Лабораторный практикум

Для выполнения лабораторных работ студентам необходимо предварительно изучить принципы и правила написания программ на языке ФОРТРАН ([1] или сайт accel.ru). Описание лабораторных работ содержит лишь минимальные сведения.

Совет начинающим программистам. Результат вычислений, выведенный на экран, должен быть не только правильным, но и «красиво» оформленным (удобочитаемым - с соответствующими текстовыми пояснениями числовых данных). Для этих целей служит механизм форматирования выходных данных.

1. Основные элементы программирования

Лабораторная работа № 1. Первая программа.

Задание 1.

Первый вопрос начинающего программиста – как ввести информацию с клавиатуры и как вывести информацию на экран.

В редакторе напишите программу, в которой используется только процедура вывода текста на экран - *write* (*текст*). Выводимый текст - ('Hello, computer') - заключён в одиночные кавычки.

```
program first
  write(*,*) 'Hello, computer'
end
```

Исходный текст программы транслируется и в случае отсутствия синтаксических ошибок создаётся исполняемый файл. Этот файл посылается на исполнение. Результат – «Hello, computer», программа выполнена, но на экране результат работы может быть и не отражен.

Задание 2.

Добавьте в предыдущую программу оператор *read.*

```
program first
write(*,*) 'Hello, computer'
  read(*,*)
end
```

Теперь при выполнении программы на экране появится надпись

Hello, computer.

Чтобы вернуться к исходному тексту в окне редактора, нажмите клавишу *Enter*.

Разница между этими программами заключается в том, что при последовательном выполнении операторов программы, после оператора *write* - вывода на экран текста ('Hello, computer') - следует оператор ввода данных *read* без параметров (т.е. фиктивное чтение). Программа ожидает ввода данных. Нажимая клавишу *Enter,* осуществляем операцию фиктивного чтения данных и возвращаемся к исходному тексту программы.

Задание 3.

Тексты (или отдельные символы) можно выводить на экран, используя переменные, объявленные как строки или символы – *character(*количество символов)*. Тексты могут быть написаны на любом языке (например, на русском – «Привет, компьютер»).

```
Character*25 abc {Описание строковой переменной }
abc= 'Привет, компьютер'  {Присвоение строковой переменной abc значения }
write(*,*) abc  {Вывод значения строковой переменной на экран}
read(*,*)
end
```

Результат выполнения этой программы такой же, как и предшествующей программе - «Привет, компьютер».

Лабораторная работа № 2. Линейные программы.

Линейная программа – программа, в которой идет последовательное выполнение операторов без повторений пройденных участков программы и разветвлений.

Задание 1.

Напишите программу, в которой производятся простейшие арифметические вычисления, и результаты выведите на экран. Для этого нужно:

1. Объявить несколько переменных целого (*integer*) и вещественного (*real*) типов.
2. Ряд переменных использовать как входные параметры - ввод конкретных значений через процедуру ввода данных - *read(список переменных).*
3. Записать простейшие арифметические действия.

4. Ряд переменных использовать как выходные параметры - вывод конкретных значений через процедуру вывода данных - *write (список переменных).*

Задание 2.

Задайте вещественное число *x*. Используя только операции умножения, сложения и вычитания, произведите вычисления приведённых выражений за указанное количество операций. Приведённые выражения нужно соответствующим образом преобразовать (в т.ч. с использование скобочных конструкций).

- $2x^4 - 3x^3 + 4x^2 - 5x + 6$ (не более 4-х умножений, не более 4-х сложений и вычитаний)
- $1 - 2x + 3x^2 - 4x^3$ (не более 8-и операций)
- $1 + 2x + 3x^2 + 4x^3$ (не более 8-и операций)

Задание 3.

Задайте значения параметров переменными типа *real*. Вычислите следующие выражения. Для тестирования задайте значения *a, b, c, d* в диапазоне $1 \div 4$ в алгебраических выражениях и $a = \pi/4,\ \pi/2,\ \pi$ - в тригонометрических выражениях.

1. $x = \big((a+b)(b+c)+d\big)(a-b)$
2. $x = \Big(\big(a/b + 1\big)\big(b/a - 1\big) - 1\Big)/(a+b)$
3. $x = cos(a) + sin(a) + cos(3a) + sin(3a)$
4. $x = \dfrac{sin(2a) + sin(5a) - sin(3a)}{cos(a) + 1 - 2\,sin^2(2a)}$
5. $x = sin^2(a) + cos^2(a)$
6. $x = ctg\big(3/4\,\pi + 3/2\,a\big)$
7. $x = \dfrac{1 - tg(a)}{1 + tg(a)}$
8. $x = \dfrac{\sqrt{a} + \sqrt{b}}{a}$
9. $x = 1 + \dfrac{b^2 + c^2 - a^2}{2bc}$
10. $x = \dfrac{a^2 - 1}{\sqrt{a} - \sqrt{b}}$

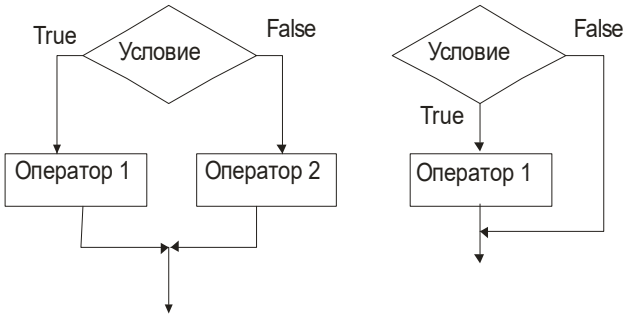
Лабораторная работа № 3. Разветвляющиеся вычислительные процессы: условный оператор *if*.

Для того чтобы в зависимости от конкретных значений обрабатываемых данных обеспечить выполнение двух (или более) различных последовательностей операторов, используются оператор ветвления программы – *if***.** Ветвление программы осуществляется в зависимости от вычисляемого значения логического условного выражения - «правда» или «ложь» .

```
if    < условие >   then
        < оператор1 >     {Выполняется, если условие = 'правда'}
else
        <оператор2>       {Выполняется, если условие = 'ложь'}
endif
```

Вначале вычисляется <*условие*>. Если оно имеет значение – «правда», то выполняются < *оператор1* > и управление передается на конец оператора *if.* Если оно имеет значение – «ложь», то выполняются < *оператор2* >. Часть *else* оператора *if* может быть опущена и тогда, в случае значения «ложь», весь оператор *if* как бы пропускается.

< *Оператор1* > и < *оператор2* > - операторы любого типа, в том числе, и условные. В последнем случае возникает вложенная логическая конструкция. На некоторых уровнях вложенности может отсутствовать часть *else.* Чтобы не было путаницы уровней вложенности, существует правило – часть *else* соответствует ближайшей к ней «сверху» части *then* условного оператора *if*.



Структурная схема условного оператора

Логическое условное выражение может быть представлено простыми операциями отношения – равно, не равно, больше, меньше, меньше-равно, больше-равно – *a.eq.d, c.ne.s, m.gt.n, r.lt.p, k.le.z, f.ge.q*.

Сложные условные выражения (т.е. проверка нескольких соотношений переменных совместно) строятся с использованием логических операций – *и, или, не*.

Например, (*a.eq.d*) *.and.* (*c.ne.s*), (*m.gt.n*). *or.* (*r.le.p*), (*s.ne.k*).

При записи сложных условных выражений необходимо помнить, что логические операции имеют более высокий приоритет исполнения, чем операции отношения. Поэтому, чтобы не нарушалась логика конструкции, нужно элементы, содержащие операции отношения, заключать в круглые скобки. При построении сложных логических конструкций с использованием элементов логики – *и, или, не*, анализ структуры можно разложить на простейшие операции сравнения двух входных сигналов.

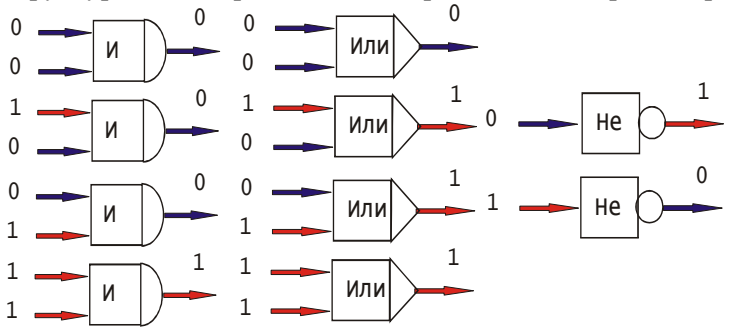


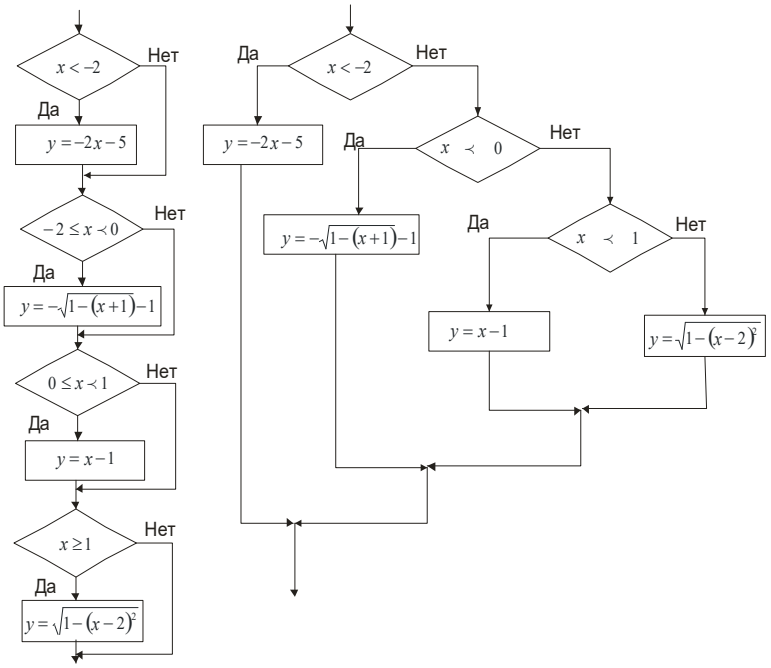
Схема действия логических операторов

<u>*Совет начинающим программистам*</u>. По возможности старайтесь избегать очень длинных и сложных логических конструкций (возможный источник логических ошибок). Как правило, их можно разделить на отдельные завершённые фрагменты.

