.4. *Написать программу вычисления машинного эпсилон для чисел типа* `float`, `long double`, `__float128`. *Ожидаемые ответы:*

Тип	Степень двойки	Машинный эпсилон
`float`	-23	`1.19209e-07`
`double`	-52	`2.220446e-16`
`long double`	-63	`1.0842e-19`
`__float128`	-112	`1.92593e-34`

Задание 1.5. *Написать программу вычисления шага сетки и построить график зависимости шага сетки $\varepsilon(x)$ от значения x.*

Ожидаемый результат приведён на рис. 1.1.

Задание 1.6. *Пояснить полученный результат.*

Задание 1.7. *Пояснить оценку шага сетки для машинно–представимых чисел (кроме денормализованных):*

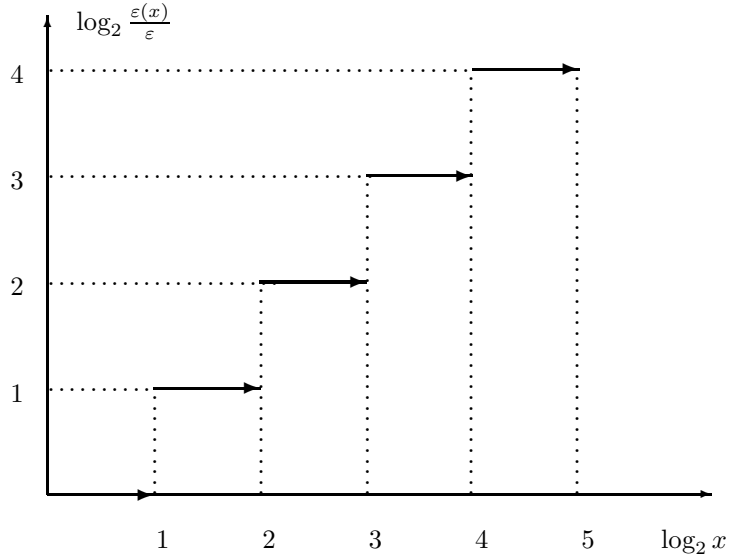
$$\frac{\varepsilon \cdot x}{2} \leqslant \varepsilon(x) \leqslant \varepsilon \cdot x.$$

Задание 1.8. *Когда остановится цикл?*

```
double x=1;
while( x+1>x ) x++;
```

Чему равен шаг сетки для найденного `x`*?*

Рис. 1.1 Шаг сетки $\varepsilon(x)$



Глава 2.

Вычислительная погрешность

Пусть есть некоторая величина x, и пусть в результате вычислений она определена приближённо. Обозначим найденную величину $\tilde{x}$.

Различают два вида погрешности: абсолютную и относительную. Абсолютная погрешность некоторой величины равна модулю разности между её приближённым и точным значениями:

$$\Delta x = |\tilde{x} - x|, \tag{2.1}$$

Если величина x отделена от нуля

$$|x| \geqslant q > 0,$$

то может быть определена относительная погрешность:

$$\delta x = \left| \frac{\tilde{x}}{x} - 1 \right|. \tag{2.2}$$

Кроме того определяются погрешность и относительная погрешность "со знаком":

$$\Delta x = \tilde{x} - x. \tag{2.3}$$

$$\delta x = \frac{\tilde{x}}{x} - 1. \tag{2.4}$$

В дальнейшем в пособии они будут использоваться в таблицах при описании главного члена погрешности.

Если приближённое число получено в результате округления, то абсолютная погрешность оценивается по абсолютной величине половиной шага сетки, и если искомая величина находится в "нормальном" диапазоне (т.е. не входит в диапазон денормализованых чисел), то относительная погрешность по абсолютной величине оценивается половиной машинного эпсилон.

Пусть имеется некоторый алгоритм вычисления искомой величины y по начальным данным x:

$$y = f(x).$$

Пусть начальные данные заданы с абсолютной погрешностью Δx или относительной погрешностью δx, и пусть Δy и δy — абсолютная и относительная погрешность вычисления результата.

Определение 2.1. *Алгоритм вычисления называется устойчивым по начальным данным, если малая погрешность начальных данных Δx (или δx) приводит к малой погрешности результата Δy (или δy).*

Отсутствие устойчивости означает, что незначительные погрешности начальных данных могут привести к значительным погрешностям результата или вовсе неверному результату. О подобных неустойчивых задачах также говорят, что они *чувствительны* к погрешностям исходных данных.

Определение 2.2. *Задача называется поставленной корректно, или корректной, если для любых начальных данных из некоторого класса её решение существует, единственно и устойчиво по начальным данным.*

2.1. Прямой анализ вычислительной погрешности

В этом разделе рассматривается оценка влияния вычислительной погрешности на результат проведённых вычислений. Для абсолютной и относительной погрешности в этом пункте используются определения (2.1), (2.2).

При сложении или вычитании чисел оценки их абсолютных погрешностей складываются:

$$\Delta(a \pm b) \leqslant \Delta a + \Delta b + \varepsilon(a \pm b)/2. \tag{2.5}$$

Последнее слагаемое в (2.5) соответствует округлению результата вычисления до подходящего машинно–представимого числа.

При умножении или делении чисел друг на друга складываются их относительные погрешности:

$$\begin{pmatrix} \delta(a \cdot b) \\ \delta(a/b) \end{pmatrix} \leqslant \delta a + \delta b + \varepsilon/2. \tag{2.6}$$

В случае вычисления функции погрешность её вычисления в зависимости от погрешности её входных данных в упрощенном

виде оценивается соотношением

$$\Delta f(x_1, x_2, \ldots) \approx \left|\frac{\partial f}{\partial x_1}\right| \Delta x_1 + \left|\frac{\partial f}{\partial x_2}\right| \Delta x_2 + \ldots$$

2.2. Эффект потери точности

Итак, в результате выполнения действий с числами с плавающей точкой, результат может оказаться (и обычно оказывается) неточным. В лучшем случае такая неточность связана с округлением результата до подходящего машинно-представимого числа. В этом лучшем случае относительная ошибка ограничена машинным эпсилон.

Однако существуют ситуации, когда использование арифметики с конечным числом знаков и округлением приводит к катастрофическим результатам.

Продемонстрируем этот эффект на примере чисел в десятичном представлении. Пусть значения округляются до 9 значащих цифр: $\pi \approx 3.14159265$, $\sqrt{10} \approx 3.16227766$. Разность этих чисел содержит меньшее число правильных знаков: $\sqrt{10} - \pi = 0.02068501 = 2.06850100 \cdot 10^{-2}$ (вместо $2.06850066 \cdot 10^{-2}$). Это произошло за счёт совпадения первых двух значащих цифр; после вычитания они сократились (0.02068501), при приведении к нормальному виду $2.06850100 \cdot 10^{-2}$ две последние цифры были заполнены нулями. Таким образом в приведённом примере оказалось потеряно два знака (две значащие цифры).

2.2.1. Суммирование рядов

Задание 2.1. *Вычислить значение функции как сумму конечного числа слагаемых ряда Тейлора:*

$$\begin{aligned} \exp x &\approx \textstyle\sum_{k=0}^{N} \dfrac{x^k}{k!}, \\ \sin x &\approx \textstyle\sum_{k=0}^{N} (-1)^k \dfrac{x^{2k+1}}{(2k+1)!}. \end{aligned} \tag{2.7}$$

Так как ряды (2.7) для всех действительных x сходятся абсолютно, то вычисление суммы достаточного числа членов ряда

позволит получить его приближённое значение с любой наперёд заданной точностью. Можно доказать, что, начиная с некоторого слагаемого, ошибка, происходящая от ограничения ряда конечным числом членов, не превосходит по модулю последнего прибавленного члена ряда. Однако при выполнении арифметических действий приближённо, с конечным числом значащих цифр, результат суммирования может отличаться от точного очень значительно.

Для экспоненты вычисление ряда может быть реализовано в виде функции:

```
double my_exp(double x)
{
  double sum=1.0, h, eps=1.e-16; int i=2;
    h=x;
    do
    {
    sum+=h;
    h*=x/i;
    i++;
    }
    while( fabs(h)>eps );
  return sum;
}
```

Комментарии по реализации. Следует обратить внимание, что в коде не используется вычисление и сохранение значения факториала в явном виде. Вычисление в программе значение факториала и использование его значения в явном виде — это признак неправильно работающей программы.

Результаты работы этой функции приведены в таблице 2.1. Отметим, что для положительных значений сумма ряда вычисляется близкой к "правильному" значению. Абсолютная погрешность может казаться большой, но она соответствует одному или нескольким шагам сетки для вычисляемого значения.

Для приведённых в конце таблицы значений вычисленная сумма ряда оказывается неприемлемо большой и по абсолютной, и по относительной величинам.

Задание 2.2. *Пояснить полученные в таблице 2.1 результаты. (Почему можно считать, что для положительных значений сумма ряда вычисляется близкой к "правильному" значе-*

Таблица 2.1 Вычисление суммы ряда для экспоненты.

x	exp(x)	Абс. погр.	Отн. погр.
1	2.71828	4.440892e-16	2.220446e-16
10	22026.5	-3.637979e-12	-1.110223e-16
40	2.35385e+17	64.0	2.220446e-16
50	5.18471e+21	9437184.0	1.776357e-15
-1	0.367879	1.110223e-16	2.220446e-16
-10	4.53999e-05	-9.206492e-14	-2.027865e-09
-40	4.24835e-18	0.3116952	7.336845e+16
-50	1.92875e-22	2041.833	1.058630e+25
-65	5.90009e-29	-3.12089e+10	-5.289560e+38

нию? Какой вычислительный эффект происходит при вычислении ряда для отрицательных значений? Почему?)

Задание 2.3. *Определить максимальный по модулю член ряда (это слагаемое, округлённое к ближайшему машинно–представимому числу, имеет максимальную абсолютную погрешность), вычислить шаг сетки для этого слагаемого, с учётом сравнения вычисленного шага сетки и погрешности вычисления ряда, завершить пояснение полученных в таблице 2.1 результатов.*

Задание 2.4. *Учитывая, что для положительных величин написанная функция даёт разумный результат, предложить способ достаточно точного вычисления экспоненты для отрицательных величин.*

Задание 2.5. *Написать функцию суммирования ряда для синуса и проверить её работоспособность (повторить "ошибку инженерного калькулятора Microsoft" конца 90-ых годов XX-го века). Предложить "простой" способ достаточно точного вычисления синуса.*

Задание 2.6. *Вычислить значение функции* $\ln(1+x)$ *как сумму конечного числа слагаемых ряда:*

$$\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \ldots + (-1)^{k+1}\frac{x^k}{k} + \ldots \qquad (2.8)$$

Ряд сходится при $|x| < 1$.

Аналогично (2.8):

$$\ln(1-x) = -x - \frac{x^2}{2} - \frac{x^3}{3} - \frac{x^4}{4} - \ldots - \frac{x^k}{k} - \ldots$$

Пусть $y > 1$. Воспользуемся представлением:

$$1 - x = 1/y \qquad (x = 1 - 1/y > 0).$$

Тогда

$$\ln y = x + \frac{x^2}{2} + \frac{x^3}{3} + \frac{x^4}{4} + \ldots + \frac{x^k}{k} \ldots \tag{2.9}$$

Кроме того вычитая (2.9) и (2.8), получим

$$\ln\left(\frac{1-x}{1+x}\right) = \ln(1-x) - \ln(1+x) = \\ = -2\left(x + \frac{x^3}{3} + \ldots + \frac{x^{2k+1}}{2k+1} + \ldots\right).$$

Пусть $y = 2^m \cdot z,\ z \in (0.5; 1)$. Положим:

$$z = \frac{1-x}{1+x}.$$

Тогда

$$x = \frac{1-z}{1+z},$$

и потому справедливо представление:

$$\ln y = m \ln 2 - 2\left(x + \frac{x^3}{3} + \ldots + \frac{x^{2k+1}}{2k+1} + \ldots\right). \tag{2.10}$$

Задание 2.7. *Вычислить значение функции* $\ln y$ *как сумму конечного числа слагаемых рядов (2.9) и (2.10). В чём преимущества и недостатки их использования? (См. [8, Гл. 3, § 7]).*

Для того чтобы понять, как работают функции математической библиотеки `sin(x)` и `cos(x)` и почему выбраны именно такие алгоритмы, необходимо освоить темы: обусловленность базиса, ортогональные многочлены, многочлены Чебышева, рекуррентное соотношение для многочленов Чебышева и его вычислительная устойчивость, многочлен наилучшего равномерного приближения, вычисление коэффициентов разложения по чебышевскому базису. Всё это изучается на следующих курсах.

2.2.2. Решение квадратного уравнения

Требуется вычислить корни квадратного уравнения:

$$x^2 + bx + c = 0. \tag{2.11}$$

Предполагается, что величины b и c таковы, что $D > 0$, и потому у уравнения (2.11) имеются два действительных корня.

Рассмотрим функцию, реализующую вычисление корней по стандартным формулам из школьного учебника:

```
int kvadr(double b, double c, double *x1, double *x2)
{
double d;
d=b*b-4*c;
if(d<0) return 0;
*x1=(-b+sqrt(d))/2;
*x2=(-b-sqrt(d))/2;
return 1;
}
```

При $b^2 \gg |4c|$ результат вычисления одного из корней оказывается неудовлетворительным. Например, при $b = 10^9$ и $c = 1$ число $4c$ оказывается меньше шага сетки числа b^2, результат вычитания округляется к числу b^2, корень из дискриминанта d оказывается равным $|b|$ и один из корней оказывается нулевым.

Этот пример показывает, что при коэффициенте b, по абсолютной величине значительно превосходящем c, вычисление по "школьным" формулам приводит к вычитанию близких чисел, то есть возникает эффект потери точности.

Для решения задачи следует от эффекта потери точности избавиться:

$$\begin{array}{rcl} x_1 & = & -(b + \mathrm{sign}\,(b) \cdot \sqrt{D})/2, \\ x_2 & = & c/x_1. \end{array} \tag{2.12}$$

Задание 2.8. *Сравнить результаты и оценить погрешность вычисленных корней квадратного уравнения (2.11) двумя способами: по "школьным" формулам и по формулам (2.12).*

При выполнении задания 2.8 оценку погрешности провести, во–первых, на примерах задач с известным решением. При такой проверке выбираются две величины x_1, x_2, и задаются коэффициенты $b = -(x_1 + x_2)$, $c = x_1 \cdot x_2$. Погрешности нахождения

корней при этом вычисляются непосредственно по определению. Во-вторых, для оценки погрешности найденные корни подставляются в левую часть (2.11) — это называется вычислить невязку. Меньшее по модулю значение невязки соответствует лучшей точности определения корня (попробуйте пояснить почему).

2.3. Устойчивые и неустойчивые методы решения задач

Иногда при решении корректно поставленной задачи может оказаться неустойчивым метод её решения. Рассмотренные примеры из предыдущего параграфа при возникновении эффекта потери точности можно рассматривать как неустойчивые.