Рассмотрим следующий пример применения условного оператора *if*.

$$y=\begin{cases}-2x-5 & -3\le x<-2\\ -\sqrt{1-(x+1)}-1 & -2\le x<0\\ x-1 & 0\le x<1\\ \sqrt{1-(x-2)^2} & 1\le x<3\end{cases}$$

Задача может быть решена по разным структурным схемам.



Структурные схемы алгоритмов

Алгоритм можно сформулировать следующим образом.

1. Ввести значение аргумента *x***.**
2. Проверить принадлежит ли это значение области определения функции.
3. Проверить в каком диапазоне лежит значение аргумента *x* и произвести вычисление функции *y* по соответствующей формуле.

Практический совет.

1. Если нет особых требований к быстродействию алгоритма, то при написании программы лучше отдавать предпочтение более простым и надежным алгоритмам.
2. При необходимости организовать разветвления вычислительного процесса, в зависимости от логических условий, по трем и более направлениям, нужно использовать алгоритмы с использованием вложенных структур оператора *if*.

Задание 1.

Поиск наибольшего числа из трех вещественных чисел.

Задание 2.

Добавьте в предшествующую программу поиск минимального числа и выведите числа в следующем порядке – *«минимальное число, среднее число, максимальное число»* - числовые значения с соответствующими текстовыми пояснениями (*среднее число* - среднее по величине из трёх введённых чисел).

Задание 3. Вычислите значение функции.

1. $$y=\begin{cases} -3+\sqrt{9-(9+x)^2} & -9\le x<-6 \\ 3+x & -6\le x<-3 \\ \sqrt{(3+x)^2} & -3\le x<0 \\ 3-x & 0\le x<3 \end{cases}$$

2. $$y=\begin{cases} -2 & x\le -2 \\ 2+x & -2<x<0 \\ 2-x & x\ge 0 \end{cases}$$

3. $$y=\begin{cases} \sqrt{4-x^2} & -2\le x\le 2 \\ x-2 & 2<x\le 4 \\ 2 & x>4 \end{cases}$$

4. $$y=\begin{cases}\sin(x) & 0\le x\le\pi\\ \sin(x-\pi) & \pi<x\le 2\pi\\ -\sin(x-2\pi) & 2\pi<x\le 3\pi\end{cases}\Bigg\}$$

5. $$y=\begin{cases}\sin(x) & 0\le x\le\pi/2\\ 1 & \pi/2<x\le 3\pi/2\\ -\sin(x-\pi) & 3\pi/2<x\le 2\pi\\ x-2\pi & x>2\pi\end{cases}\Bigg\}$$

Лабораторная работа № 4. Разветвляющиеся вычислительные процессы: оператор выбора *case*.

Для того чтобы в зависимости от конкретных значений исходных данных обеспечить выполнение двух (или более) различных последовательностей операторов, используются оператор выбора *case.* Оператор *case* – осуществляет ветвление программы по значению ключа. Он позволяет выбирать одно из нескольких возможных продолжений программы. Определение направления движения связано с *ключом* – выражением любого порядкового типа.

Select case <ключи выбора>
case *<символ(ы) из перечня ключи выбора>*
<оператор_1>
case *<символ(ы) из перечня ключи выбора>*
<оператор_2>
case *<символ(ы) из перечня ключи выбора>*
<оператор_3>
...
case *<символ(ы) из перечня ключи выборса>*
<оператор_n>

} *<список выбора>*

else
<оператор_k>
End select

Сначала вычисляется значение выражения *<ключ выбора>,* затем в последовательности операторов *<список выбора>* отыскивается оператор, константа которого совпадает с ключом выбора. Одному оператору из списка выбора может соответствовать несколько констант выбора, которые записываются через запятую. Выбранный оператор выполняется, и оператор выбора в целом завершает свою работу. Если в списке выбора не будет найдена константа, соответствующая ключу выбора, то управление передается оператору, следующему за словом *else* , и затем оператор выбора завершает свою работу. Часть *else <оператор>* может быть опущена, тогда, если значение ключа не соответствует списку выбора, оператор *case* просто завершает свою работу и выполнение задания передается следующему за ним оператору.

Примечание: *<символ(ы) из перечня ключи выборса>* можно задавать также в виде диапазона значений, например 5:10 или $'i':'n'$. Допустимы и следующие формы записи :10 или 5: , т.е. первая запись справедлива для всех значений меньше или равно 10, вторая запись – для всех значений больше или равно 5.

Задание 1.

Определите время года по номеру месяца. Структура оператор *case* должна содержать четыре позиции по временам года – зима, весна, лето, осень – соответствующие им константы (метки) означают номер месяца в порядке следования.

Задание 2.

Промоделируйте работу лототрона. Выигрышные номера:

1. Квартира в Москве;
2. Автомобиль;
3. Туристическая путёвка;
4. Зубная щётка;

Все остальные номера – наилучшие пожелания в следующей лотерее.

Для выбора номера используйте генератор случайных чисел в диапазоне от 1 до 12 .

Пример: Генерация случайного числа (ГСЧ) в диапазоне от 1 до 10.

```
! ---случайное число a----
integer   a
! --------------старт генератора-ГСЧ-----------------------------------
!---- не изменять---------
Integer   sv, ag(100), ap(100)
       sv=50
       call random_seed(size=sv)
       call random_seed(put=ap)
       call random_seed(get=ag)
!---вычисление случайного числа----------------------
            call random_number(hav)
! генерация целых чисел в диапазоне от 0 до 1.Умножение на 10 расширяет диапазон – от 0 до 10.
              a=anint(hav*10)
```

Для генерации при очередном запуске программы одной и той же последовательности случайных чисел обращения к подпрограмме

```
           call random_seed(put=ap)
           call random_seed(get=ag)
```

нужно исключить.

(Более подробно ГСЧ - см. лабораторную работу №6).

Задание 3.

Задайте целое число в пределах n≤100 (возраст человека). Сформируйте правильную фразу, используя фрагменты определения - «.. год» «.. года», «.. лет», и выведите её на экран. Например,

«возраст человека 1 год»
«возраст человека 22 года»
«возраст человека 15 лет»

Лабораторная работа № 5. Операторы повторения.

Используются два различных оператора, позволяющих запрограммировать повторяющиеся участки программы. Различия в их действиях следующие:

1. Повторение операций *заданное число раз* вне зависимости от условий.
2. Повторение операций пока логическое условие, проверяемое **до** выполнения основного тела цикла, *истинное*.
3. Повторение операций пока логическое условие, проверяемое **после** выполнения основного тела цикла, *ложное*.

Таким образом, перекрывается весь спектр возможных ситуаций повторения действий.

Циклы применяются как для работы с индексированными переменными (например, элементами массива), индексы которых совпадают по имени с переменной цикла, а также и для вычислений, когда в теле цикла изменение значений каких-либо переменных не связано напрямую с переменной цикла (зависит от какой либо целочисленной переменной, значения которой изменяются в процессе выполнения очередного шага цикла).

Необходимо отметить, что переменная цикла не может принимать значения вне диапазона, указанного в описании массива. Но в объявленном диапазоне индексов можно задавать произвольные начальные и конечные значения переменной цикла.

1. Оператор цикла - *DO*.

do <переменная цикла>= < нач. знач> ,<кон. знач.> ,<шаг.>

Сначала определяется начальное значение и присваивается переменной цикла. Производится выполнение оператора. Затем значение переменной цикла увеличивается на «*шаг*». Цикл завершается, когда текущее значение переменной цикла становится больше *<кон. знач.>*. Если условие *переменная цикла* больше или равна *<кон. знач.>* изначально нарушено, то оператор не будет выполнен..

2. Оператор цикла с предусловием - *DO WHILE*

do while <логическое условие>
<оператор>

Если логическое условие - *«правда»,* то *оператор* выполняется до тех пор, пока логическое условие не примет значение *«ложь»*. В теле цикла обязательно должны изменяться значения параметров логического выражения (иначе может получиться бесконечный цикл). Если логическое условие ложно изначально, то цикл не выполняется ни разу.

Циклы завершается оператором *end do*.

Задание 1. Оператор цикла *DO*.

1. Найдите сумму членов арифметической прогрессии

$a,\ a+d,\ a+2d,\ \ldots\ a+(n-1)d$

Значения величин *a, d, n* вводить с клавиатуры.

Задание 2. Оператор цикла с предусловием.

В цикле с клавиатуры вводятся произвольные числа до тех пор, пока не будет введено отрицательное число. Посчитать, сколько введено положительных чисел.

Использование оператора *do while <логическое условие>*

Задание 3. Оператор цикла с постусловием.

Задать стороны прямоугольника *целыми* числами. На сколько квадратов можно его разрезать, если каждый раз отрезать квадрат максимально возможной площади (Рис.1). Использование оператора *do* совместно с оператором *if<логическое условие>exit* .

Примечание: Различие в выполнении Задания 2 и Задания 3 заключается в том, что логическое условие проверяется «до» или «после» выполнения вычислительного блока операций цикла.

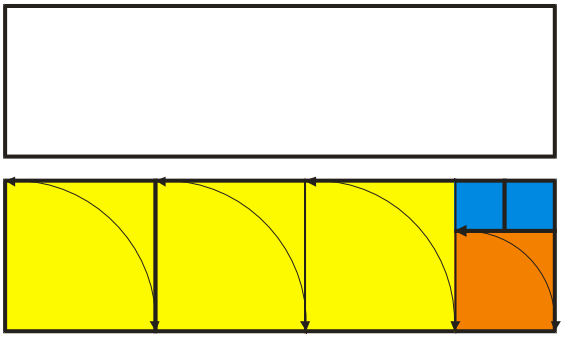


Рис.1. Схема выбора квадратов

Алгоритм задачи:

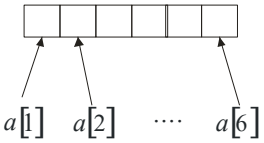
- Определяем наибольшую сторону.
- Определяем, сколько раз на ней может уместиться меньшая сторона.
- Для остатка большей стороны повторяется та же процедура.
- Цикл завершается, когда остаток становится равным нулю. (Остаток от деления целых чисел определяется оператором *mod* – большее/меньшее.)
- Ответ представить в виде (например, для a=17, b=4) –

  4 квадрата 4*4

  4 квадрата 1*1 всего 8 квадратов

Лабораторная работа № 6.Одномерные массивы.

Массив - набор данных, характерной чертой которого является то, что все компоненты его одного типа. Тип может быть любым. Элементы массива располагаются в памяти непрерывным блоком и имеют одно и то же имя.



Каждая из ячеек массива имеет свой порядковый номер. В данном примере нумерация идёт от 1 до 6. Элементы массива имеют имя – *a*. В принципе, нумерация тех же самых элементов массива может быть произведена в другом диапазоне индексов, например, – от 0 до 5, от -10 до -5 или от -2 до 3. Выбор индексов зависит от условий решаемой задачи.

Чтобы описать массив необходимо задать следующие параметры:

- количество элементов массива;

- тип элементов массива;
- нумерацию элементов.

Массив не является стандартным типом данных, поэтому он задаётся в части описаний.

Часто в вычислительных задачах используются случайные числа. Для их получения используется генератор случайных чисел (ГСЧ). Для работы ГСЧ необходимо задать некое стартовое число. По умолчанию оно всегда одно и то же. Поэтому от запуска к запуску программы последовательность случайных чисел повторяется. Для получения отличающихся последовательностей случайных чисел стартовое число нужно изменить.

Генератор случайных чисел.

В зависимости от версии Фортрана процедура формирования последовательности псевдослучайных чисел (ПСЧ) реализуется различными подпрограммами. В Gfortran'е получение СЧ реализуется следующим образом.

Подпрограмма RANDOM_NUMBER(*ran*) возвращает псевдослучайное число (или массив ПСЧ) из равномерно распределённого в интервале $0 \leq x < 1$. Тип *ran* – вещественный. Стартовая точка (затравка) для генератора случайных чисел устанавливается подпрограммой RANDOM_SEED(*[size],[put],[get]*).

Возможны два режима работы генератора:

- При очередном запуске программы выдаётся каждый раз новая последовательность случайных чисел.
- При очередном запуске программы выдаётся одна и та же последовательность случайных чисел.

Параметры подпрограммы RANDOM_SEED:

Size – стандартное целое число равное размеру *n* создаваемого процессором затравочного массива.

Put – стандартный целый массив, используемый процессором для изменения затравки.

Get - стандартный целый массив, в который заносятся текущие значения затравки.

При вызове RANDOM_SEED задаётся не более одного параметра.

Пример: Формирование двухмерных массивов вещественных и целых чисел в различных диапазонах значений.

```
real  bb(25,10)
integer   aa(25,10), cc(25,10)
! --------------старт генератора-------------------------------------
Integer   sv, ag(100), ap(100)
       sv=50
       call random_seed(size=sv)
       call random_seed(put=ap)
       call random_seed(get=ag)
! ---------------------------------------------------------
       do j=1,10
         do i=1,10
            call random_number(hav)
! генерация целых чисел в диапазоне от 0 до 1.
               aa(i,j)=anint(hav*10)
! генерация вещественных чисел в диапазоне от 0 до 10.
               bb(i,j)=hav
! генерация целых чисел в диапазоне от -1 до 1.
               cc(i,j)=anint(hav*2)-1
        enddo
      enddo
!----вывод сформированных массивов по заданному формату-------------
!--- Форматы см. на accel.ru → «Введение в Фортран» →«Печать текста и чисел на экран»
      write(*,10)  ((aa(i,j),i=1,10),j=1,10)
      write(*,11)  ((bb(i,j),i=1,10),j=1,10)
      write(*,10)  ((cc(i,j),i=1,10),j=1,10)
10   format(2x,10i8)
11   format(2x,10f10.8)
      read(*,*)
      end
```

Результат действий

```
 0  1  1  1 -1 -1 -1  1 -1  0
 0  0  0 -1  0  0 -1  0  1  0
 1  1 -1  1 -1  0  0  1  0  0
 1  1 -1  0  0  0 -1 -1  1  0
 1  0  0  0  0  0 -1  1  1 -1
 0  0  1  0  0  0 -1 -1 -1  1
 0 -1  0  1 -1  0 -1  0 -1  1
 0  0  0  1 -1  1  0  1  0  0
 1 -1  1 -1 -1  0  0  0  0  1
 0  0 -1  0  1  1 -1  0 -1  0
```

```
0.195  0.183  0.924  0.027  0.206  0.972  0.143  0.930  0.663  0.505
0.767  0.366  0.342  0.137  0.287  0.903  0.971  0.220  0.854  0.003
0.835  0.273  0.930  0.779  0.151  0.671  0.250  0.316  0.487  0.096
0.625  0.229  0.540  0.134  0.846  0.819  0.456  0.007  0.021  0.028
0.768  0.466  0.448  0.626  0.578  0.020  0.035  0.353  0.351  0.195
0.489  0.638  0.226  0.515  0.552  0.533  0.337  0.112  0.303  0.413
0.111  0.168  0.007  0.807  0.625  0.482  0.269  0.653  0.620  0.143
0.311  0.479  0.706  0.898  0.343  0.183  0.116  0.077  0.374  0.500
0.102  0.771  0.684  0.509  0.029  0.054  0.043  0.499  0.301  0.728
0.393  0.085  0.304  0.987  0.766  0.698  0.722  0.891  0.464  0.611
```

```
 6  5  4  5  8 10  7  6  5  9
 6  1  6  2  7  2  0  1  2  7
 3  4  9  5  8  4  5  1  2  0
10  0  3  5  3  3  4  9  4  1
 8  5  7  2  9  8  2  8  8  1
 3  5 10  9  0  1 10  4  1  1
 6  8  5 10  9  0  0  9  8  9
10  6  5  4  8  4  3  3  6  9
 7  7  3  1  2  5  9  0  3  5
 7  8 10  4  3  4  3  9  5  7
```

Для генерации при очередном запуске программы одной и той же последовательности случайных чисел обращения к подпрограмме

```
call random_seed(put=ap)
call random_seed(get=ag)
```

нужно исключить.

Задание 1.

Используя генератор случайных чисел, создайте массив из 10 элементов целого типа и вычислите их сумму.

Задание 2.

Используя генератор случайных чисел, создайте массив из *N* элементов целого типа. Найдите элементы, имеющие минимальное и максимальное значения, запомните их местоположение в массиве (порядковые индексы) и вычислите количество элементов массива, которые расположены между ними. Для определения наибольшего элемента введите дополнительную переменную (например, *maxel*) и занесите в неё значение первого элемента массива. В цикле последовательно сравнивайте элементы массива с содержимым переменной *maxel*. Если какой-либо из очередных элементов имеет большее значение, чем *maxel*, то это значение нужно занести в *maxel,* заменив прежнее значение. Одновременно нужно запоминать местоположение наибольшего элемента в массиве (т.е. его порядковый номер). Аналогично определите наименьший элемент массива.

Задание 3.

Используя генератор случайных чисел, создайте массив из *N* элементов целого типа в диапазоне от -100 до +100. Вычислите количество положительных и отрицательных чисел.

Лабораторная работа № 7. Многомерные массивы.

Определение многомерного массива аналогично определению одномерного массива.
Чтобы описать массив необходимо задать следующие параметры:

- количество элементов массива по каждому измерению;
- тип элементов массива;
- нумерацию элементов по каждому измерению.

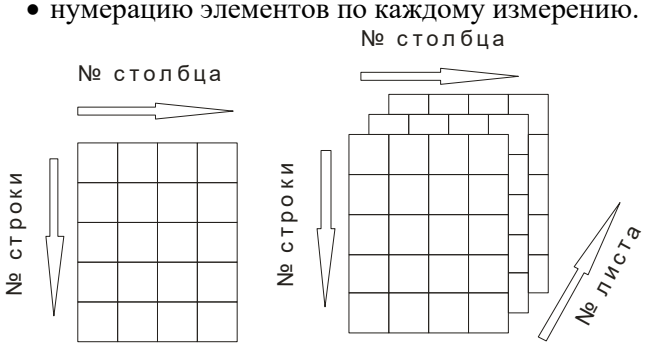


Рис.2. Двумерный и трёхмерный массивы

При работе с многомерными массивами, как правило, используются структуры, построенные на вложенных циклах. Например, так можно заполнить двухмерный массив вещественными числами от 0 до 1, используя генератор случайных чисел.