В качестве ещё одного примера неустойчивого алгоритма рассмотрим задачу вычисления определённого интеграла:

$$I_n = \int_0^1 x^n e^{x-1} \mathrm{d}x, \qquad n = 0, 1, 2, \ldots$$

Интегрируя по частям, получим рекуррентное соотношение:

$$I_n = \int_0^1 x^n e^{x-1} \mathrm{d}x = \\ = x^n e^{x-1}\Big|_0^1 - n \int_0^1 x^{n-1} e^{x-1} \mathrm{d}x = \\ = 1 - n I_{n-1}. \quad (2.13)$$

С учётом рекуррентного соотношения и

$$I_0 = \int_0^1 e^{x-1} \mathrm{d}x = e^{x-1}\Big|_0^1 = 1 - e^{-1}$$

или

$$I_1 = \int_0^1 x e^{x-1} \mathrm{d}x = x e^{x-1}\Big|_0^1 - \int_0^1 e^{x-1} \mathrm{d}x = e^{-1}$$

последовательно вычислим значения I_2, I_3, I_4, ...

```
#include <stdio.h>
#include <math.h>
```

Таблица 2.2 Результаты вычислений по рекуррентной формуле (2.13).

n	I_n	n	I_n	n	I_n
1	0.367879	2	0.264241	3	0.207277
4	0.170893	5	0.145533	6	0.126802
7	0.112384	8	0.100932	9	0.0916123
10	0.0838771	11	0.0773522	12	0.0717732
13	0.0669478	14	0.0627311	15	0.0590338
16	0.0554593	17	0.0571919	18	-0.0294537
19	1.55962	20	-30.1924	21	635.04
22	-13969.9	23	321308	24	-7.7114e+06

```
int main(void)
{
double a;
int n;
  a=1.0-1.0/M_E;
  for(n=1;n<50;n++)printf("%d : %lg\n",n,a=1.0-n*a);
return 0;
}
```

Результаты работы программы приведены в таблице 2.2. Уже с величины I_{18} очевидно, что значения вычислены неправильно. Действительно,

$$0 \leqslant x^n e^{x-1} \leqslant 1 \qquad \forall x \in [0,1] \quad \forall n \in \mathbb{N}$$

и потому

$$0 \leqslant I_n \leqslant 1 \quad \forall n \in \mathbb{N}.$$

Неправильный результат расчётов связан с изменением погрешности в процессе вычислений. В самом деле, величины $I_0 = 1 - e^{-1}$ и $I_1 = e^{-1}$ машинно–представимыми числами не являются. Пусть погрешность вычисления величины I_1 оценивается δ:

$$|\tilde{I}_1 - I_1| \leqslant \delta.$$

Тогда, пренебрегая погрешностью округления, погрешность вычисления величин оценивается следующим образом:

$$|\tilde{I}_2 - I_2| = |(1 - 2\tilde{I}_1) - (1 - 2I_1)| = 2|\tilde{I}_1 - I_1| \leqslant 2\delta,$$

$$|\tilde{I}_3 - I_3| = |(1 - 3\tilde{I}_2) - (1 - 3I_2)| = 3|\tilde{I}_2 - I_2| \leqslant 6\delta, \quad \ldots,$$
$$|\tilde{I}_n - I_n| = |(1 - n\tilde{I}_{n-1}) - (1 - nI_{n-1})| = n|\tilde{I}_{n-1} - I_{n-1}| \leqslant n!\delta.$$

Таким образом, предложенный рекуррентный алгоритм вычислительно неустойчив.

На этом примере удобно продемонстрировать отличие вычислительного программирования от целочисленного. Преобразуем (2.13) к виду:

$$I_{n-1} = \frac{1 - I_n}{n}.$$

Тогда, если погрешность вычисления величины I_n составляет

$$|\tilde{I}_n - I_n| \leqslant \delta_n,$$

то

$$|\tilde{I}_{n-1} - I_{n-1}| = \left|\frac{1 - \tilde{I}_n}{n} - \frac{1 - I_n}{n}\right| = \left|\frac{\tilde{I}_n - I_n}{n}\right| \leqslant \frac{\delta_n}{n},$$

$$|\tilde{I}_{n-2} - I_{n-2}| = \left|\frac{\tilde{I}_{n-1} - I_{n-1}}{n-1}\right| \leqslant \frac{\delta_n}{n(n-1)}, \ldots$$

Или, оценивая погрешность вычисления величины I_n через погрешность вычисления I_{n+20}:

$$|\tilde{I}_n - I_n| \leqslant \frac{|\tilde{I}_{n+20} - I_{n+20}|}{(n+1)(n+2)\cdots(n+20)}.$$

Обычный программист в этом месте попадает в тупик: *"Действительно, алгоритм устойчив, но величина I_{n+20} не известна! Как решать задачу — не ясно."*

Программист, разобравшийся в проблематике вычислительного программирования, в тупик не попадает. В самом деле, так как $0 \leqslant I_n \leqslant 1 \ \forall n$, то $|I_n - 0.5| \leqslant 0.5$. И потому, заменив величину I_{n+20} приближённым значением 0.5 с оценкой погрешности $\Delta = 0.5$, вычислим искомую величину I_n с погрешностью порядка шага сетки:

```
a=0.5;
for(i=n+20;i>n;i--)a=(1.0-a)/i;
printf("%d : %lg\n",i,a);
```

Глава 3.

Функция одной переменной.

В главе 3 рассматриваются простейшие вычислительные задачи для функции одной переменной — нахождение корня, нахождение минимума, приближение функции по набору её значений. При написании программ этой и следующих глав следует учесть две особенности.

Во-первых, сравнение чисел с плавающей точкой на равенство или неравенство считается “небезопасными”. В чём причина?

Допустим, некоторая величина может быть вычислена двумя разными способами. В результате этих вычислений обычно получаются разные значения, в лучшем случае отличающиеся на величину, сравнимую с шагом сетки. Это привело к формальному запрету использования операторов `==` и `!=` для таких чисел: в зависимости от настроек компилятора наличие в программе таких операций приводит к ошибке (программа не компилируется) или предупреждению (компилятор сообщает об опасном коде). За прошедшие десятилетия сформировалось правило: “если используешь числа с плавающей точкой, не сравнивай их на точное равенство, а если сравнение на точное равенство необходимо — не используй чисел с плавающей точкой”. И, как итог, в программистском сообществе в настоящее время принято считать, что человек, использующий такие сравнения (или их аналоги, “обманывающие” компилятор) **не умеет программировать**, и потому к работе программистом его допускать не следует.

Замечание для студентов. Наличие в программе сравнений на точное равенство чисел с плавающей точкой приводит к тому, что преподаватель в лучшем случае такую программу не засчитывает как правильную. Обычно — увеличивает число программ для зачёта.

Во-вторых, качество метода оценивается числом вызовов функции. Поэтому при тестировании программы используется специальный счётчик (некоторая целочисленная переменная, сначала равная нулю, и увеличивающая на единицу своё значе-

ние при каждом вызове функции). В конце работы метода значение счётчика сравнивается с ожидаемым. Превышение значения означает, что метод работает неправильно. Возможно, что запрограммирован “не тот метод”, или метод запрограммирован небрежно (например, значение, которое можно было вычислить один раз для многократного последующего использования, многократно перевычисляется).

Замечание для студентов. Разумеется, в этом случае программа преподавателем традиционно не засчитывается.

3.1. Поиск корня функции

Рассматриваются простейшие методы решения уравнений вида

$$f(x) = 0 \tag{3.1}$$

— метод деления отрезка пополам, метод хорд и метод Ньютона.

3.1.1. Метод деления отрезка пополам

Предполагается, что функция f определена на отрезке $[a, b]$ ($f : [a, b] \in \mathbb{R} \to \mathbb{R}$), непрерывна на этом отрезке ($f \in C[a, b]$) и на концах отрезка знаки её различны ($\operatorname{sign} f(a) \neq \operatorname{sign} f(b)$). В этом случае на этом отрезке существует хотя бы одна точка $c \in [a, b]$ такая, что $f(c) = 0$.

Требуется реализовать функцию с заголовком:

```
int root_half(double *x, double a, double b,
              double (*f)(double), double epsilon);
```

Возвращаемое значение функции: 0, если корень не найден (метод не применим), целое положительное число, соответствующее числу итераций метода. Первый аргумент функции `x` — указатель на полученный корень; второй и третий аргументы `a,b` — границы отрезка; четвертый `f` — указатель на функцию, задающую уравнение; последний, пятый аргумент `epsilon` — заданная точность.

Описание метода

В случаях $\operatorname{sign} f(a) = 0$, $\operatorname{sign} f(b) = 0$ итерационный метод не запускается, так как корень уже найден. Сразу возвра-

щается требуемый корень (один из корней, если их два). При $\operatorname{sign} f(a) = \operatorname{sign} f(b)$ метод не применим.

Метод деления пополам состоит в нахождении середины $c = (a + b)/2$ отрезка $[a, b]$. В этой точке вычисляется значение функции $f(c)$. Если $\operatorname{sign} f(\mathrm{c}) = 0$, то корень найден, и метод заканчивает свою работу. Иначе, в случае $\operatorname{sign} f(a) = \operatorname{sign} f(\mathrm{c})$ точка a переносится в точку c, а в случае $\operatorname{sign} f(b) = \operatorname{sign} f(\mathrm{c})$ в точку c переносится точка b. В результате этих действий длина отрезка $[a, b]$ уменьшается в два раза.

Условия остановки

Выбор условия остановки зависит от рассматриваемой задачи. Условие

$$|f(c)| < \varepsilon$$

может оказаться не слишком удачным.

Например, при $\varepsilon = 10^{-15}$ для тестовой функции

$$f(x) = (x - x_*)^{17} \tag{3.2}$$

любая точка x из окрестности корня $|x - x_*| < 0.1$ удовлетворяет условию $|f(x)| < 10^{-15}$.

Если в исходной постановке задачи, сведённой к численному решению уравнения, такой результат был удовлетворителен (найдено такое значение x, что значение функции мало), то всё в порядке. Если же требовалось значение, близкое именно к x_*, то задача при такой остановке счёта оказалась не решённой.

В то же время для тестовой функции

$$f(x) = \exp(x - \pi) - y_* \tag{3.3}$$

не для каждого y_* существуют машинно–представимые x такие, что $|f(x)| < 10^{-15}$. Например, выберем некоторое машинно–представимое число $x > 35$ и зададим

$$y_* = \frac{\exp(x) + \exp(x + \varepsilon(x))}{2}.$$

Так как

$$\exp(x + \varepsilon(x)) - \exp(x) \approx \exp(x) \cdot \varepsilon(x) \approx \exp(x) \cdot \varepsilon \cdot x > 0.06,$$

то для выбранного таким образом y_* окажется $f(x) < -0.03$ и $f(x + \varepsilon(x)) > 0.03$.

Если такая остановка счёта будет единственно возможной, то метод зациклится (не остановится из–за попадания в вечный цикл). В самом деле, достаточно быстро величины a и b примут значения двух соседних машинно–представимых чисел x и $x+\varepsilon(x)$. Величина $c = (a+b)/2$ машинно–представимым числом не является и будет округлена к числу a или b, отрезок $[a, b]$ не изменится, и на следующей итерации ситуация полностью повторится.

Аналогичная ситуация будет в случае выбора в качестве условия остановки счёта длины отрезка $[a, b]$: `fabs(b-a)<epsilon`.

Если величина `epsilon` окажется меньше шага сетки для искомого корня, то метод также зациклится.

Рассмотрим теперь условия остановки счёта по достижении малой относительной длины отрезка $[a, b]$:
`fabs(b-a)<epsilon*fabs(a+b)/2`.
При выбранной погрешности метода, бóльшей машинного эпсилон, корни, не находящиеся в малой окрестности нуля, будут находиться. Если же корень мал, могут возникнуть проблемы.

Задание 3.1. *Попробуйте привести пример такой ситуации.*

Итак, в качестве подходящего "универсального" условия можно выбрать условие истинности комбинации

```
( fabs(b-a)<epsilon*fabs(a+b)/2 || fabs(b-a)<epsilon ),
```

означающей что отрезок мал по абсолютной или относительной величине.

Ещё один возможный вариант остановки

```
( fabs(a2-a1)<epsilon && fabs(b2-b1)<epsilon ),
```

где `a1`, `b1` и `a2`, `b2` — начало и конец отрезка до и после итерации. Это условие соответствует малому изменению отрезка — либо отрезок изменился, но его длина уже мала, либо шаг сетки велик и отрезок не изменился совсем.

Задание 3.2. *Реализовать метод поиска корня делением отрезка пополам. Проверить работоспособность различных условий остановки счёта.*

3.1.2. Метод хорд

Предположения те же, что и в случае метода деления отрезка пополам: функция f определена на отрезке $[a, b]$ ($f : [a, b] \in \mathbb{R} \to \mathbb{R}$), непрерывна на этом отрезке ($f \in C[a, b]$) и на концах отрезка знаки её различны ($\operatorname{sign} f(a) \neq \operatorname{sign} f(b)$).

Требуется реализовать функцию с заголовком:

```
int root_chords(double *x, double a, double b,
                double (*f)(double), double epsilon);
```

Аналогично методу деления отрезка пополам, возвращаемое значение функции: 0, если корень не найден (метод не применим), целое положительное число, соответствующее числу итераций метода. Первый аргумент функции `x` — указатель на полученный корень; второй и третий аргументы `a,b` — границы отрезка; четвертый `f` — указатель на функцию, задающую уравнение; последний, пятый аргумент `epsilon` — заданная точность.

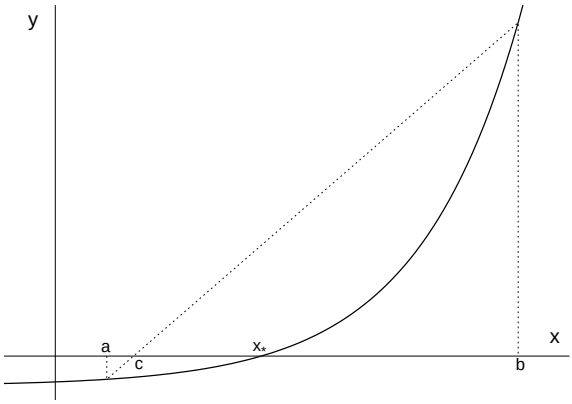


Рис. 3.1 Метод хорд для функции $f(x) = e^{x-1} - e$, $a = 0.5$, $b = 4.5$.

Описание метода

Как и для метода деления отрезка пополам, в случаях $\operatorname{sign} f(a) = 0$, $\operatorname{sign} f(b) = 0$ или $\operatorname{sign} f(a) = \operatorname{sign} f(b)$ итерационный метод не запускается. В первых двух случаях сразу воз-

вращается требуемый корень (один из корней, если их два). В последнем случае метод не применим.

Метод хорд состоит в приближении функции $y = f(x)$ прямой, принимающей в точках a и b значения $f(a)$ и $f(b)$:

$$y = \frac{f(b) - f(a)}{b - a} \cdot (x - a) + f(a).$$

Далее ищется точка c пересечения этой прямой и оси абсцисс (см. рис. 3.1):

$$\frac{f(b) - f(a)}{b - a} \cdot (\mathrm{c} - a) + f(a) = 0,$$

$$c = a - \frac{f(a)}{f(b) - f(a)}(b - a) = \frac{af(b) - bf(a)}{f(b) - f(a)}. \qquad (3.4)$$

В этой точке вычисляется значение функции $f(c)$. Далее, как и в методе деления отрезка пополам, в случае $\operatorname{sign} f(\mathrm{c}) = 0$, корень найден, и метод заканчивает свою работу. При $\operatorname{sign} f(a) = \operatorname{sign} f(\mathrm{c})$ точка a переносится в точку c, а в случае $\operatorname{sign} f(b) = \operatorname{sign} f(\mathrm{c})$ в точку c переносится точка b.

Условия остановки

Как и в случае метода деления отрезка пополам, условие малости значения функции может оказаться неэффективным. Если функция обладает свойствами выпуклости, то в процессе итераций смещаться будет только один из концов отрезка, а второй останется неизменным. Поэтому условие малости длины отрезка (как абсолютной, так и относительной) в качестве условия остановки счёта, в отличие от метода деления отрезка пополам, не эффективно.

Условие малого изменения отрезка и условие малой разности между последовательными значениями величины c в качестве условий остановки счёта могут оказаться вполне работоспособными.