Организация вычислений в структурах вложенных циклов следующая:

1. Выполнятся оператор внешнего цикла – задаётся начальное значение переменной цикла.
2. Для текущего значения переменной внешнего цикла выполняется тело внутреннего цикла в указанных пределах изменения переменной внутреннего цикла.
3. Возвращаемся к внешнему циклу, изменяем значение переменной внешнего цикла на 1 (шаг).
4. Вновь переходим к выполнению тела внутреннего цикла.
5. И т.д. до тех пор, пока не будет исчерпан весь диапазон значений переменой внешнего цикла.

Задание 1.

Заполните массив размерностью 5×6 вещественными числами (положительными и отрицательными). Найдите минимальный и максимальный по модулю элементы массива. Запомните их координаты. Массив вывести на экран в виде таблицы.

Задание 2.

Заполните массив размерностью 10×10 целыми числами в диапазоне от -5 до 5. Найдите нулевые элементы массива. Запомнить их координаты. Массив вывести на экран в виде таблицы.

Задание 3.

Сформируйте матрицу размерностью $N \times N$ (предельный размер 10×10) в соответствии с приведённой ниже схемой заполнения ячеек целыми числами. Элементы главных диагоналей матрицы равны нулю. Остальные секторы матриц заполнить в соответствии с Рис.3. Рассмотрите матрицы с чётным и нечётным значением N. Полученные матрицы вывести на экран в виде двумерных таблиц.

0	1	1	1	1	0
3	0	1	1	0	4
3	3	0	0	4	4
3	3	0	0	4	4
3	0	2	2	0	4
0	2	2	2	2	0

0	1	1	1	1	1	0
3	0	1	1	1	0	4
3	3	0	1	0	4	4
3	3	3	0	4	4	4
3	3	0	2	0	4	4
3	0	2	2	2	0	4
0	2	2	2	2	2	0

Рис.3. Схема заполнения матрицы

Практический совет.

Прежде чем приступать к «программированию» (т.е. к написанию каких либо кодов) нужно выяснить какими характерными чертами обладает поставленная задача – структурными или математическими, физическими особенностями. На основе этого анализа можно приступать к разработке алгоритма. В предлагаемой задаче характерными особенностями является наличие четырёх секторов, следовательно, можно рассмотреть частные задачи заполнения секторов последовательностями одинаковых чисел.

При разработки алгоритмов Заданий-3,4,5 полезно иметь пред собой заполненные числами таблицы, что облегчит выбор нужных диапазонов переменных циклов.

Основная задача распадается на четыре подзадачи – заполнением по отдельности каждого из секторов по строкам (двойной цикл). Боковые сектора также нужно разделить на две части – верхнюю (от верхней строки сектора до самой длинной строки сектора) и нижнюю (оставшаяся часть сектора).

Задание 4.

Сформируйте матрицу размерностью $N \times N$ (предельный размер 10×10) в соответствии с приведённой ниже схемой заполнения ячеек (по диагоналям целыми числами в диапазоне от 1 до N^2 с шагом +1. Полученную матрицу вывести на экран в виде двумерной таблицы.

1	2	4	7	11	16
3	5	8	12	17	22
6	9	13	18	23	27
10	14	19	24	28	31
15	20	25	29	32	34
21	26	30	33	35	36

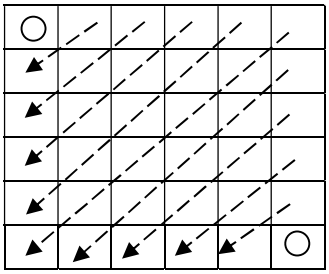


Рис.4. Схема заполнения матрицы

При формировании матрицы полезно рассматривать по отдельности две диагональные части – верхнюю, включающую в себя большую диагональ, и нижнюю.

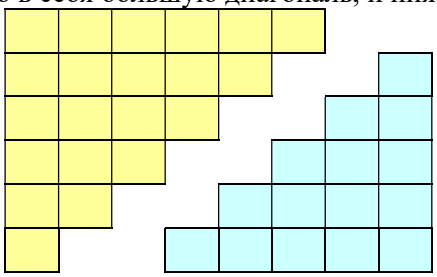


Рис.5. Схема заполнения матрицы

Возможны различные алгоритмы заполнения матрицы. Предварительно нужно проанализировать принцип изменения индексов элементов, входящих, например, в строку и определить индексы начального и конечного элемента строки. Рассмотрим верхний сектор, его верхнюю строку.

1	2	4	7	11	16
	1	*2*	*3*	*4*	*5*

.

Первый ряд – элементы первой строки верхнего сектора таблицы *a(i,j)*.

Второй ряд – разница между соседними элементами строки *r(j)*

Следовательно, алгоритм заполнения строки $a(i,j+1)=a(i,j)+r(j)$

По этому принципу рассмотрим и остальные строки. Следовательно, помимо основной матрицы необходимо сформировать дополнительный вектор *r(j)*.

По аналогии с верхним сектором нужно проанализировать нижний сектор массива *a(i,j)*.

Задание 5.

Сформируйте матрицу размерностью $N \times N$ в соответствии с приведённой ниже схемой заполнения ячеек целыми числами.

1	2	3	4	5	6	7
24	25	26	27	28	29	8
23	40	41	42	43	30	9
22	39	48	49	44	31	10
21	38	47	46	45	32	11
20	37	36	35	34	33	12
19	18	17	16	15	14	13

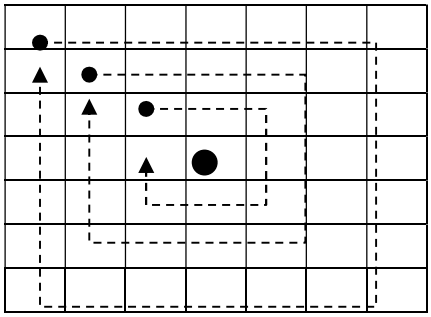


Рис.6. Схема заполнения матрицы

Рассмотрите матрицы с нечётным значением *N*. Порядок заполнения матрицы – скручивающаяся по часовой стрелке спираль. Начало спирали левый верхний угол матрицы – элемент с координатами (1,1). Конец спирали - центральный элемент матрицы. Предлагаемая схема заполнения матрицы основана на записи элементов матрицы по виткам спирали. Каждый виток спирали разделён на четыре участка – верхний горизонтальный, правый вертикальный, нижний горизонтальный, левый вертикальный. Нужно записать алгоритм изменения диапазона индексов на каждом витке спирали и на каждом участке отдельной спирали. Задача решается с использованием двойного цикла – внешний цикл определяется количеством витков спирали, внутренние циклы обеспечивают заполнение горизонтальных и вертикальных участков каждого витка спирали. Исходные значения переменных внутренних циклов определяются из задания размеров матрицы *a(i,j)*.

Лабораторная работа № 8.Алгоритмы сортировки.

Сортировка пузырьком

Расположим массив снизу вверх, от нулевого элемента - к последнему элементу.

Идея метода: шаг сортировки состоит в проходе снизу вверх по массиву. По пути прохода сравниваются значения пары соседних элементов. Если элементы некоторой пары находятся в неправильном порядке, то меняем их местами.

Рис.7. Схема продвижения «лёгкого» элемента к началу массива

После нулевого прохода по массиву «вверху» оказывается самый «легкий» элемент - отсюда аналогия с пузырьком. Следующий проход делается до второго сверху элемента, таким образом, второй по величине элемент поднимается на правильную позицию.

Делаем проходы по все уменьшающейся нижней части массива до тех пор, пока в ней не останется только один элемент. На этом сортировка заканчивается, так как последовательность упорядочена по возрастанию.

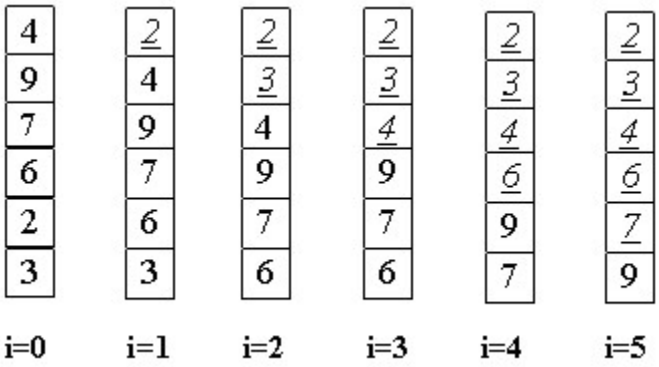


Рис.8. Схема сортировки элементов массива

Сортировка вставками

Будем разбирать алгоритм, рассматривая его действия на i-м шаге. Последовательность к этому моменту разделена на две части: готовую часть $a[0]...a[i]$ и неупорядоченную - $a[i+1]...a[n]$. На каждом последующем (i+1)-м шаге алгоритма берем $a[i+1]$ и вставляем на нужное место в готовую часть массива. Поиск подходящего места для очередного элемента входной последовательности осуществляется путем последовательных сравнений с элементом, стоящим перед ним. В зависимости от результата сравнения элемент либо остается на текущем месте (вставка завершена) либо они меняются местами и процесс повторяется.

Можно использовать и другой алгоритм – очередной элемент $a[i+1]$ заносим в буфер x и сравниваем значение x с содержимым уже отсортированной части массива до тех пор, пока не дойдём до элемента меньшего x или начала массива. Затем все элементы большие x сдвигаем на одну позицию вправо, а в освободившуюся ячейку заносим значение из буфера x. Этот алгоритм более эффективен, так как в нём меньше операций пересылки.

Пример: Последовательность на текущий момент. Часть a[0]..a[2] уже упорядочена (Рис.9.).

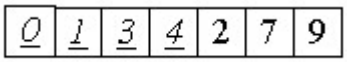


Рис.9. Последовательность элементов массива на текущий момент

Производим вставку числа 2 в отсортированную последовательность. Сравниваемые пары выделены.