Теорема 3.1. *(Сходимость метода хорд.) Пусть $f(x)$ — дважды непрерывно дифференцируемая функция на отрезке $[a, b]$, знаки функции на концах отрезка различны ($f(a) \cdot f(b) < 0$), производные $f'(x)$ и $f''(x)$ на этом отрезке непрерывны, сохраняют постоянные знаки и $f''(b)f(b) > 0$. Тогда метод хорд*

сходится и сходится со скоростью геометрической прогрессии [8, Гл. IV, § 4].

Рассмотрим пример программы:

```
#include <stdio.h>
#include <math.h>

int nf; // число обращений к функции

int sign(double x)
{
  if(x>0)return 1;
  else if(x<0)return -1;
  return 0;
}
double g(double x)
{
  nf++;
  return exp(x-M_PI)-1;
}

int root_chords(double *x, double a, double b,
                double (*f)(double), double epsilon)
{
  double fa,fb,fc,c[2];
  int i=0,ifa,ifb,ifc,k;
  c[0]=a; c[1]=b; fa=f(a); fb=f(b);

// Проверка условий запуска итераций:
  if(fabs(fa)<epsilon){*x=a; return 1;}
  if(fabs(fb)<epsilon){*x=b; return 1;}
  ifa=sign(fa); ifb=sign(fb);
  if(ifa*ifb != -1) return 0;

for(k=1; fabs(c[1]-c[0])>epsilon; k++)
  {
// последовательное сохранение промежуточной точки
    i=!i;
    c[i]=(a*fb-b*fa)/(fb-fa);
    fc=f(c[i]); ifc=sign(fc);
```

```
// если значение функции мало:
    if(fabs(fc)<epsilon){*x=c[i]; return k+1;}
// выбор подотрезка [a,c] или [c,b] :
    if(ifc*ifa==1){a=c[i]; fa=fc;}
    else {b=c[i]; fb=fc;}
  }
*x=c[i]; // возвращается последнее значение величины C
return k; // возвращается число итераций
}

int main()
{
  double x; int res;
  nf=0;
  res=root_chords(&x,-10,45,g,1.e-15);
  if(res)
  {
  printf("x=%lg\nчисло итераций=%d\n",x,res-1);
  printf("число обращений к функции=%d\n",nf);
  }
  else
  {
  printf("метод не применим\n");
  }
return 0;
}
```

Отметим, что значения функции в программе вычисляются один раз и сохраняются в локальных переменных. Последовательные значения промежуточной точки c сохраняются в массиве из двух компонент. Последнее вычисленное значение имеет индекс i, предыдущее — оставшееся с индексом $1-i$. Для изменения индекса используется логическое отрицание.

Итак, проверка для тестовой функции (3.3) и отрезка $[-10; 45]$ показала, что, несмотря на теорему, метод хорд решения не находит. В чём же дело?

Дело в том, что на выбранном отрезке экспонента очень плохо приближается прямой. Величина c очень мало отличается от величины a. За счёт округления она оказывается равной величине a, и метод заканчивает свою работу.

Выбор отрезка $[-10, 15]$ позволяет методу успешно завершиться, однако число итераций оказывается очень большим — почти четыреста тысяч, что также связано с тем, что изменяется только левый конец отрезка, и на используемых в методе отрезках функция прямой приближается плохо.

Задачи, в которых метод хорд работает хорошо, безусловно есть. В этом пособии это определение корня при решении интегрального уравнения.

3.1.3. Метод Ньютона

В отличие от предыдущих методов, оперирующих понятием "отрезок неопределённости" и состоящих в уменьшении его длины, метод Ньютона использует понятие "линейное приближение" и формирует последовательность потенциально сходящуюся к корню. Для применения метода Ньютона функция должна быть определена и дифференцируема во всех точках этой последовательности.

Описание метода

Пусть имеется какое–нибудь приближенное значение корня $x_n \approx x_*$. Положим

$$x_{n+1} = x_n + h_n,$$

где h_n считаем достаточно малой величиной для применимости формулы Тейлора:

$$f(x_n + h_n) \approx f(x_n) + h_n \cdot f'(x_n). \tag{3.5}$$

Остаточный член в формуле Тейлора при этом предполагается пренебрежимо малым. Целью является выбор смещения h_n так, что $f(x_n+h_n) = 0$. Величина h_n находится в результате решения линейного уравнения

$$f(x_n) + h_n \cdot f'(x_n) = 0,$$

$$h_n = -(f'(x_n))^{-1} \cdot f(x_n). \tag{3.6}$$

Внеся эту поправку, получим следующее приближение к искомому корню x_*:

$$x_{n+1} = x_n - (f'(x_n))^{-1} \cdot f(x_n). \tag{3.7}$$

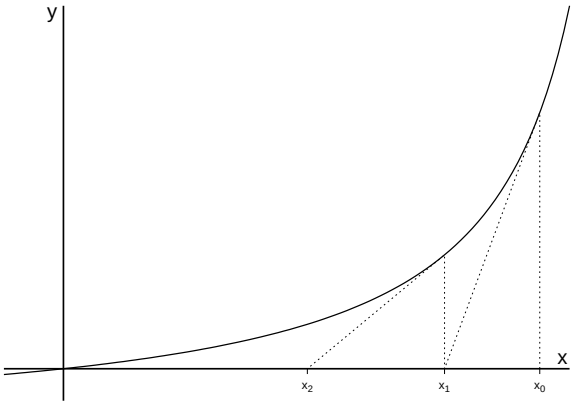


Рис. 3.2 Метод Ньютона для функции $f(x) = \frac{1}{1-x} - 1$, $x_0 = 0.8$, $x_1 = 0.64$, $x_2 = 0.4096$.

Геометрически метод Ньютона эквивалентен замене небольшой дуги кривой $y = f(x)$ касательной, проведенной в некоторой точке кривой $(x_n; f(x_n))$ (см. рис. 3.2)

Теорема 3.2. *(Сходимость метода Ньютона.)*
Пусть функция

$$f(x) = 0$$

отделена и дважды непрерывно дифференцируема на отрезке $[a, b]$, $f(a) \cdot f(b) < 0$, *причем* $f'(x)$ *и* $f''(x)$ *отличны от нуля и сохраняют определенные знаки при* $a \leqslant x \leqslant b$, *то, исходя из начального приближения* $x_0 \in [a, b]$, *удовлетворяющего неравенству* $f(x_0)f''(x_0) > 0$, *можно вычислить методом Ньютона (3.6), (3.7) единственный корень* x_* *уравнения (3.1) с любой степенью точности [8, Гл. IV, § 5].*

Условия остановки

1) Как отмечалось для предыдущих методов, условие $|f(x_n)| < \varepsilon$ может оказаться удачным, неудачным и невыполнимым.

2) Основным “хорошим” условием остановки метода будет условие $|x_{n+1} - x_n| < \varepsilon$. Это условие гарантированно выполняется на любой сходящейся последовательности, но не гарантирует,

что корень будет найден с достаточной степенью точности.

Например, для представленной на рис. 3.2 функции $f(x) = \frac{1}{1-x} - 1$

$$|h| = \left| -\left(\frac{1}{(1-x)^2}\right)^{-1} \left(\frac{1}{1-x} - 1\right) \right| = |x(1-x)|$$

и при $x_0 \in (0.5 + \sqrt{0.25 - \varepsilon}; 1) \approx (1 - \varepsilon; 1)$

$$|h_0| < (0.5 + \sqrt{0.25 - \varepsilon}) \cdot (1 - 0.5 - \sqrt{0.25 - \varepsilon}) = 0.25 - (0.25 - \varepsilon) = \varepsilon.$$

Поэтому при выбранном начальном приближении условие остановки будет выполнено на первом же шаге.

3) Основным "плохим" условием остановки метода будет ограничение числа итераций.

Задание 3.3.

Реализовать функцию поиска корня методом Ньютона.

1. Проверить работоспособность метода Ньютона для функции $f(x) = \operatorname{arctg}(x - e)$. *Убедиться:* $\exists\Delta$ *такое, что при* $|x - e| < \Delta$ *метод Ньютона сходится, а при* $|x - e| > \Delta$ *— расходится. Вывести уравнение для определения величины* Δ *и решить его.*

2. Проверить работоспособность метода Ньютона для функции $f(x) = (x + 1) \cdot x \cdot (x - 1)$. *Выяснить, каковы "области притяжения" трёх существующих корней этой функции. (Под "областью притяжения корня" понимается множество точек таких, что метод, начиная работу с этих точек, заканчивает свою работу в требуемой окрестности искомого корня.)*

3. Проверить работоспособность метода Ньютона для функции $f(x) = 10 \operatorname{arctg}(20x^2 - 200) + \sqrt{x^2 + 1}$. *Выяснить, каковы "области притяжения" двух существующих корней этой функции. Пояснить необходимость использования условия остановки метода при превышении заданного числа итераций.*

Задание 3.4. *Для поиска значений* $\sqrt{a}$ *и* $\sqrt[3]{a}$ *могут использоваться итеративные (рекуррентные) формулы*

$$x_{n+1} = \frac{1}{2}\left(x_n + \frac{a}{x_n}\right)$$

и

$$x_{n+1} = \frac{1}{3}\left(2x_n + \frac{a}{x_n^2}\right)$$

соответственно. Поясните, как эти формулы связаны с методом Ньютона поиска корня функции. Допустимо ли выбирать в качестве начального приближения значение $x_0 = -1$? Как изменяется погрешность от итерации к итерации? Какая точность вычисления достижима?

3.1.4. Другие методы

Исследования по численному решению уравнений для функции одной переменной продолжаются в том числе и в настоящее время. По–видимому, эти исследования порождаются не прикладным, а "научно–спортивным" и "научно–методическим" интересом. В то же время проблематика решения задач для систем уравнений остаётся актуальной. С методической точки зрения в настоящее время интересны в первую очередь исследования, направленные на комбинирование методов. В зависимости от поведения функции в алгоритме выбирается по какому методу совершается очередная итерация. Или же, при использовании многопроцессорной техники, одновременно выполняется несколько итераций разными методами и по результатам их работы выбирается наилучшая.

1) **Метод секущих:**

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

Основан на линейной аппроксимации функции $f(x)$ с использованием двух предыдущих итераций x_i и x_{i-1}.

2) **Комбинированный метод** (см., например, [7, Гл. V, § 6]) состоит в выборе компромисса между приближением корня по методу деления отрезка пополам

$$c_1 = \frac{a+b}{2}$$

и по методу хорд

$$c_2 = \frac{af(b) - bf(a)}{f(b) - f(a)}.$$

Выбирая некоторым образом $0 < q < 1$, очередное приближение определяется по формуле

$$c = q \cdot c_1 + (1 - q) \cdot c_2.$$

В качестве коэффициента компромисса предлагается использовать константы $q = 0.5$, $q = 0.25$, $q = 0.2$ или величину, зависящую от итерации $q^n = 2^{-n}$.

3) Модифицированный метод хорд.

В качестве модификации[1)] метода хорд предлагается изменить расчётную формулу (3.4):

$$c = a - \frac{g(f(a))}{g(f(b)) - g(f(a))}(b - a) = \frac{a \cdot g(f(b)) - b \cdot g(f(a))}{g(f(b)) - g(f(a))}, \tag{3.8}$$

где

$$g(x) = \gamma \operatorname{arctg}\left(\frac{x}{\gamma}\right)$$

и γ — настроечный параметр метода.

Такая модификация ограничивает слишком большие по модулю значения функции, сводя итерации для таких величин к итерациям метода деления пополам, а при малых отрезках с малыми значениями функции на концах итерации сводятся к методу хорд.

4) **Модифицированный метод Ньютона.**

В качестве простой модификации метода Ньютона вместо итерационной формулы (3.7) используется

$$x_{n+1} = x_n + \gamma_n \cdot h_n, \tag{3.9}$$

где h_n определяется (3.6), а величина γ_n определяется из условия

$$|f(x_n + \gamma_n \cdot h_n)| < |f(x_n)|.$$

[1)] Авторская модификация И.С. Григорьева. Авторам в литературе не встречалась.

Модификацией Исаева–Сонина называется упрощающее предположение:

$$\gamma_n \in \{1, 2^{-1}, 2^{-2}, 2^{-3}, \ldots\}.$$

Заметим, что для монотонных функций модификация Исаева–Сонина оказывается эффективной. (Таким образом, для функции $f(x) = \operatorname{arctg}(x-e)$ модификация Исаева–Сонина метода Ньютона находит корень для любого начального приближения.)

5) **Модифицированный метод Ньютона (старая версия).**

Модификация метода Ньютона, заменяющая (3.6) на

$$h_n = -(f'(x_0))^{-1} \cdot f(x_n) \tag{3.10}$$

и рекомендуемая в старых учебниках из–за отсутствия необходимости перевычисления и обращения производной, в связи с изменением быстродействия компьютеров, по мнению авторов, не актуальна уже несколько десятилетий.

6) Метод Мюллера приближения параболой предполагает, что известны значения функции на концах отрезка $[a; b]$ и эти значения различных знаков

$$f(a) \cdot f(b) < 0.$$

Вычисляется значение функции в середине отрезка

$$c = \frac{a+b}{2}.$$

По трём значениям строится интреполяционный многочлен Лагранжа (его степень не выше 2-ой), ищется единственный корень на отрезке $[a; b]$, затем отрезок уменьшается (из трёх отрезочков оставляется один, тот, на концах которого у функции разные знаки).

7) **Метод Трауба–Островского:**

$$\begin{cases} z_n = x_n - \dfrac{f(x_n)}{f'(x_n)}, \\ x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \cdot \frac{f(x_n)-f(z_n)}{f(x_n)-2f(z_n)}. \end{cases}$$

8) **Методы Неджибова:**

$$\begin{cases} z_n = x_n - \dfrac{f(x_n)}{f'(x_n)}, \\ x_{n+1} = z_n - \dfrac{f(x_n)}{f'(x_n)} \cdot \dfrac{f(z_n)}{f(x_n) - 2\lambda f(z_n)}, \end{cases}$$

где $\lambda \in \mathbb{R}$. Метод Трауба–Островского является частным случаем метода Неджибова при $\lambda = 1$.

9) **Методы Чебышева** для решения уравнения $f(x) = 0$ основаны на разложении по формуле Тейлора функции f^{-1}, обратной к f. Они могут иметь произвольно высокий порядок точности, определяемый количеством членов разложения для f, но обычно ограничиваются небольшими порядками:

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

10) **Вариант Шарма–Гуха метода Трауба–Островского:**

$$\begin{cases} z_n = x_n - \dfrac{f(x_n)}{f'(x_n)}, \\ y_n = z_n - \dfrac{f(z_n)}{f'(x_n)} \cdot \dfrac{f(x_n)}{f(x_n) - 2f(z_n)}, \\ x_{n+1} = y_n - \dfrac{f(y_n)}{f'(x_n)} \cdot \dfrac{f(x_n) + af(z_n)}{f(x_n) + (a-2)f(z_n)}, \end{cases} \tag{3.11}$$

где параметр $a \in \mathbb{R}$.

Задание 3.5. *Реализовать методы численного решения уравнения. Проверить их работоспособность на различных тестовых примерах.*

Задание 3.6. *Реализовать для решения системы уравнений метод Ньютона, модификацию Исаева–Сонина метода Ньютона и модификацию Исаева–Сонина–Федоренко [12, Гл. 1].*

3.2. Поиск минимума функции

Рассматриваются простейшие методы поиска одного из локальных минимумов функции

$$f(x) \to \min \tag{3.12}$$

— метод деления отрезка пополам и метод золотого сечения.

Если функция $f(x)$ выпукла, то минимум на отрезке единственный.

Определение 3.1. *Функция f, определённая на отрезке $[a, b]$ ($f : [a, b] \in \mathbb{R} \to \mathbb{R}$), называется унимодальной, если существует некоторая точка x_* интервала (a, b), такая что при $x \in [a; x_*]$ функция $f(x)$ монотонно убывает, а при $x \in [x_*; b]$ $f(x)$ монотонно возрастает.*

Как следует из определения 3.1, для унимодальной функции минимум на отрезке существует, единственный и внутренний.