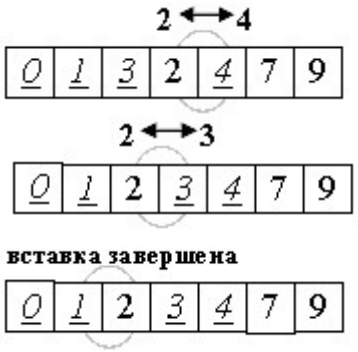


Рис.10. Схема вставки элемента массива на «нужное» место

Таким образом, в процессе вставки мы «просеиваем» элемент $a[i+1]$ к началу массива, останавливаясь в случае, когда

1. Найден элемент, меньший $a[i+1]$
   или
2. Достигнуто начало последовательности.

Задание 1.
1. Создайте массив *вещественных* (*real*) случайных чисел .
В полученном массиве найти и вывести на экран элементы с максимальным и минимальным значением, а также их индексы (порядковые номера) в исходном массиве.
3. Отсортируйте массив по возрастанию (убыванию), используя изложенные выше методы, и выведите все результаты на экран.

Задание 2. **Анализ больших массивов чисел**.

Предположим что, сформирован массив N случайных чисел в диапазоне [0-1]. Этот диапазон можно разделить на 10 поддиапазонов [0-0.1, 0.1-0.2, 0.2-0.3, … 0.9-1.0]. В каждый поддиапазон попадёт определённое количество чисел – n1, n2, n3, … n10. Создайте массив вещественных чисел, задав его размер с клавиатуры. Разделите диапазон значений чисел массива на 10 частей. Определите количество чисел массива, попавших в каждый поддиапазон. Рассмотрите массивы из 50, 500, 1000, 5000 чисел.

Для того чтобы определить, в какой поддиапазон попадает число можно воспользоваться следующим алгоритмом. Диапазон значений элементов массива от 0 до 1. Поддиапазоны имеют номера 1,2, .. 10. Определим номер поддиапазона, куда попадает, например, число 0,31567. Разделим это число на ширину поддиапазона 0,31567/0,1 = 3,1567, прибавим 1 и отбросим дробную часть. Номер поддиапазона - 4 (значение типа *integer*).

2.Численные алгоритмы

Численные алгоритмы – решение алгебраических и трансцендентных уравнений, решение систем линейных алгебраических уравнений, решение систем обыкновенных дифференциальных уравнений, решение уравнений в частных производных, оптимизация, обработка числовых данных. Из этого спектра алгоритмов рассмотрим наиболее простые алгоритмы типа «разделяй-и-властвуй». При сохранении общей идеологии в практической реализации алгоритмов существуют заметные различия.

Лабораторная работа № 9.

Поиск корня алгебраического и трансцендентного уравнения.

Задание 1. Деление отрезка пополам.

Функция $f(x)$ задана на отрезке $[a, b]$ и имеет только один корень ξ . Сначала проверяется условие – если $f((a+b)/2)=0$, то поиск завершён, иначе организуется итерационный процесс, на каждой итерации которого интервал поиска корня функции уменьшается вдвое. Критерием выбора той или иной половины рассматриваемого интервала для дальнейшего поиска является условие, что непрерывная функция $f(x)$ принимает значения разных знаков на концах подынтервалов либо $[a,\ a+(a+b)/2]$, либо $[a+(a+b)/2,\ b]$.

Алгоритм решения:

Полагая, что $b > a$ и ζ и δ - малые значения.

Шаг 1: Вычисляем значения функции $f(x)$ на концах интервала - $f(a)$, $f(b)$ **.** Проверяем условие наличия корня на интервале $[a, b]$ - если корень существует, то переходим к шагу 2, иначе – выход.

Шаг 2: Определяем середину интервала $x_{cp}=(a+b)/2$ и вычисляем в этой точке значение функции $f(x_{cp})$. Если $f(x_{cp})\neq 0$, то переходим к шагу 3, иначе – выход.

Шаг 3: Вычисляем произведение – $f(a)f(x_{cp})$

Если $f(a)f(x_{cp})<0$, то присваиваем $b = x_{cp}$, иначе присваиваем $a = x_{cp}$ и $f(a)=f(x_{cp})$

Шаг 4: Признак завершения счета. Если $abs(b-a)<\delta$, то перейти **к шагу 5**,
иначе перейти **к шагу 2**.

Шаг 5: Вычисляем $\xi=(a+b)/2$ и $f(\xi)$. Проверяем - если $f(\xi)<\zeta$, вычисления закончены, иначе уменьшаем значение δ и переходим к **шагу 2.**

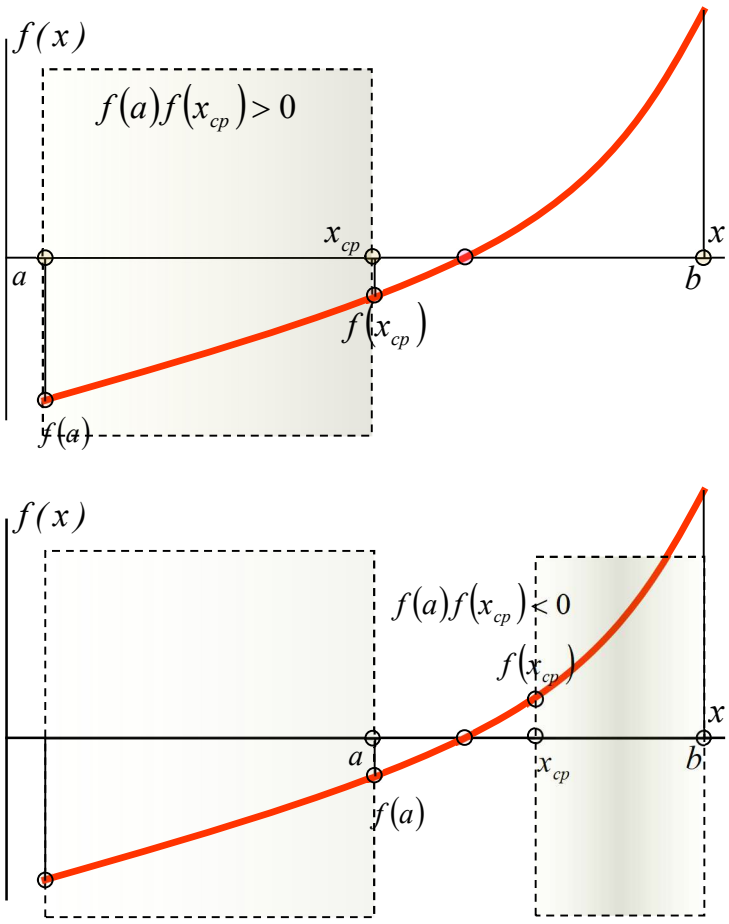


Рис. 21. Алгоритм «Деление отрезка пополам». Затенённые области исключаются из последующих итераций, границы поиска либо a, либо b перемещаются на место x_{cp} в зависимости от знака произведения $f(a)f(x_{cp})$.

Задание 2. Метод хорд.

Более быстрый способ нахождения корня ξ уравнения $f(x)=0$ на отрезке $[a,b]$ заключается в замене криволинейной функции $f(x)$ прямолинейным отрезком $A-B$ - хордой, соединяющей точки $A[a,f(a)]$ и $B[b,f(b)]$ (Рис.22). Уравнение прямой, проходящей через две точки $A[a,f(a)]$ и $B[b,f(b)]$ записывается как

$$\frac{x-a}{b-a}=\frac{f(x)-f(a)}{f(b)-f(a)} \quad (1)$$

Для доказательства сходимости процесса поиска корня, считаем, что он отделён и вторая производная $f''(x)>0$ сохраняет знак на отрезке $[a,b]$. Кривая функции $f(x)$ выпукла вниз и, следовательно расположена ниже своей хорды $A-B$ (Рис. 22).

Отсюда, полагая $f(x)=0$, получаем первое приближение корня

$$x_0=a-\frac{f(a)}{f(b)-f(a)}(b-a) \quad (2)$$

Затем вычисляем значение функции $f(x)$ в точке x_0 - $f(x_0)$ и определяем точку $A_1[x_0,f(x_0)]$. Через точки $A_1[x_0,f(x_0)]$ и $B[b,f(b)]$ проводим новую хорду A_1-B.

Получаем второе приближение корня

$$x_1=x_0-\frac{f(x_0)}{f(b)-f(x_0)}(b-x_0) \quad (3)$$

Далее повторяется процесс построения новых хорд и определения последующих приближений значений корня. Обобщая (3) можно записать

$$x_{i+1} = x_i - \frac{f(x_i)}{f(b) - f(x_i)}(b - x_i) \qquad (4)$$

Таким образом, строится монотонно возрастающая последовательность

$$a < x_0 < x_1 < \ldots < x_n < \xi < b$$

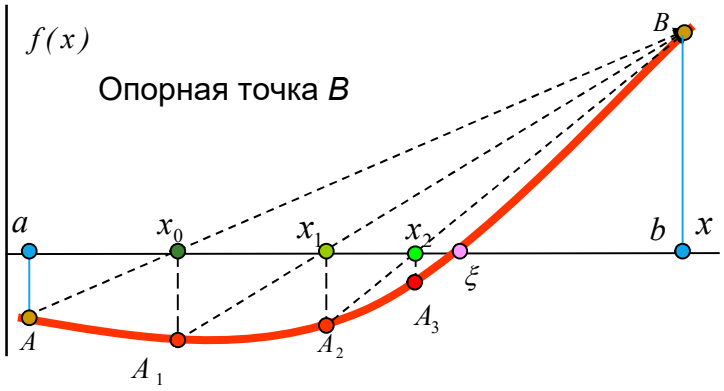


Рис.22. Поиск корня уравнения $f(x) = 0$. Алгоритм: метод хорд. Опорная точка B.