Определение 3.2. *Назовём интервал, на котором функция заведомо имеет минимум, интервалом неопределённости.*

Утверждение 3.1. *Пусть функция f определена на отрезке $[a, b]$*

$$f : [a, b] \in \mathbb{R} \to \mathbb{R},$$

непрерывна

$$f \in C[a, b]$$

и для некоторой внутренней точки x отрезка $[a, b]$

$$f(x) < f(a) \text{ и } f(x) < f(b). \tag{3.13}$$

В этом случае на отрезке существует хотя бы один внутренний локальный минимум. (То есть отрезок $[a, b]$ является интервалом неопределённости из определения 3.2).

3.2.1. Метод деления отрезка пополам

Требуется реализовать функцию с заголовком:

```
int min_half(double *x, double a, double b,
             double (*f)(double), double epsilon);
```

Возвращаемое значение функции — целое положительное число, соответствующее числу итераций метода. Первый аргумент функции `x` — указатель на полученный корень; второй и третий аргументы `a,b` — границы отрезка; четвертый `f` — указатель на функцию, задающую уравнение; последний, пятый аргумент `epsilon` — заданная точность.

Описание метода

Вначале интервал неопределённости совпадает со всем отрезком $[a;b]$. Целью метода является уменьшение длины интервала неопределённости до достижения заданной точности.

Итак, на отрезке $[a;b]$ выбирается 3 внутренних точки $a < x_1 < x_2 < x_3 < b$. Эти точки могут быть выбраны произвольно, например, значениями "по-умолчанию":

$$x_1 = \frac{3}{4}a + \frac{1}{4}b, \quad x_2 = \frac{1}{2}a + \frac{1}{2}b, \quad x_3 = \frac{1}{4}a + \frac{3}{4}b.$$

В этих точках вычисляются значения функции и находится минимальное значение:

$$\min\{f(x_1); f(x_2); f(x_3)\}.$$

Найденная точка минимума становится новой средней точкой, а две соседние — концами отрезка:
а) если меньше в x_1, то $a^{new} = a$, $x_2^{new} = x_1$, $b^{new} = x_2$ (см. рис. 3.3);
б) если меньше в x_2, то $a^{new} = x_1$, $x_2^{new} = x_2$, $b^{new} = x_3$ (см. рис. 3.4);
в) если меньше в x_3, то $a^{new} = x_2$, $x_2^{new} = x_3$, $b^{new} = b$ (см. рис. 3.5).
В каждом из этих случаев

$$x_1^{new} = (a^{new} + x_2^{new})/2, \qquad x_3^{new} = (b^{new} + x_2^{new})/2.$$

Жирным отрезком над осью абсцисс на рис. 3.3–3.5 показан уменьшившийся отрезок неопределённости.

В случае равенства значений выбирается любой из подходящих случаев. В результате отрезок (интервал неопределённости) уменьшается.

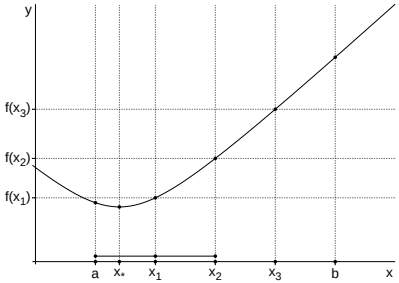


Рис. 3.3 Случай а) $f(x_1) \leqslant f(x_2)$ и $f(x_1) \leqslant f(x_3)$.

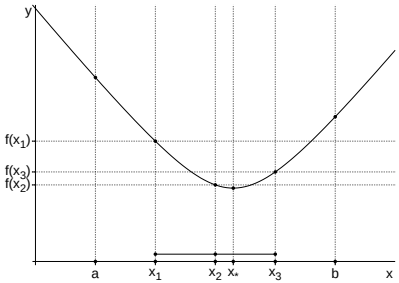


Рис. 3.4 Случай б) $f(x_2) \leqslant f(x_1)$ и $f(x_2) \leqslant f(x_3)$.

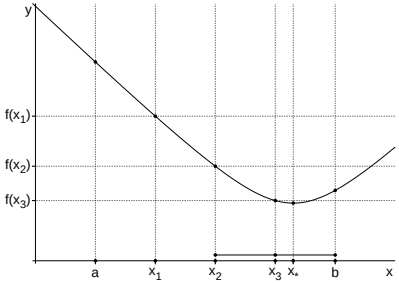


Рис. 3.5 Случай в) $f(x_3) \leqslant f(x_1)$ и $f(x_3) \leqslant f(x_2)$.

Условие остановки

Аналогично делению отрезка пополам поиска корня в качестве подходящего "универсального" условия можно выбрать условие истинности комбинации

```
( fabs(b-a)<epsilon*fabs(a+b)/2 || fabs(b-a)<epsilon ),
```

означающей, что отрезок мал по абсолютной или относительной величине.

В качестве ответа возвращается середина этого отрезка.

Заметим, что метод деления отрезка пополам может остановиться в окрестности любого из минимумов функции на отрезке, в том числе и в конце отрезка. При этом найденное значение минимума на более широком множестве минимумом не является.

Утверждение 3.2. *В случае если хотя бы одна из точек* x_1, x_2 *или* x_3 *удовлетворяет условию утверждения 3.1, то метод сойдётся к одному из внутренних локальных минимумов функции. В случае выполнения условий унимодальности или выпуклости функции при наличии внутреннего минимума на отрезке* $[a; b]$ *метод сойдётся к единственному существующему на отрезке минимуму.*

Задание 3.10. *1. Реализовать метод поиска минимума делением отрезка пополам. Проверить работоспособность различных условий остановки счёта.*
2. Подобрать пример задачи, для которой остановка только по абсолютной величине приводит к зацикливанию метода.
3. Подобрать пример задачи, для которой остановка только по относительной величине приводит к зацикливанию метода.
4. Реализовать метод поиска минимума делением отрезка пополам, использующий значение первого параметра `double *x` *функции в качестве начального значения одного из* x_i *(если в левой трети отрезка, то* x_1*; если в правой трети, то* x_3*; иначе —* x_2*). Если условие нахождения этой точки внутри отрезка (*$a < x < b$*) нарушено, или нарушено условие (3.13), то* x_1, x_2, x_3 *инициализируются значениями "по-умолчанию".*
5. Проверить работу метода для функции $f(x) = x/(1 + x^2)$ *на отрезке* $[-2; 10]$ *для разбиения по умолчанию. Для какого начального значения первого параметра метод сходится к глобальному минимуму?*
6. Обосновать утв. 3.2.

3.2.2. Метод золотого сечения

Требуется реализовать функцию с заголовком:

```
int min_gold(double *x, double a, double b,
             double (*f)(double), double epsilon);
```

Возвращаемое значение функции — целое положительное число, соответствующее числу итераций метода. Первый аргумент функции `x` — указатель на полученный корень; второй и третий аргументы `a,b` — границы отрезка; четвертый `f` — указатель на функцию, задающую уравнение; последний, пятый аргумент `epsilon` — заданная точность.

Описание метода

Пусть на отрезке $[a, b]$ некоторым образом выбираются две точки x_1 и x_2:

$$a < x_1 < x_2 < b.$$

Если $f(x_1) < f(x_2)$ (рис. 3.6), то в качестве нового интервала неопределённости выбирается отрезок $[a; x_2]$. Иначе новым интервалом неопределённости выбирается $[x_1; b]$ (рис. 3.7).

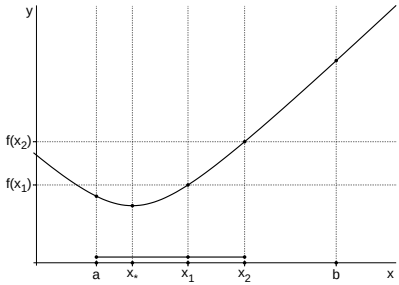


Рис. 3.6 $f(x_1) < f(x_2)$.

Как же следует выбирать точки x_1 и x_2 внутри отрезка $[a; b]$?

Поскольку неизвестно, какой из двух возможных случаев реализуется, следует минимизировать длину максимального из двух возможных интервалов неопределённости:

$$\max\{x_2 - a; b - x_1\} \to \min.$$

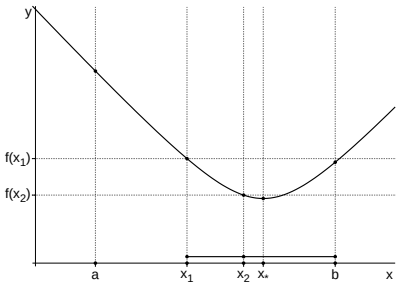


Рис. 3.7 $f(x_1) \geqslant f(x_2)$.

Если одна из точек уже выбрана, то задача может быть решена выбором симметричной относительно середины отрезка точки — при этом происходит разделение большего из двух подотрезков.

Пусть $(x_1 - a) = \varphi(b - a) = (b - x_2)$. Тогда $\varphi < 1/2$ и $(x_2 - x_1) = (1 - 2\varphi)(b - a)$. В методе золотого сечения предполагается, что после уменьшения отрезка оставшаяся внутренняя точка и некоторая новая точка зададут такую же пропорцию разделения уменьшенного интервала неопределённости, как и исходный.

Заметим, что внутренний подотрезок $[x_1; x_2]$ в этом случае после разделения не может быть бóльшим. В самом деле пусть исходный отрезок делился на три части:

$$(x_1 - a) : (x_2 - x_1) : (b - x_2) = \varphi : (1 - 2\varphi) : \varphi.$$

Если внутренний подотрезок был наибольшим $(1-2\varphi > \varphi)$, то он разделится на два отрезка длины $(1-3\varphi)|b-a|$ и $\varphi|b-a|$. Таким образом отрезок следующей итерации метода будет разделён на три части в отношении

$$\varphi : (1 - 3\varphi) : \varphi,$$

что не совпадет с предыдущей пропорцией.

Таким образом $1 - 2\varphi < \varphi$, и потому отрезок следующей итерации метода будет разделён на три части в отношении

$$(1 - 2\varphi) : (3\varphi - 1) : (1 - 2\varphi).$$

Сохранение пропорции разбиения отрезка

$$\frac{\varphi}{1 - 2\varphi} = \frac{1 - 2\varphi}{3\varphi - 1}$$

сводится к квадратному уравнению

$$\varphi^2 - 3\varphi + 1 = 0$$

и определяет требуемую величину:

$$\varphi = \frac{3 - \sqrt{5}}{2}.$$

Другой возможный корень квадратного уравнения не подходит:

$$\varphi = \frac{3 + \sqrt{5}}{2} > 1.$$

Таким образом в случае $f(x_1) < f(x_2)$ $a^{new} = a$, $x_2^{new} = x_1$, $b^{new} = x_2$ и x_1^{new} — требует определения; в противном случае $a^{new} = x_1$, $x_1^{new} = x_2$, $b^{new} = b$ и x_2^{new} — требует определения.

Итак, найденная величина — это одна из дробей Фибоначчи:

$$\varphi = \frac{3 - \sqrt{5}}{2} \approx 0.38\ldots, \qquad \psi = 1 - \varphi = \frac{\sqrt{5} - 1}{2} \approx 0.62\ldots \quad (3.14)$$

В методе золотого сечения внутренние точки выбираются по формулам

$$x_1 = a + \varphi(b - a), \qquad x_2 = a + \psi(b - a) = b - \varphi(b - a). \quad (3.15)$$

На очередной итерации метода требуется определить одну новую точку и значение функции в ней; длина отрезка на итерации уменьшается в $\frac{1+\sqrt{5}}{2}$ раз.

Условие остановки

В методе золотого сечения условие остановки аналогично условию остановки метода деления отрезка пополам предыдущего пункта.

В [13, с. 53] рекомендуется перевычислять на каждой итерации обе внутренние точки и значения в них: “при оперировании только с одной новой точкой ошибки округления при вычислении могут привести к потере интервала, содержащего минимум”. Разумеется, вычисление двух точек и значений функции в них на каждой итерации резко ухудшают эффективность процедуры минимизации, и такая реализация метода золотого сечения начинает проигрывать методу деления отрезка пополам.

В самом деле, пусть r_0 — исходная длина отрезка. Тогда после первой итерации метода деления пополам длина отрезка уменьшится вдвое:

$$r_1 = \left(\frac{1}{2}\right) r_0.$$

В результате выполнения k итераций:

$$r_k = \left(\frac{1}{2}\right)^k r_0,$$

при этом k итераций потребуют $2k$ вычислений функции. Аналогичная формула для метода золотого сечения:

$$r_k = \psi^k r_0.$$

Причины рекомендации Химмельблау — в вычислительной неустойчивости формул (3.15). Этой рекомендации следовать не стоит — необходимо заменить эти вычислительно-неустойчивые формулы устойчивым аналогом[1)]: если на k-ой итерации $f(x_{1,k}) < f(x_{2,k})$, то $a_{k+1} = a_k$, $b_{k+1} = x_{2,k}$, $x_{2,k+1} = x_{1,k}$, $x_{1,k+1} = a_{k+1} + \psi(x_{2,k+1} - a_{k+1})$; иначе $b_{k+1} = b_k$, $a_{k+1} = x_{1,k}$, $x_{1,k+1} = x_{2,k}$, $x_{2,k+1} = b_{k+1} - \psi(b_{k+1} - x_{1,k+1})$.

Главным в этих формулах является то, что положение новой точки рассчитывается не на основе длины всего отрезка, а на основе длины той части отрезка, который эта точка должна разделить. (В [5, Гл. 1, § 4.2] и [6, Ч. 1, Гл. 1, § 4.2] даётся рекомендация выбора наиболее удалённой из двух возможных точек (3.15) от оставшейся с предыдущей итерации, но такое решение кажется менее элегантным).

Вот как это может быть реализовано:

```
double x1,x2,fx1,fx2,
       phi=(3-sqrt(5))/2,psi=(sqrt(5)-1)/2;

x1=a+phi*(b-a); fx1=f(x1);
x2=a+psi*(b-a); fx2=f(x2);

while ( fabs(a-b)>epsilon &&
        fabs(a-b)>epsilon*fabs(a+b)/2 )
```

[1)] Авторская модификация И.С. Григорьева. Авторам в литературе не встречалась.

```
{
  if(fx1<fx2)
  {
    b=x2; x2=x1; fx2=fx1;
    x1=a+psi*(x2-a);
// x1=a+phi*(b-a);
    fx1=f(x1);
  }
  else
  {
    a=x1; x1=x2; fx1=fx2;
    x2=b-psi*(b-x1);
// x2=b-phi*(b-a);
    fx2=f(x2);
  }
}
```

В приведённом коде для сравнения в комментариях приведены стандартные вычислительно-неустойчивые формулы.

Вычислительно-устойчивый вариант реализации метода золотого сечения эффективнее метода деления отрезка пополам:

$$\frac{1}{2} > \left(\frac{\sqrt{5}-1}{2}\right)^2 = \left(\frac{3-\sqrt{5}}{2}\right) \approx 0.3819660\ldots$$

Задание 3.11. *1. Реализовать метод золотого сечения. Проверить его работу в тестовом случае для функции* $f(x) = |x-\pi|$. *2. Проверить устойчивость формул, изменив величины:* $\varphi = 0.38$ *и* $\psi = 0.62$.

3.2.3. Другие методы минимизации

Методы поиска минимума функции одной переменной используются в качестве составной части в методах поиска минимума в многомерном случае и двумя рассмотренными методами не исчерпываются.

Задание 3.12. *Ознакомиться с другими методами одномерной или многомерной минимизации (см., например, [1, 10, 13]) и в качестве упражнения реализовать один из них.*

Для ознакомления с проблематикой многоэкстремальных задач можно рекомендовать монографию [9].

3.3. Приближение функции на отрезке

3.3.1. Интерполяционный многочлен Лагранжа

Предполагается, что функция f определена на отрезке $[a,b]$ ($f:[a,b]\in\mathbb{R}\to\mathbb{R}$), и известны её значения в заданных точках этого отрезка:

$$a\leqslant x_1<x_2<\ldots<x_{n-1}<x_n\leqslant b,$$

$$f(x_1)=f_1,\quad f(x_2)=f_2,\quad \ldots,\quad f(x_n)=f_n.$$

Требуется построить многочлен $L_{n-1}(x)$ минимальной степени (или степени не выше $n-1$), совпадающий со значениями функции в выбранных точках:

$$L_{n-1}(x_i)=f_i,\qquad i=1,\ldots,n. \tag{3.16}$$

Пусть $L_{n-1}(x)=c_{n-1}x^{n-1}+\ldots+c_2x^2+c_1x+c_0$. Тогда система уравнений (3.16) имеет вид

$$\begin{cases} c_0+c_1x_1+c_2x_1^2+\ldots+c_{n-1}x_1^{n-1}=f_1,\\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\\ c_0+c_1x_n+c_2x_n^2+\ldots+c_{n-1}x_n^{n-1}=f_n. \end{cases} \tag{3.17}$$

Это система линейных алгебраических уравнений относительно неизвестных коэффициентов c_0, c_1, $\ldots$, c_{n-1}. Так как по предположению узлы различны, то определитель матрицы этой системы (определитель Вандермонда) отличен от нуля, и система (3.17) имеет единственное решение. Таким образом, доказаны существование и единственность интерполяционного многочлена Лагранжа.

Рассмотрим многочлены

$$\Phi_j(x)=\prod_{\substack{i=1\\ i\neq j}}^{n}\frac{x-x_i}{x_j-x_i},\qquad j=1,\ldots,n.$$

Так как

$$\Phi_j(x_i)=\delta_{ij},$$

и степени многочленов $\Phi_j(x)$ равны $(n-1)$, то интерполяционный многочлен Лагранжа может быть представлен в виде

$$L_{n-1}(x) = \sum_{j=1}^{n} f_j \Phi_j(x)$$

(так как это многочлен нужной степени, принимающий требуемые значения в заданных точках, и в силу единственности интерполяционного многочлена Лагранжа).

Другой способ построения и вычисления интерполяционного многочлена Лагранжа, основанный на *разделённых разностях* (см., например, [2,3]), остаётся заинтересованным студентам для самостоятельного изучения.

Определим многочлен

$$\omega_n(x) = \prod_{i=1}^{n} (x - x_i).$$

Теорема 3.3. *Пусть* $f \in C^n[a,b]$. *Тогда*

$$||L_{n-1}(x) - f(x)||_{C[a,b]} \leqslant \frac{||f^{(n)}||_{C[a,b]}}{n!} ||\omega_n(x)||_{C[a,b]}. \qquad (3.18)$$

Доказательство. Определим функцию

$$\varphi(x) = f(x) - L_{n-1}(x) - K\omega_n(x),$$

где K — некоторый коэффициент. Возьмём некоторую точку отрезка, не совпадающую с узлами интерполяции:

$$t \in [a,b], \qquad t \neq x_i, \qquad i = 1, \ldots, n.$$

Выберем K так, чтобы $\varphi(t) = 0$:

$$K = \frac{f(t) - L_{n-1}(t)}{\omega_n(t)}.$$

Это допустимо, так как $\omega_n(t) \neq 0$.

При таком выборе K функция $\varphi(x)$ обращается в ноль в $(n+1)$ точке $x_1, \ldots, x_n, t$. По теореме Ролля её производная $\varphi'(x)$ обращается в ноль по крайней мере в n точках. Применяя

теорему Ролля к $\varphi'(x)$, получаем, что её производная $\varphi''(x)$ обращается в ноль по крайней мере в $(n-1)$ точке. Продолжая применение теоремы Ролля, получим, что $\varphi^{(n)}(x)$ обращается в ноль по крайней мере в одной точке $\xi \in [a,b]$. Поскольку

$$\varphi^{(n)}(x) = f^{(n)}(x) - K \cdot n!,$$

то

$$0 = f^{(n)}(\xi) - K \cdot n!$$

и

$$K = \frac{f^{(n)}(\xi)}{n!}.$$

То есть, для любой выбранной точки t из отрезка $[a,b]$, не совпадающей с точками разбиения x_i, $i = 1, \ldots, n$, существует некоторая точка $\xi = \xi(t)$ из отрезка $[a,b]$, такая что

$$f(t) - L_{n-1}(t) = \frac{f^{(n)}(\xi(t))}{n!}\omega_n(t).$$

Пусть точка t — точка максимума модуля левой части (по теореме Вейерштрасса она существует). Тогда

$$||L_{n-1} - f||_{C[a,b]} = |f(t) - L_{n-1}(t)| = \\ = \frac{|f^{(n)}(\xi(t))|}{n!}|\omega_n(t)| \leqslant \frac{||f^{(n)}||_{C[a,b]}}{n!}||\omega_n(x)||_{C[a,b]},$$

что и требовалось доказать.

Вычислительный эксперимент. Пусть $f(x) = \sin(x)$. Требуется оценить погрешность приближения интерполяционным многочленом Лагранжа на отрезке $[0;1]$ при равномерном выборе узлов.

Согласно (3.18) для этой оценки необходимо вычислить или оценить норму n-й производной приближаемой функции и норму многочлена $\omega_n(x)$. Для функции $\sin(x)$ оценка нормы n-й производной (для любого n):

$$||f^{(n)}||_{C[0,1]} \leqslant 1.$$

Норму многочлена $\omega_n(x)$ можно оценить совсем грубо:

$$||\omega_n(x)||_{C[a,b]} = \max_{x\in[a;b]} \left|\prod_{i=1}^{n}(x-x_i)\right| \leqslant$$
$$\leqslant \prod_{i=1}^{n} \max_{x\in[a;b]} |x-x_i| \leqslant \prod_{i=1}^{n}(b-a) = (b-a)^n. \quad (3.19)$$

В случае симметричных относительно центра отрезка узлов возможна более аккуратная оценка. В самом деле, для квадратного трёхчлена:

$$\max_{x\in[a;b]} \left|\left(x-\frac{a+b}{2}-d\right)\left(x-\frac{a+b}{2}+d\right)\right| =$$
$$= \max_{x\in[a;b]} \left|\left(x-\frac{a+b}{2}\right)^2 - d^2\right| = \max_{x\in\{a;\frac{a+b}{2};b\}} \left|\left(x-\frac{a+b}{2}\right)^2 - d^2\right| =$$
$$= \max\left\{\frac{(b-a)^2}{4} - d^2; d^2\right\} \leqslant \frac{(b-a)^2}{4}. \quad (3.20)$$

При этом учитывается, что максимум модуля соответствует одному из экстремумов на отрезке, экстремумы достигаются в вершине параболы и на концах отрезка, в силу симметрии расположения узлов, значения на концах отрезка равны, в силу принадлежности узлов отрезку, величина d по модулю не превосходит половины длины отрезка.

Если узлов нечётное число, то в симметричном случае один из узлов парного не имеет и, следовательно, оказывается серединой отрезка. При этом

$$\max_{x\in[a;b]} \left|x-\frac{a+b}{2}\right| = \frac{b-a}{2}.$$

Объединяя эти оценки, получим:

$$||\omega_n(x)||_{C[a,b]} = \frac{(b-a)^n}{2^n}. \quad (3.21)$$

Итак, в рассматриваемом случае погрешность приближения функции $\sin(x)$ на отрезке $[0;1]$ оценивается:

$$||L_{n-1} - f||_{C[0,1]} \leqslant \frac{1}{n!}\cdot 1$$

— при использовании оценки (3.19) или

$$||L_{n-1} - f||_{C[0,1]} \leqslant \frac{1}{n!} \cdot \frac{1}{2^n}$$

— при использовании оценки (3.21).

Задание 3.13. *Написать функцию вычисления значения интерполяционного многочлена Лагранжа с заголовком:*
`double lagr(double *x, double *f, double t, int n);`
Возвращаемое значение функции — вычисленное значение интерполяционного многочлена Лагранжа. Первый аргумент функции `*x` *— указатель на массив узлов; второй аргумент* `*f` *— указатель на массив значений интерполируемой функции в узлах; третий аргумент* `t` *— точка, в которой вычисляется значение интерполяционного многочлена; четвертый аргумент* `n` *— число узлов (размер массива).*

Задание 3.14. *Провести описанный вычислительный эксперимент. В качестве оценки нормы погрешности вычислить максимум модуля разности между точным значением функции и значением многочлена Лагранжа в заданном наборе точек* $\xi_j = j/M$, $j = 0, \ldots, M$ *при "достаточно большом" M (например, при $M = 100000$). Сравнить вычисленную оценку с теоретической, выявить зависимость от числа узлов n.*

Ожидаемый результат приведён в табл. 3.1.

Таблица 3.1. Оценка погрешности интерполирования $\sin(x)$ на $[0; 1]$ по n равноотстоящим узлам.

n	Погрешность	Оценка $1/(2^n \cdot n!)$
2	5.999376e-02	1.250000e-01
3	7.196026e-03	2.083333e-02
4	2.766630e-04	2.604167e-03
5	2.660180e-05	2.604167e-04
6	7.943134e-07	2.170139e-05
7	6.093499e-08	1.550099e-06
8	1.433783e-09	9.688120e-08
9	9.049822e-11	5.382289e-09
10	1.744271e-12	2.691144e-10
11	9.540632e-14	1.223247e-11
12	9.436896e-15	5.096864e-13

n	Погрешность	Оценка $1/(2^n \cdot n!)$
13	1.398881e-14	1.960332e-14
14	3.141931e-14	7.001187e-16
17	1.414736e-13	2.144972e-20
33	4.271900e-09	1.340678e-47
65	9.052833e+00	3.286397e-111
101	3.823759e+11	4.184518e-191

Как видно из таблицы, при малых n погрешность удовлетворяет ожиданиям. Наилучший результат достигается при $n = 12$. Далее погрешность начинает расти, и результат становится катастрофическим. Этот эффект связан с увеличением числа операций и ростом вычислительной погрешности; он хорошо известен, и именно из-за него появилась необходимость в других методах интерполяции. Подробнее это будет изучаться на следующих курсах.

Задание 3.15. *В предыдущем упражнении заменить выбор равномерных узлов на чебышёвские узлы*

$$x_j = \cos\left(\frac{\pi(2j+1)}{2n}\right) \cdot \frac{b-a}{2} + \frac{a+b}{2}, \quad j = 0, 1, \ldots, n-1.$$

Написать программу, и вычислить оценку погрешности в зависимости от числа узлов n. Сравнить с предыдущим результатом.

Задание 3.16. *Для функции $y = \sin \pi x$ построить интерполяционный многочлен Лагранжа, выбрав узлы $x_0 = 0$, $x_1 = 1/6$, $x_2 = 1/2$. Оценить погрешность интерполяции на отрезке $[0; 1/2]$. ([8, Гл. 14, §12, пример 1, с. 530]).*

3.3.2. Линейный сплайн

Как и в предыдущем случае, предполагается, что функция определена на отрезке $[a, b]$ ($f : [a, b] \in \mathbb{R} \to \mathbb{R}$) и известны её значения в заданных точках отрезка:

$$a \leqslant x_1 < x_2 < \ldots < x_{n-1} < x_n \leqslant b,$$

$$f(x_1) = f_1, \quad f(x_2) = f_2, \quad \ldots, \quad f(x_n) = f_n.$$

Определение 3.3. *Линейным интерполяционным сплайном* $I_2[f](x)$ *для функции* $f(x)$ *и узлов* x_i *называется непрерывная кусочно-линейная функция с возможными точками излома* x_i, *принимающая в этих точках заданные значения* f_i.

Иными словами, на отрезках $[x_i, x_{i+1}]$, $i = 1, 2, \ldots, n-1$ линейный сплайн $I_2[f](x)$ является линейной функцией, значения которой в точках x_i, x_{i+1} совпадают со значениями f_i, f_{i+1}:

$$I_2[f](x) = f_i + \frac{f_{i+1} - f_i}{x_{i+1} - x_i}(x - x_i), \text{ при } x \in [x_i, x_{i+1}].$$

Теорема 3.4. *Пусть* $f \in C^2[a, b]$. *Тогда*

$$||f - I_2[f]||_{C[a,b]} \leqslant \frac{1}{8}||f''||_{C[a,b]}h^2, \tag{3.22}$$

где

$$h = \max_{i=1,2,\ldots,n-1}(x_{i+1} - x_i).$$

Доказательство. На каждом из отрезков $[x_i; x_{i+1}]$ линейный сплайн является интерполяционным многочленом Лагранжа первой степени. По теореме 3.18

$$||f - L_1||_{C[x_i;x_{i+1}]} \leqslant \frac{||f''||_{C[x_i;x_{i+1}]}}{2!} \max_{x\in[x_i;x_{i+1}]} |(x - x_i)(x - x_{i+1})|.$$

Функция

$$\omega_2(x) = (x - x_i)(x - x_{i+1})$$

является квадратичной параболой, принимающей нулевые значения на краях отрезка, и потому максимум её модуля на отрезке достигается в вершине:

$$\max_{x\in[x_i;x_{i+1}]} |(x - x_i)(x - x_{i+1})| =$$
$$= \left|\left(\frac{x_i + x_{i+1}}{2} - x_i\right)\left(\frac{x_i + x_{i+1}}{2} - x_{i+1}\right)\right| = \frac{(x_{i+1} - x_i)^2}{4}.$$

Следовательно, на $[x_i; x_{i+1}]$:

$$||f - I_2[f]||_{C[x_i;x_{i+1}]} \leqslant \frac{(x_{i+1} - x_i)^2}{8}||f''||_{C[x_i;x_{i+1}]},$$

откуда и получается утверждение теоремы.

Задание 3.17. *Написать функцию вычисления значения линейного сплайна в заданной точке. Провести вычислительный эксперимент, аналогичный вычислительному эксперименту предыдущего пункта. Убедиться, во-первых, что погрешность удовлетворяет (3.22). Во-вторых, убедиться, что при увеличении числа отрезков разбиения в k раз (что соответствует уменьшению h в k раз) погрешность уменьшается в k^2 раз.*

3.3.3. Кубический сплайн Эрмита

Пусть функция определена на отрезке $[a, b]$

$$f : [a, b] \in \mathbb{R} \to \mathbb{R},$$

дифференцируема, известны её значения и значения производных в заданных точках отрезка:

$$a \leqslant x_1 < x_2 < \ldots < x_{n-1} < x_n \leqslant b,$$

$$f(x_1) = f_1, \quad f(x_2) = f_2, \quad \ldots, \quad f(x_n) = f_n,$$
$$f'(x_1) = d_1, \quad f'(x_2) = d_2, \quad \ldots, \quad f'(x_n) = d_n.$$

Определение 3.4. *Кубическим сплайном Эрмита $P[f](x)$ для функции $f(x)$ и узлов x_i называется непрерывно–дифференцируемая кусочно-кубическая функция с возможными точками разрыва второй производной x_i, принимающая в этих точках заданные значения f_i и значения производной d_i.*

Иными словами, на отрезках $[x_i, x_{i+1}]$, $i = 1, 2, \ldots, n-1$ сплайн Эрмита $P[f](x)$ является кубическим многочленом, значения и производные которого в точках x_i, x_{i+1} совпадают со значениями f_i, d_i, f_{i+1}, d_{i+1}:

$P[f](x) = c_0^i + c_1^i(x - x_i) + c_2^i(x - x_i)^2 + c_3^i(x - x_i)^3$ при $x \in [x_i, x_{i+1}]$,

где

$$\begin{aligned} c_0^i &= f_i, \\ c_1^i &= d_i, \\ c_2^i &= \frac{3f_{i+1} - 3f_i - d_{i+1}h_i - 2d_ih_i}{h_i^2}, \\ c_3^i &= \frac{2f_i - 2f_{i+1} + d_{i+1}h_i + d_ih_i}{h_i^3}, \end{aligned} \tag{3.23}$$

$h_i = x_{i+1} - x_i$.