Приближенное значение корня уравнения $f(x) = 0$ определяем как

$$\bar{\xi} = \lim_{n \to \infty} x_n \qquad (5)$$

Алгоритм решения:

Полагая, что $b > a$ и ζ и δ - малые значения.

Шаг 1: Вычисляем значения функции $f(x)$ на концах интервала - $f(a)$, $f(b)$. Проверяем условие наличия корня на интервале $[a, b]$ - если корень существует, то переходим к шагу 2, иначе – выход.

Шаг 2: Вычисляем значения второй производной $f''(x)$ - $f''(a)$, $f''(b)$. Выбираем опорную точку.

Шаг 3: Используя уравнение хорды (3.2) и формулу (3.5) определяем приближённое значение корня x_0 и запоминаем его в ячейке x_{cm}.

Шаг 4: Используя уравнение хорды (1) и формулу (4) определяем приближённое значение корня x_0.

Шаг 5: Вычисляем $abs(x_{cm} - x_0)$.

Если $abs(x_{cm} - x_0) < \delta$ и $f(x_0) < \zeta$, то выход, иначе уменьшаем значение δ и переходим к **шагу 3.**

Шаг 6: Выход.

Для тестирования программы найдите корень уравнения $y = (x-1)^2 - 2$ на интервале [1, 4]. Результат - $x_0 = 2.414$.

Варианты функций

Аргумент *x* определен в диапазоне *[0÷1]*

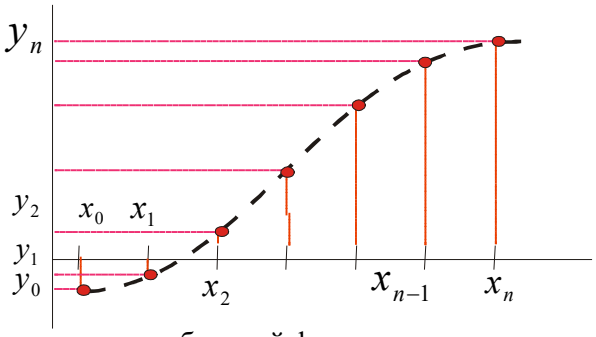
1. $f(x) = -\cos(\pi x / 2) + 2x$
2. $f(x) = \sin(\pi x / 2) - (1 - x)$
3. $f(x) = sh(x) - \cos(\pi x)$
4. $f(x) = 1 - x - tg(\pi x / 4)$
5. $f(x) = e^{-x} - \sin(\pi x / 2)$
6. $f(x) = (1 - x)^2 - 2x$
7. $f(x) = \arcsin(x) - (1 - x)^2$
8. $f(x) = \ln((1 + x) - (1 - x)$

Второй семестр

Приближение функций. Понятие приближения функций.

В своей практической деятельности мы часто сталкиваемся с ситуацией, когда те или иные параметры системы или объекта можно определить только в конечном числе точек (узлов). А для проведения численного моделирования необходимо знание значений параметров системы во всей области определения. Наиболее часто исходная функция $y = f(x)$ приближается многочленами.

$$f(x) \approx P_n(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n$$



X	Y
x_0	y_0
x_1	y_1
x_2	y_2
…	…
…	…
x_n	y_n

Рис.1. Представление табличной функции

В общем случае задачу можно сформулировать следующим образом: Данную функцию $f(x)$ требуется приближённо заменить (аппроксимировать) некоторой функцией $\varphi(x)$, так чтобы отклонение $\varphi(x)$ от $f(x)$ во всей области определения было наименьшим.

Интерполяция – приближённая замена табличной функции аналитической функцией**, значения которой в узлах таблицы совпадают со значениями табличной функции.**

Существуют несколько схем реализующих подобную замену. Они отличаются друг от друга как формой исходных таблиц (равноотстоящие узлы или произвольно расположенные), так и по способу выбора узлов, содержащих информацию, используемую для построения интерполирующей функции. Интерполирующая функция может строиться сразу для всего рассматриваемого интервала аргумента или отдельно для разных частей этого интервала.

Одним из основных понятий, связанных с табличными функциями, является понятие конечной разности.

Пусть $y = f(x)$- заданная функция, а $\Delta x = x_{i+1} - x_i = h = const$ - фиксированная величина приращения аргумента (шаг). Тогда выражение

$$\Delta y \equiv \Delta f(x) = f(x + \Delta x) - f(x)$$

называется первой конечной разностью функции y.

В приведённых формулах используется разностный оператор - Δ.

$\Delta x = x_{i+1} - x_i$ - разность значений переменной x в двух соседних узлах таблицы.

$\Delta y_i = y_{i+1} - y_i$ - разность значений табличной функции в двух соседних узлах таблицы.

Аналогично определяются конечные разности функции y высших порядков.

$$\Delta^2 y_i = \Delta y_{i+1} - \Delta y_i$$

$$\Delta^3 y_i = \Delta^2 y_{i+1} - \Delta^2 y_i$$

$$\Delta^4 y_i = \Delta^3 y_{i+1} - \Delta^3 y_i$$

.....

$$\Delta^n y_i = \Delta^{n-1} y_{i+1} - \Delta^{n-1} y_i$$

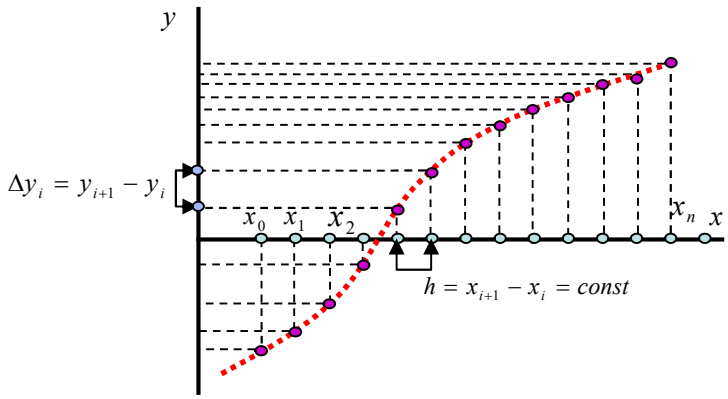


Рис.2. Графическое представление конечных разностей и узлов табличной функции.

Таблица 1. Табличная функция и конечные разности. Горизонтальная таблица.

Аргумент	**Функция**	**1-я к.раз**	**2-я к.раз**	**3-я к.раз**	**4-я к.раз**	**5-я к.раз**
x	y	Δy	$\Delta^2 y$	$\Delta^3 y$	$\Delta^4 y$	$\Delta^5 y$
x_0	y_0	Δy_0	$\Delta^2 y_0$	$\Delta^3 y_0$	$\Delta^4 y_0$	$\Delta^5 y_0$
x_1	y_1	Δy_1	$\Delta^2 y_1$	$\Delta^3 y_1$	$\Delta^4 y_1$	
x_2	y_2	Δy_2	$\Delta^2 y_2$	$\Delta^3 y_2$		
x_3	y_3	Δy_3	$\Delta^2 y_3$			
x_4	y_4	Δy_4				
x_5	y_5					

Из таблицы 1 видно, что разность более высокого прядка строится на основе разностей предшествующего порядка. Следует обратить внимание на индексацию разностей. Нижний индекс означает, к какому узлу таблицы принадлежит данная разность. Верхний индекс ($\Delta^i y_j$) означает порядок конечной разности (а не возведение в степень *i*). Для таблицы, содержащей n узлов можно построить конечные разности до (n-1) порядка включительно.

Горизонтальная таблица разностей используется при построении интерполяционного многочлена Ньютона (первая и вторая интерполяционные формулы Ньютона).

Лабораторная работа № 10.

Интерполирование.

Первая интерполяционная формула Ньютона.

$$P_n(x) = a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1) + a_2(x - x_0)(x - x_1)(x - x_2) + \ldots\ldots \quad (1)$$

где x_i - координаты узлов табличной функции, а a_i - коэффициенты, значения которых определяются исходя из значений табличной функции в соответствующих узлах. В зависимости от количества членов многочлена получаем различные приближения интерполирующей функции к табличной функции. Так первые два члена представляют линейную интерполяцию между двумя узлами (x_0, x_1), первые три члена – квадратичную интерполяцию, первые четыре члена – кубическую интерполяцию, и т.д. Если функция задана в $(n+1)$ узле, то можно построить многочлен n -ой степени.

Рассмотрим процесс вычисления коэффициентов a_i. Полагая в (1) $x = x_0$, получаем

$P_n(x_0) = y_0 = a_0$, т.е. $a_0 = y_0$.

Для определения a_1 вычислим конечную разность многочлена в точках x и $x + h$

$$P_n(x) = a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1) + a_3(x - x_0)(x - x_1)(x - x_2) + \ldots\ldots$$

$$P_n(x + h) = a_0 + a_1(x + h - x_0) + a_2(x + h - x_0)(x + h - x_1) + \quad (2)$$

$$+ a_3(x + h - x_0)(x + h - x_1)(x + h - x_2) + \ldots\ldots$$

Полагая в (2) $x = x_0$, получаем

$$\Delta P(x_0) = P_n(x+h) - P_n((x) \Rightarrow \Delta P(x_0) = a_1 * h\,,$$
$$\Delta P(x_0) = \Delta y_0 \Rightarrow \qquad a_1 = \Delta y_0 / (1!*h) \qquad (3)$$

Последовательно, продолжая этот процесс, получим

$$a_j = \Delta^i y_{n-1} / (i!*h^i) \qquad (4)$$

Следует отметить, что формулы (2,3) записаны для интерполяции на интервале $[x_0, x_1]$ для «опорного» нулевого узла x_0 . Для i -го узла в формулах нужно произвести замену индексов при переменных x_0 , x_1 , x_2 и т.д. на x_i , x_{i+1} , x_{i+2} и т.д.