Задание 3.18. *Обосновать (3.23).*

Теорема 3.5. *Пусть* $f \in C^4[a,b]$. *Тогда*

$$||f - P[f]||_{C[a,b]} \leq \frac{1}{2^4 \cdot 4!}||f^{(4)}||_{C[a,b]}h^4 \tag{3.24}$$

где

$$h = \max_{i=1,2,\ldots,n-1}(x_{i+1} - x_i).$$

(Без доказательства. Для заинтересованных студентов доказательство см., например, [3, § 19.3]).

Задание 3.19. *Написать функцию вычисления значения Эрмитова сплайна в заданной точке. Провести вычислительный эксперимент, аналогичный вычислительным экспериментам предыдущих пунктов. Убедиться, во-первых, что погрешность удовлетворяет (3.24). Во-вторых, убедиться, что при увеличении числа отрезков разбиения в* k *раз (что соответствует уменьшению* h *в* k *раз) погрешность уменьшается в* k^4 *раз.*

3.3.4. Линейная регрессия

Линейная регрессия является частным случаем решения переопределённой системы линейных алгебраических уравнений методом наименьших квадратов и простейшим случаем регрессионной модели.

Предполагается, что между двумя величинами есть линейная зависимость:

$$y = ax + b.$$

Параметры a и b не известны и их требуется определить.

Для определения этих неизвестных были проведены измерения (наблюдения):

$$(x_i, y_i) \qquad i = 1, 2, \ldots, N. \tag{3.25}$$

Предполагается, что в этих данных присутствуют случайные ошибки:

$$y_i = ax_i + b + \varepsilon_i.$$

В методе линейной регрессии величины a и b выбираются так, чтобы сумма квадратов ошибок была минимальной:

$$\sum_{i=1}^{N} (y_i - ax_i - b)^2 \to \min_{a,b}. \tag{3.26}$$

Откуда следует:

$$\begin{array}{l} \left(\sum_{i=1}^{N} x_i\right) a + \left(\sum_{i=1}^{N} 1\right) b = \left(\sum_{i=1}^{N} y_i\right), \\ \left(\sum_{i=1}^{N} x_i^2\right) a + \left(\sum_{i=1}^{N} x_i\right) b = \left(\sum_{i=1}^{N} x_i y_i\right). \end{array} \tag{3.27}$$

Задание 3.20. *Ответить на вопросы:*
1. Существует ли минимум в рассматриваемой задаче (3.26)?
2. Если минимум существует, то единственен ли он? (Обоснование необходимо.)
3. Как связана система уравнений (3.27) с (3.26)?

Задание 3.21. *Написать программу вычисления коэффициентов a и b, входными параметрами которой являются измерения (3.25) (например, указатели на массивы* `double *x`, `double *y` *и длина массивов* `int n`*), и проверить её работоспособность. Программа должна возвращать целое число:* `0` *— ответ получен,* `1` *— ответ не найден (метод не применим).*

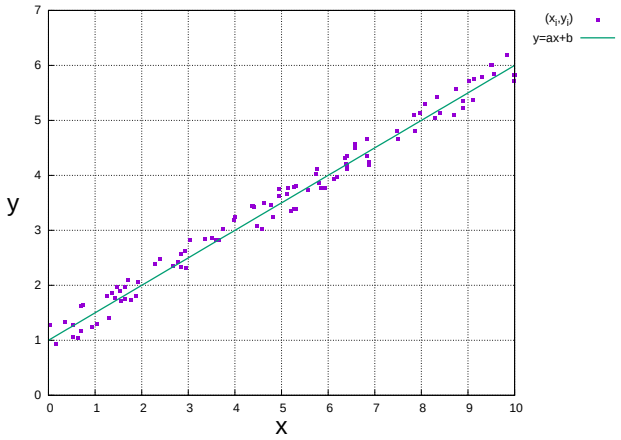


Рис. 3.8 Линейная регрессия.

В качестве тестовых данных можно использовать:

```
for(i=0;i<n;i++){
x[i]=x_min+(dx*rand())/RAND_MAX;
err=epsilon*( (2.0*rand()) /RAND_MAX-1.0);
y[i]=a*x[i]+b+err;
}
```

Предполагается, что для массивов `x[]`, `y[]` выделена память, используемые далее `x_min`, `dx` — начало и длина отрезка возможного изменения измеряемой величины x_i, `epsilon` — предельная величина ошибки измерения, `a, b` — заданные величины (вычисленные линейной регрессией значения $\tilde{a}$, $\tilde{b}$ следует сравнить с заданными и оценить погрешность).

Пример сгенерированных таким образом тестовых данных при $x_i \in [0; 10]$, `a=0.5, b=1, epsilon=0.3, n=100` представлен на рис. 3.8.

Функция

```
int rand( void );
```

генерирует случайные числа, возвращает псевдослучайное целое число в диапазоне от 0 до `RAND_MAX`, `RAND_MAX` — константа; определена в `stdlib.h` (в `C++` необходимо пространство имён `cstdlib`).

Задание 3.22. *Продемонстрировать и оценить, как изменяется погрешность при изменении* `epsilon`, `n`.

Глава 4.

Вычисление определённых интегралов

4.1. Квадратурные формулы интерполяционного типа

Для приближённого вычисления определённого интеграла используются квадратурные формулы (*квадратуры*) вида

$$I(a,b,f)=\int_a^b f(x)\mathrm{d}x \approx S_n(a,b,f)=\sum_{i=1}^{n} c_i f(x_i).$$

Постоянные c_i называются *коэффициентами* (*весами*) квадратуры, а x_i — её *узлами*.

Для функции $f(x)$ погрешность квадратурной формулы $S_n(a,b,f)$ определяется как

$$I(a,b,f)-S_n(a,b,f).$$

В последующем изложении под $R_n(a,b,f)$ понимается главный член такой погрешности $R_n(a,b,f)=C(b-a)^s$, s — порядок погрешности.

Квадратурные формулы интерполяционного типа — это квадратуры, построенные на основе замены $f(x)$ интерполяционным многочленом Лагранжа:

$$f(x)\approx L_n(x)=\sum_{i=1}^{n} f(x_i)\prod_{j\neq i}\frac{x-x_j}{x_i-x_j}.$$

В этом случае

$$S_n(a,b,f)=\int_a^b L_n(x)\mathrm{d}x=\sum_{i=1}^{n} f(x_i)\int_a^b\prod_{j\neq i}\frac{x-x_j}{x_i-x_j}\mathrm{d}x.$$

То есть

$$c_i = \int_a^b \prod_{j \neq i} \frac{x - x_j}{x_i - x_j} \mathrm{d}x.$$

Проблематика построения квадратурных формул и оценок их погрешности изучается на старших курсах, а пока воспользуемся готовыми результатами:

$$I(a,b,f) \approx (b-a) f(a), \qquad R(a,b,f) = C(b-a)^2$$

— формула левых прямоугольников;

$$I(a,b,f) \approx (b-a) f(b), \qquad R(a,b,f) = C(b-a)^2$$

— формула правых прямоугольников;

$$I(a,b,f) \approx (b-a) f\left(\frac{a+b}{2}\right), \qquad R(a,b,f) = C(b-a)^3$$

— формула центральных прямоугольников;

$$I(a,b,f) \approx \frac{(b-a)}{2}(f(a) + f(b)), \quad R(a,b,f) = C(b-a)^3$$

— формула трапеций;

$$I(a,b,f) \approx \frac{(b-a)}{2}\left(f\left(\frac{(a+b)}{2} - \frac{(b-a)}{2\sqrt{3}}\right) + \right.$$
$$\left. + f\left(\frac{(a+b)}{2} + \frac{(b-a)}{2\sqrt{3}}\right)\right), \quad R(a,b,f) = C(b-a)^5$$

— двухточечная квадратура Гаусса;

$$I(a,b,f) \approx \frac{(b-a)}{6}\left(f(a) + 4f\left(\frac{(a+b)}{2}\right) + f(b)\right),$$
$$R(a,b,f) = C(b-a)^5$$

— формула Симпсона;

$$I(a,b,f) \approx$$
$$\approx \frac{(b-a)}{8}\left(f(a) + 3f\left(\frac{(2a+b)}{3}\right) + 3f\left(\frac{(a+2b)}{3}\right) + f(b)\right),$$
$$R(a,b,f) = C(b-a)^5$$

— четырёхточечная квадратура Ньютона–Котеса;

$$I(a,b,f) \approx \frac{(b-a)}{3}\left(f\left(\frac{a+b}{2}-\frac{\sqrt{2}}{4}(b-a)\right)+f\left(\frac{a+b}{2}\right)+\right.$$
$$\left.+f\left(\frac{a+b}{2}+\frac{\sqrt{2}}{4}(b-a)\right)\right), \quad R(a,b,f)=C(b-a)^5$$

— трёхточечная квадратура Чебышева;

$$I(a,b,f) \approx \frac{(b-a)}{18}\left(5f\left(\frac{a+b}{2}-\frac{\sqrt{3}}{2\sqrt{5}}(b-a)\right)+8f\left(\frac{a+b}{2}\right)+\right.$$
$$\left.+5f\left(\frac{a+b}{2}+\frac{\sqrt{3}}{2\sqrt{5}}(b-a)\right)\right), \quad R(a,b,f)=C(b-a)^7$$

— трёхточечная квадратура Гаусса;

$$I(a,b,f) \approx \frac{(b-a)}{12}\left(f(a)+5f\left(\frac{a+b}{2}-\frac{b-a}{2\sqrt{5}}\right)+\right.$$
$$\left.+5f\left(\frac{a+b}{2}+\frac{b-a}{2\sqrt{5}}\right)+f(b)\right), \quad R(a,b,f)=C(b-a)^7$$

— четырёхточечная квадратура Маркова;

$$I(a,b,f) \approx \frac{(b-a)}{180}\left(9f(a)+49f\left(\frac{a+b}{2}-\frac{b-a}{2}\sqrt{\frac{3}{7}}\right)+\right.$$
$$\left.+64f\left(\frac{a+b}{2}\right)+49f\left(\frac{a+b}{2}+\frac{b-a}{2}\sqrt{\frac{3}{7}}\right)+9f(b)\right),$$
$$R(a,b,f)=C(b-a)^9$$

— пятиточечная квадратура Маркова.

Погрешности методов интегрирования $R(a,b,f)$ в приведённых формулах содержат коэффициент C, зависящий от функции f. Для последующих действий явный вид зависимости $C(f)$ не важен (важно предположение о существовании C), и, потому, эта зависимость не конкретизируется.

Задание 4.1. *Написать программы вычисления интеграла по приведённым квадратурным формулам. Применить их к вычислению интегралов от многочленов и сравнить с их точными значениями (по формуле Ньютона–Лейбница). Сравнить погрешность (разность между приближённым и точным значениями) и шаг сетки в окрестности точного значения. Прокомментировать полученные результаты.*

Ожидаемый результат: многочлены степени на два меньшей, чем степень в оценке погрешности $R(f)$, должны вычисляться точно. Разумеется, с учётом вычислительной погрешности точное совпадение не получается.

4.2. Правило Рунге оценки погрешности

Предполагается, что главный член погрешности имеет вид $R_n(a,b,f) = C(b-a)^s$, где C — некоторая константа, зависящая от функции f на отрезке $[a;b]$, и s — порядок погрешности. Интеграл вычисляется двумя способами: во-первых, по квадратурной формуле

$$I_1 = S_n(a,b,f);$$

главный член погрешности при этом

$$I(a,b,f) - S_n(a,b,f) = C(b-a)^s. \tag{4.1}$$

Во-вторых, отрезок $[a;b]$ разбивается на два отрезка $[a;(a+b)/2]$ и $[(a+b)/2;b]$, интеграл вычисляется как сумма квадратур на этих подотрезках

$$I_2 = S_n\left(a, \frac{a+b}{2}, f\right) + S_n\left(\frac{a+b}{2}, b, f\right).$$

Главные члены погрешности на этих подотрезках

$$I\left(a, \frac{a+b}{2}, f\right) - S_n\left(a, \frac{a+b}{2}, f\right) = \\ = C_2\left(\frac{a+b}{2} - a\right)^s = C_2\left(\frac{b-a}{2}\right)^s,$$

$$I\left(\frac{a+b}{2}, b, f\right) - S_n\left(\frac{a+b}{2}, b, f\right) = \\ = C_3\left(b - \frac{a+b}{2}\right)^s = C_3\left(\frac{b-a}{2}\right)^s.$$

Используется предположение Рунге: константы главных членов погрешности во всех этих трёх вычислениях совпадают: $C = \mathrm{C}_2 = \mathrm{C3}$. Тогда:

$$I(a, b, f) - I_2 = 2C\frac{(b-a)^s}{2^s}. \tag{4.2}$$

Вычитая (4.1) и (4.2), получаем:

$$I_2 - I_1 = C(b-a)^s(1 - 2^{1-s}).$$

Последнее соотношение позволяет оценить погрешность формул I_1 и I_2. Разумеется, в качестве расчётных значений следует использовать более точную квадратуру I_2; её погрешность оценивается величиной $(I_2 - I_1)/(2^{s-1} - 1)$.

Основной результат: имеется способ вычисления оценки главного члена погрешности.

Замечание. Этот способ не имеет строгого доказательства, более того, существуют примеры, когда вычисленная таким образом оценка погрешности не соответствует реальной. Для построения такого примера используется неотрицательная функция, обращающаяся в ноль во всех расчётных точках, интеграл от которой положителен: $f(x) = 1 - \cos(\omega x)$. Тем не менее способы такого вида эффективно используются в практических расчётах.

Замечание. Учесть полученную оценку главного члена погрешности для построения более точной квадратурной формулы возможно — это называется экстраполяцией по Ричардсону.

Замечание. Правило Рунге в настоящее время не используется для оценки погрешности. Более эффективным оказывается использовать вложенные квадратурные формулы, позволяющие по одному набору узлов одновременно вычислить и приближённое значение интеграла, и оценку погрешности.

Например, используется формула Симпсона и формула трапеций на двух подотрезках:

$$I(a, b, f) = \frac{(b-a)}{6}\left(f(a) + 4f\left(\frac{(a+b)}{2}\right) + f(b)\right) + C_5(b-a)^5,$$

$$I(a,b,f) = I(a,(a+b)/2,f) + I((a+b)/2,b,f) =$$
$$= \frac{(b-a)}{4}\left(f(a) + 2f\left(\frac{(a+b)}{2}\right) + f(b)\right) + C_3(b-a)^3.$$

Пренебрегая погрешностью формулы Симпсона:

$$C_3(b-a)^3 \approx -\frac{1}{12}\left(f(a) - 2f\left(\frac{(a+b)}{2}\right) + f(b)\right).$$

Итак, расчёт производится по более точной квадратуре Симпсона, вычисляется оценка погрешности для квадратуры трапеций, **предположительно превосходящая** по модулю неизвестную искомую погрешность и потому использующаяся в качестве оценки и для погрешности квадратуры Симпсона.

Не углубляясь в особенности этих "полуэмпирических приёмов" (см., например, [2, § 3.6]), остановимся на простейшем варианте оценки погрешности с использованием правила Рунге.

4.3. Составные квадратурные формулы

Для уменьшения погрешности отрезок интегрирования $[a;b]$ разделяется на подотрезки:

$$a = x_0 < x_1 < x_2 < \ldots < x_{m-1} < x_m = b.$$

Интеграл от функции на всём отрезке $[a;b]$ заменяется на сумму интегралов по подотрезкам $[x_i;x_{i+1}]$ $i = 0,1,\ldots,m-1$, а они вычисляются по выбранным квадратурным формулам.

$$I(a,b,f) = \int_a^b f(x)\mathrm{d}x = \sum_{i=0}^{m-1} I(x_i,x_{i+1},f) =$$
$$= \sum_{i=0}^{m-1}(S_n(x_i,x_{i+1},f) + R_n(x_i,x_{i+1},f)).$$

В силу аддитивности интеграла общая погрешность вычисления интеграла по составной квадратурной формуле складывается из локальных погрешностей на каждом шаге:

$$|R(a,b,f)| = \left|\sum_{i=0}^{m-1} R_n(x_i,x_{i+1},f)\right|,$$

— эта величина называется *средневзвешенной* оценкой погрешности. Кроме того, при оценках используется

$$\sum_{i=0}^{m-1} |R_n(x_i, x_{i+1}, f)|$$

— *гарантирующая* оценка погрешности.

В стандартных программах численного интегрирования используется горизонтальная процедура автоматического выбора шага. Основная идея этой процедуры: оптимальное разбиение отрезка соответствует равным локальным погрешностям на каждом из шагов интегрирования. При численной реализации точное равенство труднодостижимо; в качестве упрощающения шаг интегрирования выбирается так, чтобы локальная погрешность на каждом из шагов **не превосходила** заданную величину ε — целевую оценку локальной погрешности.

Пусть x и h — начало и величина очередного отрезка интегрирования. С использованием выбранной квадратурной формулы вычисляется приближённое значение интеграла и оценка погрешности δ. Если вычисленная оценка по модулю меньше ε, проведённые вычисления принимаются, и следующий расчёт производится на новом отрезке, иначе проведённые вычисления отбрасываются. В любом случае вычисляется новая длина шага h_{new}, используемая при следующем расчёте в качестве длины h.

Итак:

$$|C \cdot h^s| = \delta, \qquad |C \cdot h_{new}^s| = \varepsilon$$

(предполагается, что вычисленная погрешность δ и целевая погрешность ε имеют одну и ту же константу C главного члена погрешности). Из полученных соотношений следует:

$$h_{new} = \sqrt[s]{\frac{\varepsilon}{\delta}} h.$$

Разумеется, попытка написать в программе именно такой способ вычисления является грубой ошибкой программирования.

В качестве примера может быть использован следующий код:

```
chi=pow(delta/epsilon, 1.0/s);
if(chi>10)chi=10;
if(chi<0.1)chi=0.1;
hnew=0.95*h/chi;
```

Во-первых, вычисляется величина δ/ε, обратная требуемой (так как случай $\delta = 0$ возможен, при этом возведение нуля в положительную степень ошибкой не является). Во-вторых, запрещается слишком быстрое увеличение или уменьшение шага интегрирования: в приведённом коде используется десятикратное ограничение изменение длины шага (эмпирическая величина; двукратное и восьмикратное ограничения тоже популярны). Отметим также, что использование условных операторов неэффективно; в стандартных программах используются `max` и `min`. В-третьих, полученный шаг пропорционально уменьшается; страховочный коэффициент `0.95` — эмпирическая величина.

4.4. Вычислительный эксперимент

Задание 4.2. *Провести вычислительный эксперимент.*

- *Написать программу вычисления интеграла по составной квадратурной формуле с горизонтальной процедурой автоматического выбора шага.*

- *Проверить её работу на примерах:*
  1) $f(x) = \sin x$*;*
  2) $f(x) = \frac{1}{x}$*, отрезок* $[a;b]$ *не содержит* 0*;*
  3) $f(x) = \frac{10}{1+25x^2}$*, отрезок* $[a;b]$ $(a < 0,\ b > 0)$*.*

- *Вычислить при* $\varepsilon = \{10^{-7}, 10^{-9}, 10^{-11}\}$*:*
  S_{-7}*,* S_{-9}*,* S_{-11} *— приближённые значения интеграла по составной квадратурной формуле;*
  n_{-7}*,* n_{-9}*,* n_{-11} *— число принятых шагов интегрирования;*
  средневзвешенную и гарантирующую погрешности.
  Оценить изменение погрешности (разности между вычисленным и точным значениями), средневзвешенной и гарантирующей погрешностей, числа шагов в зависимости от параметра ε*.*

  Ожидаемый результат. *При уменьшении величины* ε *в 100 раз средняя длина шага уменьшится в* $100^{1/s}$ *раз, соответственно число шагов интегрирования увеличится в* $100^{1/s}$ *раз, гарантирующая оценка погрешности умень-*

шится в $100^{1-1/s}$ раз (так как ошибка на каждом шаге уменьшится в 100 раз, но самих шагов станет больше).

- *Вычислить коэффициент сходимости, используемый при оценке погрешности:*

$$\frac{S_{-7} - S_{-9}}{S_{-9} - S_{-11}}.$$

Ожидаемый результат.

$$\frac{S_{-7} - S_{-9}}{S_{-9} - S_{-11}} = \frac{(S_{-7} - I(f)) - (S_{-9} - I(f))}{(S_{-9} - I(f)) - (S_{-11} - I(f))} \approx \dots$$

(предполагается, что погрешность вычисления интеграла пропорциональна целевой погрешности на шаге и числу шагов $S - I(f) \approx \mathrm{C} \cdot n \cdot \varepsilon$; C — неизвестный коэффициент пропорциональности)

$$\dots \approx \frac{Cn_{-7}10^{-7} - Cn_{-9}10^{-9}}{Cn_{-9}10^{-9} - Cn_{-11}10^{-11}} \approx \dots$$

(с учётом $n_{-9}/n_{-7} = n_{-11}/n_{-9} = 100^{1/s}$)

$$\dots \approx \frac{10^{-7} - 100^{1/s}10^{-9}}{100^{1/s}10^{-9} - 100^{2/s}10^{-11}} = \\ = \frac{10^{-7} - 100^{1/s}10^{-9}}{(100^{1/s}10^{-2})(10^{-7} - 100^{1/s}10^{-9})} = 100^{1-1/s}.$$

- *Сравнить результаты работы программы интегрирования с горизонтальной процедурой автоматического выбора шага и программы с постоянным шагом.*

4.5. Интегральное уравнение

Задание 4.3. *Решить интегральное уравнение, определив все существующие корни для заданного параметра задачи.*

1)

$$\int_0^x \frac{\cos t + 1}{\sqrt[3]{t}} \mathrm{d}t = \alpha;$$

2)
$$\int_{-x}^{x} \frac{\cos t}{\sqrt{1-t^2}}\mathrm{d}t = \alpha;$$

3)
$$\int_{0}^{x} \ln\sqrt{1+t^2}\mathrm{d}t = \alpha;$$

4)
$$\int_{0}^{x} \frac{\cos t + 1}{\sqrt{t}}\mathrm{d}t = \alpha;$$

5)
$$\int_{0}^{x} \ln\sqrt[3]{1+t^4}\mathrm{d}t = \alpha;$$

6)
$$\int_{-x}^{x} \frac{1+t^2}{\sqrt{1-t^2}}\mathrm{d}t = \alpha;$$

7)
$$\int_{0}^{x} \sqrt{t^3 + \frac{1}{t}}\mathrm{d}t = \alpha^2 + 5\sqrt{x};$$

8)
$$\int_{0}^{x} \ln\sqrt[3]{1+t^4}\mathrm{d}t = \alpha;$$

9)
$$\alpha\int_{x}^{\infty} \frac{\mathrm{d}t}{t\sqrt{1+t^4}} = \int_{0}^{x} \frac{\mathrm{d}t}{\sqrt{1+t^4}};$$

10)
$$\int_{-\infty}^{x} \frac{\mathrm{d}t}{\sqrt{1+t^4}} = \alpha x^3;$$

11)
$$\int_{0}^{x} \ln\left(\frac{e^t - e^{-t}}{e^t + e^{-t}}\right)\mathrm{d}t = \alpha;$$

12)
$$\int_{0}^{x} \cos(\sin t)\mathrm{d}t = \alpha;$$

13)
$$\int_0^x \frac{\mathrm{d}t}{\sqrt{1+t^4}} = \alpha;$$

14)
$$\int_0^x \frac{e^t - e^{-t}}{\sqrt{|t|}(e^t + e^{-t})}\mathrm{d}t = \alpha;$$

15)
$$\int_0^x \frac{x-t}{\sqrt{1+t^4}}\mathrm{d}t = \alpha;$$

16)
$$\int_{-x}^x \sqrt{1+t^5}\mathrm{d}t = \alpha;$$

17)
$$\int_0^x \operatorname{arctg} t \frac{t^2+1}{\sqrt{t^4+1}}\mathrm{d}t = \alpha;$$

18)
$$\int_0^x \frac{\sin t + 1}{\sqrt{t}}\mathrm{d}t = \alpha;$$

19)
$$\int_\alpha^x \sqrt{\frac{1+t^3}{10+t^2}}\mathrm{d}t = \alpha + x;$$

20)
$$\int_1^x \sqrt{t + \frac{1}{1+t^2}}\mathrm{d}t = \alpha x;$$

21)
$$\int_0^x \sqrt{1+\alpha^2 t^4}\mathrm{d}t = \alpha + x;$$

22)
$$\int_0^x \frac{\operatorname{arctg} t}{\sqrt[3]{t}}\mathrm{d}t = \alpha;$$

23)
$$\int_0^x (\sin t + 1.1)\left(1 - \frac{1}{t^2+1}\right)\mathrm{d}t = \alpha;$$

24)
$$\int_0^x \frac{1}{\sqrt{|t|}}\frac{t^2-1}{t^2+1}\mathrm{d}t = \alpha;$$

25)
$$\int_0^x \frac{1}{t^2+1}\frac{\cos^2 t + 1}{\sin^2 t + 1}\mathrm{d}t = \alpha.$$

26)
$$\int_0^x \frac{1}{\sqrt[3]{|t|}}\frac{\cos^2 t + 1}{\sin^2 t + 1}\mathrm{d}t = \alpha.$$

27)
$$\int_0^x \frac{\sin t + 1}{\sqrt[3]{t}}\mathrm{d}t = \alpha;$$

28)
$$\int_0^x \frac{\operatorname{arctg} t}{\sqrt{t}}\mathrm{d}t = \alpha;$$

29)
$$\int_0^x \operatorname{arctg} t\left(\frac{t^2 - 5t + 1}{\sqrt{t^4+1}}\right)\mathrm{d}t = \alpha.$$

Указания.

0. Свести задачу к виду:
$$\int_a^x f(t)\mathrm{d}t = \alpha.$$

1. Если в задаче имеется особенность, то её необходимо преобразовать к задаче без особенности (например, используя замену переменных).

2. Если правая часть не является константой, задачу необходимо преобразовать.

3. Необходимо оценить, является ли интеграл сходящимся или расходящимся. Если интеграл расходящийся, то решение существует для любого параметра α. Если же интеграл сходится, то решение может не существовать; несобственный интеграл необходимо вычислить.

4. Необходимо выяснить знак подынтегральной функции. В случае её положительности (знакоопределённости) решение единственно. Если же у функции знаки чередуются, то необходимо определить точки смены знака (нули подынтегральной функции) и вычислить соответствующие интегралы для определения границ изменения числа корней.

5. Интегрирование осуществляется с автоматическим выбором шага. Целевая погрешность на шаге ε является параметром задачи.

6. Важным условием является малое число вычислений подынтегральной функции.

Кажущаяся разумной идея разделения задачи на две подзадачи: вычисления первообразной

$$F(x) = \int_a^x f(x)\mathrm{d}x$$

и поиска корня функции

$$F(x) - \alpha = 0,$$

— в итоге приводит к неприемлемо большому числу операций, и потому является ошибкой. Наоборот, поиск корня должен производиться **внутри** вычисления интеграла — таким образом, в правильном решении интеграл должен вычисляться ровно один раз.

Итак, пусть для простоты $f(x) > 0 \ \forall x$, для текущего значения $F(x) < \alpha$ и при выбранном шаге h модуль оценки локальной погрешности меньше целевой $\delta < \varepsilon$. Если $|F(x+h) - \alpha| \leqslant \varepsilon$, то вычисления следует остановить. Если $F(x+h) < \alpha - \varepsilon$, то осуществляется переход к следующей базовой точке, и вычисления продолжаются. Если же $F(x+h) > \alpha + \varepsilon$, то искомый корень находится на интервале $(x; x+h)$. В этом случае необходимо уменьшить шаг интегрирования. Если воспользоваться делением отрезка пополам, уменьшив шаг в два раза, решение будет найдено, но потребует порядка 50 шагов. Метод хорд позволяет найти

решение за 3–5 шагов. Методы нахождения корней старших порядков вполне оправданы — вычисленные на предыдущих шагах метода интегрирования значения первообразной $F(x)$ (значения определённого интеграла $F(x) = \int_0^x f(t)\mathrm{d}t$), подынтегральной функции $f(x)$ и её производных позволяют приблизить первообразную многочленом большего порядка. Всё же эффективных реализаций этой идеи авторам не встречалось.

7. После определения корня x_* следует оценить погрешность его определения с использованием линейной оценки (см., например, [2, § 1.5.5, с. 30]). Пусть для определённости $f(x) > 0$ в некоторой окрестности вычисленного корня. Тогда

$$x_*^{\min} = x_* - \frac{F(x_*) - \alpha}{f(x_*)} - \frac{\Delta_{\text{гар}}}{f(x_*)},$$

$$x_*^{\max} = x_* - \frac{F(x_*) - \alpha}{f(x_*)} + \frac{\Delta_{\text{гар}}}{f(x_*)},$$

где x_* — вычисленное значение корня, $F(x_*)$ — вычисленное значение интеграла (согласно условию остановки отличается от α не более чем на ε), $\Delta_{\text{гар}}$ — вычисленная гарантирующая оценка погрешности.

Задание 4.4. *В чём геометрический смысл линейной оценки пункта 7?*

Глава 5.

Матрицы

5.1. Хранение

Работа с матрицами имеет специфику, связанную с необходимостью передавать матрицу в качестве параметра в функцию.

В языке C используются два подхода:

1. Элементы матрицы $a_{i,j}$, $i = 0, \ldots, n-1$, $j = 0, \ldots, m-1$ (i — номер строки, j — столбца) "укладываются" в одномерный массив длины nm. Обозначим такой массив через `b` и индекс `k`.

а) Хранение может осуществляться "по строкам". В этом случае $k = i \cdot m + j$, $i = [k/m]$, $j = k \bmod m$.

б) Хранение может осуществляться "по столбцам": $k = i + n \cdot j$, $i = k \bmod n$, $j = [k/n]$.

2. В виде двумерного массива.

а) Хранение "по строкам": для каждой строки выделяется отдельный массив длины m, а указатели на начала этих массивов-строк соберем в дополнительном массиве `a` длины n. Таким образом, `a[i]` является указателем на i-ю строку матрицы a, а `a[i][j]` — элементом $a_{i,j}$.

б) Хранение "по столбцам": для каждого столбца выделяется отдельный массив длины n, а указатели на начала этих массивов-столбцов соберем в дополнительном массиве `a` длины m. Таким образом, `a[j]` является указателем на j-й столбец матрицы a, а `a[j][i]` — элементом $a_{i,j}$.