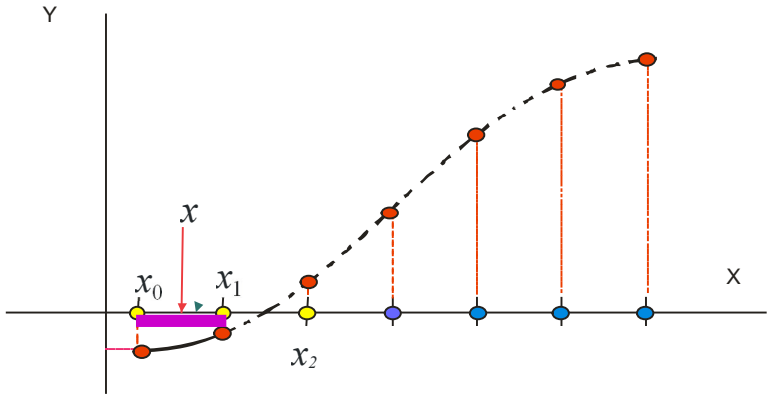


Рис. 3. Выбор узлов для квадратичной интерполяции на участке $[x_0, x_1]$.

Первой интерполяционной формуле Ньютона соответствуют конечные разности, расположенные в горизонтальных рядах Таблицы 1. Интерполирующая функция как бы «прижимается» к левой границе табличной функции. Для построения интерполирующей функции на правой границе табличной функции используется вторая интерполяционная формула Ньютона.

Задание 1.

Используя первую интерполяционную формулу Ньютона построить интерполяционный многочлен второй степени для таблично заданной функции.

Алгоритм решения.

Для тестовой функции $f(x) = \sin(x)$ в интервале $x = [x_0, x_n]$ построить табличную функцию $y(x)$, где $(n+1)$ – количество узлов, отстоящих друг от друга на величину h.

- В тестовых расчётах положить $x_0 = 0, x_n = 1,\ n \le 100$. *Индексы узлов начинаются с «0»*
- Вычислить первые и вторые конечные разности (*создать массивы*)
- Вычислить коэффициенты $a_0,\ a_1,\ a_2$ для каждого интервала (x_{i+1}, x_i) (*создать массивы*)

  Нижние индексы $a_0,\ a_1,\ a_2$ (см.табл.) относятся к коэффициентам полинома (2), верхние – к номеру соответствующего узла таблицы.

$a_0 = y_i \qquad a_1 = \Delta y_i / (1!*h) \qquad a_2 = \Delta^2 y_i / (2!*h^2)$

x	y	Δy	$\Delta^2 y$	a_0	a_1	a_2
x_0	y_0	Δy_0	$\Delta^2 y_0$	a_0^0	a_1^0	a_2^0
x_1	y_1	Δy_1	$\Delta^2 y_1$	a_0^1	a_1^1	a_2^1
x_2	y_2	Δy_2	$\Delta^2 y_2$	a_0^2	a_1^2	a_2^2
x_3	y_3	Δy_3	$\Delta^2 y_3$	a_0^3	a_1^3	a_2^3
x_4	y_4	Δy_4				
x_5	y_5					

- Сформировать массив значений аргумента $x_{i+1/2}$ в середине каждого интервала(x_{i+1}, x_i). *[Индексация $(i+1/2)$ означает только, что значение $x_{i+1/2}$ берётся в середине интервала (x_{i+1}, x_i), а сам массив имеет целочисленную индексацию]*
- Для этих значений аргумента вычислить значения тестовой функции $f(x) = \sin(x)$ и интерполяционного многочлена $P(x)$.
- Оценить разницу полученных значений $f(x)$ и $P(x)$. Результаты сравнений значений $f(x)$ и $P(x)$представить в виде таблиц
- Провести численный эксперимент для различного числа узлов. Как результат зависит от количества узлов в заданном интервале.

1) $y(x) = \sin^k(\pi x^m)$;
2) $y(x) = \cos^k(\pi x^m)$;
3) $y(x) = \sin^k(\pi x^{1/m})$;
4) $y(x) = \cos^k(\pi x^{1/m})$;
5) $y(x) = \mathrm{tg}^k(\pi x^m / 4)$;
6) $y(x) = \mathrm{tg}^k(\pi x^{1/m} / 4)$;
7) $y(x) = \pm a x^4 \pm b x^3$;
8) $y(x) = (a + b x^m)^{-k}$;
9) $y(x) = (a + b x^m)^{-1/k}$;
10) $y(x) = (a + b x^{1/m})^{-k}$;
11) $y(x) = (a + b x^{1/m})^{-1/k}$;
12) $y(x) = x^k / (a + b x^m)$;
13) $y(x) = x^k / (a + bx)^m$;
14) $y(x) = x^{1/k} / (a + bx)^m$;
15) $y(x) = x^k / (a + bx)^{1/m}$;
16) $y(x) = x^{1/k} / (a + bx)^{1/m}$;
17) $y(x) = (a + x^2)^k / (b + x^4)^m$;
18) $y(x) = (a + x^2)^{1/k} / (b + x^4)^m$;
19) $y(x) = (a + x^2)^k / (b + x^4)^{1/m}$;
20) $y(x) = (a + x^2)^{1/k} / (b + x^4)^{1/m}$;
21) $y(x) = x^k / (a + b x^m)$;
22) $y(x) = x^{1/k} / (a + b x^m)$;
23) $y(x) = x^k / (a + b x^{1/m})$;
24) $y(x) = x^{1/k} / (a + b x^{1/m})$;
25) $y(x) = \ln^k(1 + x^m)$;
26) $y(x) = \ln^{1/k}(1 + x^m)$;

Интерполяционная формула Лагранжа. (Глобальная интерполяция)

Для интерполирования табличной функции с неравноотстоящими узлами используется формула Лагранжа.

На отрезке $[x_0, x_n]$ даны $(n+1)$ различных значений аргумента - $x_0, x_1, x_2, \ldots x_n$ и известны значения функции $f(x)$ в этих узлах - $y_i = f(x_i)$. Нужно построить многочлен $L_n(x)$ степени не выше n, такой, что $L_n(x_i) = y_i$ для $i = 0, 1, 2, \ldots n$

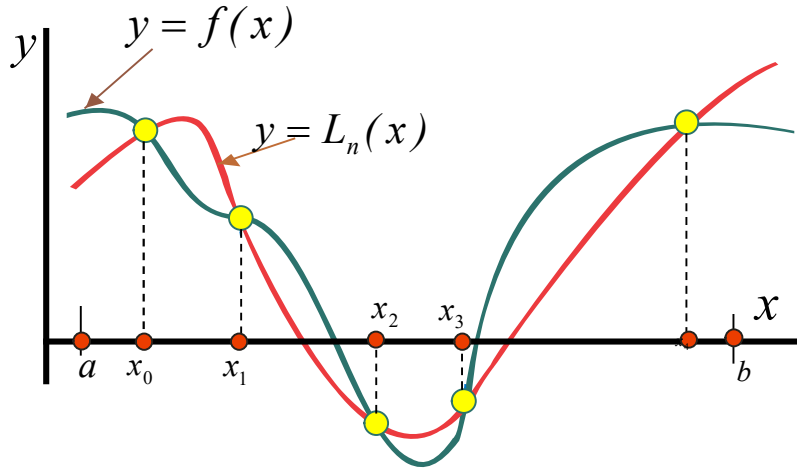


Рис. 4. Табличная функция и многочлен Лагранжа.

Для построения такого многочлена сначала нужно решить частную задачу - построить вспомогательные многочлены $p_i(x)$, такие, что $p_i(x_j) = 0$ при $j \neq i$ и $p_i(x_i) = 1$, т.е.

$$p_i(x_i) = \delta_{ij} = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases} \qquad (5)$$

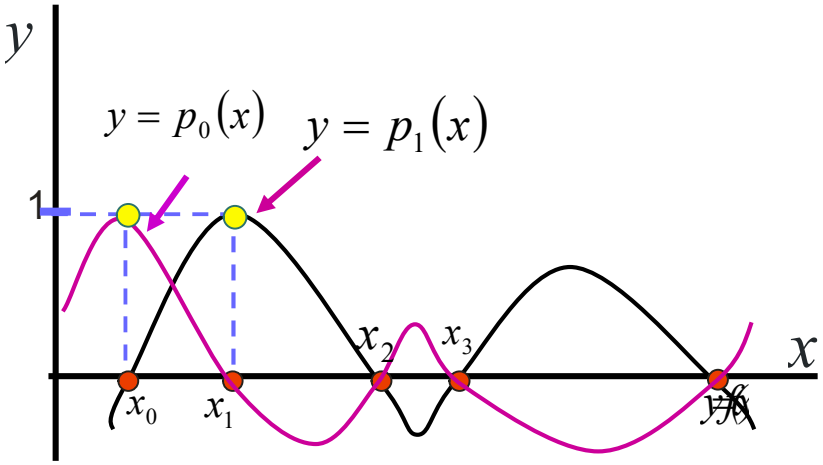


Рис.5. Вспомогательные многочлены.

Так как вспомогательный многочлен $p_i(x)$ обращается в нуль в n узлах (кроме *i-го*), то он может быть записан как

$$p_i(x) = C_i(x-x_0)(x-x_)(x-x_{2)})(x-x_3)\ldots(x-x_{i-1})(x-x_{i+1})\ldots(x-x_n) \quad (6)$$

где C_i - постоянный коэффициент.