Задание 5.1. *Написать функции заполнения квадратной матрицы значениями на основе каждого из описанных подходов:*
— тест 1

$$a_{i,j} = \begin{cases} 1, & i = j = 0, \\ 2, & i = j = 1, \ldots, n-1, \\ -1, & |i - j| = 1, \\ 0, & \textit{иначе}; \end{cases}$$

— обратная к матрице теста 1

$$a_{i,j} = n + 1 - \max\{i, j\};$$

— *тест 2*

$$a_{i,j} = |i - j|;$$

— *обратная к матрице теста 2*

$$a_{i,j} = \begin{cases} -\dfrac{n-2}{2n-2}, & i = j,\ j = 0, n-1, \\ -1, & i = j,\ j = 1, \ldots, n-2, \\ \dfrac{1}{2}, & |i - j| = 1, \\ \dfrac{1}{2n-2}, & (i,j) = (0, n-1), (n-1, 0); \\ 0, & \textit{иначе}; \end{cases}$$

— *матрица Гильберта*

$$\Gamma_{i,j} = \frac{1}{1+i+j}, \qquad i,j = 0, \ldots, n-1;$$

— *обратная к матрице Гильберта*

$$\tilde{\Gamma}_{i,j} = \frac{(-1)^{i+j}(n+i)!(n+j)!}{(i!)^2(j!)^2(n-i-1)!(n-j-1)!(i+j+1)}, \qquad i,j = 0, \ldots, n-1;$$

— *матрица Гильберта плюс единичная*

$$a_{i,j} = \Gamma_{i,j} + E_{i,j} = \frac{1}{1+i+j} + \delta_{ij}, \qquad i,j = 0, \ldots, n-1$$

(внимание: при выполнении этого задания не использовать условный оператор);
— *матрица Гильберта без левого нижнего угла (первый вариант)*

$$\tilde{\Gamma}_{i,j} = \begin{cases} \dfrac{1}{1+i+j}, & i \leqslant j + k, \\ 0, & \textit{иначе}; \end{cases}$$

— *матрица Гильберта без левого нижнего угла (второй вариант)*

$$\tilde{\Gamma}_{i,j} = \begin{cases} 0, & i > n - k, \quad j < k, \\ \dfrac{1}{1+i+j}, & \textit{иначе}; \end{cases}$$

— *матрица Вандермонда*

$$V_{i,j} = x_i^j, \qquad i,j = 0,\ldots,n-1;$$

Задание 5.2. *Реализовать функции для выполнения операций суммирования и умножения двух прямоугольных матриц, умножения матрицы на число, умножения матрицы на вектор.*

5.2. Задачи и методы

В пособии рассматривается две задачи вычислительной линейной алгебры. Во-первых, задача решения системы линейных алгебраических уравнений

$$Ax = b,$$

где A — квадратная матрица размера $n \times n$, x и b — вектор-столбцы размера n.

$$A = \begin{pmatrix} a_{0,0} & a_{0,1} & a_{0,2} & \ldots & a_{0,n-1} \\ a_{1,0} & a_{1,1} & a_{1,2} & \ldots & a_{1,n-1} \\ \vdots & \vdots & \vdots & & \vdots \\ a_{n-1,0} & a_{n-1,1} & a_{n-1,2} & \ldots & a_{n-1,n-1} \end{pmatrix},$$

$$x = \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_{n-1} \end{pmatrix}, \qquad b = \begin{pmatrix} b_0 \\ b_1 \\ \vdots \\ b_{n-1} \end{pmatrix}$$

Во-вторых, задача нахождения обратной матрицы A^{-1}.

Предполагается, что матрицы небольшие (размер матриц в тестовом задании $1000 \leqslant n \leqslant 1500$), и потому при решении этих задач могут быть использованы метод Гаусса и метод Жордана.

Метод Гаусса состоит в том, что элементарными преобразованиями над строками матрицы она приводится к верхнетреугольному виду с единичными элементами на главной диагонали (прямой ход метода Гаусса); полученная верхнетреугольная матрица элементарными преобразованиями строк приводится к единичной (обратный ход метода Гаусса). Метод Жордана похож

на метод Гаусса, но элементарными преобразованиями матрица приводится сразу к единичной. При решении системы уравнений преобразования делаются с расширенной матрицей размерности $n \times (n+1)$ в начале применения алгоритма совпадающей с (Ab); при обращении матрицы — с расширенной матрицей размерности $n \times (2n)$ в начале применения алгоритма совпадающей с (AE), где E — единичная матрица.

Предположим, что $a_{0,0} \neq 0$. Поделив первую строку на $a_{0,0}$, получим

$$A = \begin{pmatrix} 1 & a_{0,1}^{(1)} & a_{0,2}^{(1)} & \dots & a_{0,n-1}^{(1)} & \dots \\ a_{1,0} & a_{1,1} & a_{1,2} & \dots & a_{1,n-1} & \dots \\ \vdots & \vdots & \vdots & & \vdots & \dots \\ a_{n-1,0} & a_{n-1,1} & a_{n-1,2} & \dots & a_{n-1,n-1} & \dots \end{pmatrix}.$$

Умножая по очереди первую строку на $a_{i,0}$ и вычитая результат из соответствующей строки для $i = 1, \dots, (n-1)$, получим:

$$A = \begin{pmatrix} 1 & a_{0,1}^{(1)} & a_{0,2}^{(1)} & \dots & a_{0,n-1}^{(1)} & \dots \\ 0 & a_{1,1}^{(1)} & a_{1,2}^{(1)} & \dots & a_{1,n-1}^{(1)} & \dots \\ \vdots & \vdots & \vdots & & \vdots & \dots \\ 0 & a_{n-1,1}^{(1)} & a_{n-1,2}^{(1)} & \dots & a_{n-1,n-1}^{(1)} & \dots \end{pmatrix}.$$

Далее предположим, что $a_{1,1}^{(1)} \neq 0$. Разделим вторую строку на $a_{1,1}^{(1)}$:

$$A = \begin{pmatrix} 1 & a_{0,1}^{(1)} & a_{0,2}^{(1)} & \dots & a_{0,n-1}^{(1)} & \dots \\ 0 & 1 & a_{1,2}^{(2)} & \dots & a_{1,n-1}^{(2)} & \dots \\ \vdots & \vdots & \vdots & & \vdots & \dots \\ 0 & a_{n-1,1}^{(1)} & a_{n-1,2}^{(1)} & \dots & a_{n-1,n-1}^{(1)} & \dots \end{pmatrix}.$$

Умножая по очереди вторую строку на $a_{i,1}^{(1)}$ и вычитая результат из соответствующей строки в методе Гаусса для $i = 2, \dots, (n-1)$, получим:

$$A = \begin{pmatrix} 1 & a_{0,1}^{(1)} & a_{0,2}^{(1)} & \dots & a_{0,n-1}^{(1)} & \dots \\ 0 & 1 & a_{1,2}^{(2)} & \dots & a_{1,n-1}^{(2)} & \dots \\ \vdots & \vdots & \vdots & & \vdots & \dots \\ 0 & 0 & a_{n-1,2}^{(2)} & \dots & a_{n-1,n-1}^{(2)} & \dots \end{pmatrix}.$$

Метод Жордана отличается от метода Гаусса набором строк $i = 0, 2, \ldots, (n-1)$:

$$A = \begin{pmatrix} 1 & 0 & a_{0,2}^{(2)} & \ldots & a_{0,n-1}^{(2)} & \ldots \\ 0 & 1 & a_{1,2}^{(2)} & \ldots & a_{1,n-1}^{(2)} & \ldots \\ \vdots & \vdots & \vdots & & \vdots & \ldots \\ 0 & 0 & a_{n-1,2}^{(2)} & \ldots & a_{n-1,n-1}^{(2)} & \ldots \end{pmatrix}.$$

Соответственно, на третьем шаге алгоритмов получаются матрицы:

$$A = \begin{pmatrix} 1 & a_{0,1}^{(1)} & a_{0,2}^{(1)} & \ldots & a_{0,n-1}^{(1)} & \ldots \\ 0 & 1 & a_{1,2}^{(2)} & \ldots & a_{1,n-1}^{(2)} & \ldots \\ 0 & 0 & 1 & \ldots & a_{2,n-1}^{(3)} & \ldots \\ \vdots & \vdots & \vdots & & \vdots & \ldots \\ 0 & 0 & 0 & \ldots & a_{n-1,n-1}^{(3)} & \ldots \end{pmatrix}$$

— $i = 3, \ldots, (n-1)$ для метода Гаусса и

$$A = \begin{pmatrix} 1 & 0 & 0 & \ldots & a_{0,n-1}^{(3)} & \ldots \\ 0 & 1 & 0 & \ldots & a_{1,n-1}^{(3)} & \ldots \\ 0 & 0 & 1 & \ldots & a_{2,n-1}^{(3)} & \ldots \\ \vdots & \vdots & \vdots & & \vdots & \ldots \\ 0 & 0 & 0 & \ldots & a_{n-1,n-1}^{(3)} & \ldots \end{pmatrix}$$

— $i = 0, 1, 3, \ldots, (n-1)$ для метода Жордана.

Для уменьшения вычислительной погрешности в методах Гаусса и Жордана производится **выбор ведущего элемента** — определяется самый большой по модулю элемент.

а) По столбцу: осуществляется поиск в первом столбце необработанной подматрицы:

$$\max_{i=j+1,\ldots,n-1} |a_{i,j}|.$$

Затем найденная строка меняется местами с первой строкой необработанной части. С точки зрения решения системы линейных алгебраических уравнений такая перестановка может быть интерпретирована как перенумерация уравнений. Если массив

хранится как двумерный по строкам (2а), то такая перестановка может быть осуществлена за счёт смены местами указателей на строки; во всех остальных случаях необходимо переставлять каждый из элементов строки в необработанной части.

б) По строке: осуществляется поиск в первой строке необработанной подматрицы:

$$\max_{j=i+1,\ldots,n-1} |a_{i,j}|.$$

Затем найденный столбец меняется местами с первым столбцом необработанной части. С точки зрения решения системы линейных алгебраических уравнений такая перестановка может быть интерпретирована как перенумерация неизвестных. Поэтому в этом случае необходимо выделять дополнительный целочисленный массив для хранения произведённой перестановки (кортежа).

в) По всей подматрице:

$$\max_{i=k,\ldots,n-1;j=k,\ldots,n-1} |a_{i,j}|.$$

Затем найденная строка меняется местами с первой строкой необработанной части матрицы, а найденный столбец — с первым столбцом необработанной части. Как и в случае с поиском ведущего элемента по строке, необходимо хранение получившейся перестановки неизвестных.

5.3. Вычислительный эксперимент

Задание 5.3. *Необходимо написать какой-либо метод решения (Гаусса или Жордана) с выбором ведущего элемента (по столбцу, по строке или по всей матрице) системы линейных алгебраических уравнений или обращения матрицы и проверить его работоспособность.*

Построение тестовой задачи.

Для основного набора тестовых матриц размерность задачи выбирается $1000 \leqslant n \leqslant 1500$. Задаётся некоторый вектор x (например: все компоненты единичные, чередование ± 1, случайным образом и т.п.).

Для матрицы Вандермонда размерность задачи выбирается $10 \leqslant n \leqslant 100$ и вектор x задаётся — первые несколько компонент ± 1, остальные нули.

Умножением матрицы A на вектор x вычисляется столбец правой части b. Матрица A и столбец b становятся входными параметрами алгоритма, и по ним вычисляется решение $\tilde{x}$. Разность векторов $\delta x = \tilde{x} - x$ называется вектором погрешности. С использованием исходной матрицы A и правой части b (так как метод исходную матрицу и исходный столбец правой части "испортил", их необходимо восстановить) вычисляется невязка $\delta b = A\tilde{x} - b$.

Отметим, что вектор погрешности известен только в тестовых примерах; в реальных задачах его требуется каким-либо образом оценить. Вектор невязки известного точного решения не требует и вычисляется для уточнения решения и оценки погрешности.

Для векторов погрешности и невязки вычисляются нормы: $||\cdot||_1$ (сумма модулей компонент), $||\cdot||_2$ (корень из суммы квадратов компонент), $||\cdot||_\infty$ (максимальная по модулю компонента). Таким образом, в задаче решения системы линейных уравнений вычисляется шесть чисел.

Ожидаемые результаты: для "матрицы Гильберта плюс единичной" все эти нормы маленькие; в остальных случаях нормы вектора невязки маленькие (или небольшие), погрешности катастрофически велики.

Нарастание погрешности решения системы с матрицей Вандермонда соответствует увеличению погрешности интерполяции с ростом числа узлов для многочлена Лагранжа. Забегая вперёд, отметим, что матрица Гильберта и Вандермонда плохо обусловлены.

В случае вычисления обратной матрицы при известной обратной возможно вычисление погрешности $\Delta A = \tilde{A}^{-1} - A^{-1}$ (из вычисленной обратной матрицы вычитается точная обратная). Кроме того во всех случаях вычисляется матрица невязки $\delta A = A^{-1}A - E$. Разумеется, так как в процессе работы метода исходная матрица "испорчена", её необходимо восстановить (перезаписать в память).

Внимание! Если точная обратная матрица неизвестна, то вычисляется только матрица невязки.

Для матриц δA и ΔA также вычисляются по три величины

— сумма модулей элементов, корень из суммы квадратов элементов, максимальный по модулю элемент. Забегая вперёд, отметим: эти величины — векторные нормы в n^2–мерном пространстве не являются подчинёнными матричными нормами; с точки зрения оценки качества решения задачи они достаточны. Для “матрицы Гильберта плюс единичная” эти числа должны быть малы, для плохо обусловленных матриц — велики.

В завершение вычислительного эксперимента следует реализовать один и тот же метод с четырьмя разными способами хранения матрицы в памяти. В качестве проверки правильности реализации следует использовать один и тот же тестовый пример (с большой матрицей). Таким образом, в этих четырёх случаях должны быть получены в точности одни и те же числа — различаться будет только время работы.

Внимание! Если четыре тестовые программы выдают неодинаковые результаты, это свидетельствует об ошибке. Разумеется, её нужно найти и исправить.

Внимание! Запуск программ должен производиться последовательно (следующая начинает работать только после окончания работы предыдущей), а не параллельно (параллельная работа программ будет изучаться на следующих курсах). Все другие пользовательские задачи желательно остановить.

Внимание! Предполагается, что использовать функции времени студенты научились на первом курсе в первом семестре при изучении сортировок. Если студент не умеет их использовать (“забыл”, как они используются) — это важный недочёт, который необходимо исправить.

Ожидаемый результат: способ хранения по строкам существенно быстрее, чем способ хранения по столбцам. Это связано с использованием кэш-памяти (изучается на следующих курсах, для первого курса необходимо уяснить, что способ хранения данных в памяти может приводить к ускорению или замедлению работы программы).

Литература

1. *Банди Б.* Методы оптимизации. Вводный курс: Пер. с англ. М.: Радио и связь, 1988.
2. *Бахвалов Н.С., Жидков Н.П., Кобельков Г.М.* Численные методы.
3. *Богачёв К.Ю.* Практикум на ЭВМ. Методы приближения функций. М.: Изд-во ЦПИ при механико–математическом ф–те МГУ, 2002.
4. *Валединский В.Д., Корнев А.А.* Методы программирования в примерах и задачах. М.: Изд–во мех.–мат. ф-та МГУ, 2000.
5. *Васильев Ф.П.* Численные методы решения экстремальных задач. М.: Наука, 1988.
6. *Васильев Ф.П.* Методы оптимизации. М.: "Факториал Пресс", 2002.
7. *Ганшин Г.С.* Методы оптимизации и решение уравнений. М.: Наука, 1987.
8. *Демидович Б.П., Марон И.А.* Основы вычислительной математики. М.: Наука, 1966.
9. *Стронгин Р.Г.* Численные методы в многоэкстремальных задачах. М.: Наука, 1978.
10. *Сухарев А.Г., Тимохов А.В., Фёдоров В.В.* Курс методов оптимизации. М.: Наука, 1986.
11. *Уоррен, Генри С. мл.* Алгоритмические трюки для программистов, 2-е изд.: Пер. с англ. М.: "Вильямс", 2014.
12. *Федоренко Р.П.* Введение в вычислительную физику. М.: МФТИ, 1994.
13. *Химмельблау Д.* Прикладное нелинейное программирование. М.: Мир, 1975.