Полагая в (6) $x = x_i$ и учитывая, что $p_i(x_i) = 1$, получаем

$$C_i = \frac{1}{(x_i-x_0)(x_i-x_)(x_i-x_2)(x_i-x_3)\ldots(x_i-x_{i-1})(x_i-x_{i+1})\ldots(x_i-x_n)} \quad (7)$$

и, следовательно, многочлен (6) запишется как

$$p_i(x) = \frac{(x-x_0)(x-x_1)(x-x_2)\ldots(x-x_{i-1})(x-x_{i+1})\ldots(x-x_n)}{(x_i-x_0)(x_i-x_)(x_i-x_2)\ldots(x_i-x_{i-1})(x_i-x_{i+1})\ldots(x_i-x_n)} \quad (8)$$

Чтобы выполнилось исходное требование $L_n(x_i) = y_i$, интерполяционный многочлен должен иметь вид

$$L_n(x) = \sum_{i=0}^{n} y_i \frac{(x-x_0)(x-x_1)(x-x_2)\ldots(x-x_{i-1})(x-x_{i+1})\ldots(x-x_n)}{(x_i-x_0)(x_i-x_1)(x_i-x_2)\ldots(x_i-x_{i-1})(x_i-x_{i+1})\ldots(x_i-x_n)} \quad (9)$$

Следует отметить, что интерполяционный многочлен Лагранжа, в отличие от других интерполяционных функций, содержит в явном виде значения y_i, что при решении некоторых задач может оказаться важным фактором.

Задание 2.

Используя интерполяционную формулу Лагранжа построить интерполяционный многочлен для таблично заданной функции.

Алгоритм решения.

Для тестовой функции $f(x) = \sin(x)$ в интервале $x = [x_0, x_n]$ построить табличную функцию $y(x)$, где n – количество узлов, отстоящих друг от друга на величину h. В общем случае расстояние между узлами может быть различным.

- В тестовых расчётах положить $x_0 = 0, x_n = 1,\ n \le 100$.
- Сформировать массив значений аргумента x_i.
- Для этих значений аргумента вычислить значения тестовой функции $f(x) = \sin(x)$.
- Сформировать массив значений аргумента $(x1)_i$ в середине каждого интервала (x_{i+1}, x_i).
- Для этих значений аргумента вычислить значения тестовой функции $f(x) = \sin(x)$.
- Построить интерполяционный многочлен для всего интервала $x = [x_0, x_n]$

1. Построить вспомогательные многочлены (8) для этого использовать алгоритм трёх вложенных циклов.

$$p_i(x) = \frac{(x-x_0)(x-x_1)(x-x_2)\ldots(x-x_{i-1})(x-x_{i+1})\ldots(x-x_n)}{(x_i-x_0)(x_i-x_1)(x_i-x_2)\ldots(x_i-x_{i-1})(x_i-x_{i+1})\ldots(x_i-x_n)}$$

Для каждого значения x (всего их $k=(n-1)$) из массива значений аргумента $(x1)_i$ построить n вспомогательных многочленов $p_i((x1)_i)$

Внешний цикл – перебор значения $(x1)_i$ по индексам от 0 до $(n-1)$

2. Внутренний цикл - перебор значения x_i по индексам от 0 до (n) и формирование матрицы вспомогательных многочленов $r(k,i)$ размерностью $(n-1)\times n$
3. Самый внутренний цикл – вычисление произведений в числителе и знаменателе (8) с перебором значений x_j в сомножителях $(x_i - x_j)$.

- Вычислить значения многочлена (9) в серединах интервалов (x_{i+1}, x_i), используя алгоритм двух вложенных циклов.

Студентам: Постарайтесь сначала написать собственный алгоритм.
Если это вызывает сложности, воспользуйтесь приведённым кодом.

```
real x(0:100),y(0:100),x1(0:100),y1(0:100)
   real r(0:100,0:100),p(0:100)

   write(*,*) 'vvesti a b n'
   read(*,*) a,b,n
! ----- шаг таблиц, аргумент, функция ----------------
   h=(b-a)/n
   do i=0,n
     x(i)=h*i
     y(i)=f(x(i))
   enddo
 ! ----- аргумент, функция в серединах интервалов----------------
   do i=0,n-1
     x1(i)=h*i+h/2
     y1(i)=f(x1(i))
   enddo
  !----реализация формулы (9)
!------ L_n(x) = sum_{i=0}^{n} y_i (x-x_0)(x-x_1)(x-x_2)...(x-x_{i-1})(x-x_{i+1})...(x-x_n) / (x_i-x_0)(x_i-x_1)(x_i-x_2)...(x_i-x_{i-1})(x_i-x_{i+1})...(x_i-x_n)   (9)
    do k=0,n-1    !--цикл по x ≡ x_1
      do i=0,n    !--цикл по x_i
        u=1
        d=1
        do j=0,n !--цикл по x_j в сомножителях: числитель (x-x_j) и знаменатель (x_i-x_j)
          if (i.ne.j) then
            u=u*(x1(k)-x(j))
            d=d*(x(i)-x(j))
          endif
        enddo
        r(k,i)=u/d
      enddo
    enddo
!-----значение многочлена (9) , вычисление суммы
    do k=0,n-1
      sum=0
      do i=0,n
        sum=sum+y(i)*r(k,i)
      enddo
```

```
          p(k)=sum
        enddo
!-------------------------------------------------
        write(*,*) '    x        y       x1       y1'
        write(*,10) (x(i),y(i),x1(i),y1(i),i=0,n)
        write(*,*)  '       p              y1'
        write(*,11)  (p(j),y1(j),j=0,n-1)
10      format(4(x,f8.5))
11      format(2(x,e14.7))
        end
!------тестовая функция-------------------
        real function f(x)
        real x
          f=sin(x)
         return
         end
!=================================================
```

Численное дифференцирование.

Для численного определения значений производных можно использовать представление их через конечные разности

$$y' \approx \Delta y / \Delta x \quad (10)$$

Это представление является *аппроксимацией производной с помощью конечных разностей.*

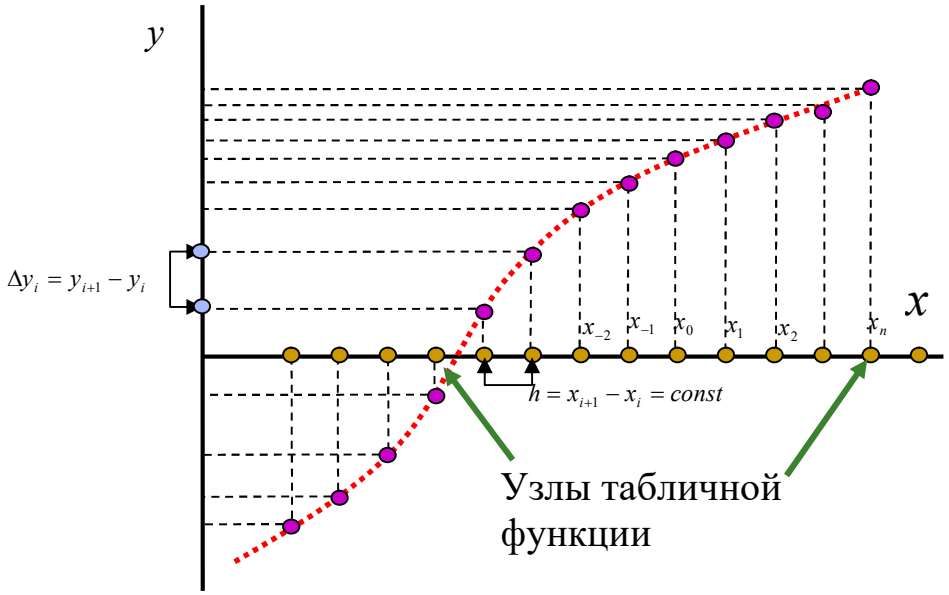


Рис.6. Графическое представление табличной функции.

Для табличной функции возможны представления производной с помощью левых разностей, правых разностей и центральных разностей.

Рассмотрим табличную функцию $f(x_i)$, определённую на сетке с постоянным шагом $\Delta x = h$ (Рис.6), первая конечная разность $\Delta y_i = y_i - y_{i-1}$. Определим первую производную как

$$y'_i = \frac{(y_i - y_{i-1})}{h} \quad (10)$$

Таким образом, производную в точке x_i выразили через разность значений функции в этой точке y_i и в точке слева от нее y_{i-1}, т. е. через левые разности. Аналогично можно поступить, выразив производную с помощью правых разностей

$$y'_i = \frac{(y_{i+1} - y_i)}{h} \quad (11)$$

или с помощью центральных разностей

$$y_i' = \frac{(y_{i+1} - y_{i-1})}{2h} \quad (12)$$

По такому же принципу можно определить производные более высокого порядка:

$$y_i'' = \frac{(y_{i+1}' - y_{i-1}')}{h} = \frac{y_{i+1} - 2y_i + y_{i-1}}{h^2} \quad (13)$$

Такие же соотношения можно получить с помощью ряда Тейлора

$$f(x + \Delta x) = f(x) + f'(x)\Delta x + \frac{1}{2!} f''(x)(\Delta x)^2 + \frac{1}{3!} f'''(x)(\Delta x)^3 + ... \quad (14)$$

Пусть функция $f(x)$ записана в виде таблицы

$$f(x_i) = y_i \qquad (i = 1,2,3,...n)$$

При $x = x_1$, $\Delta x = -h$ ряд Тейлора с точностью до членов порядка h^2 запишется как

$$y_0 = y_1 - y_1' h + O(h^2) \quad (15)$$

Отсюда производная в точке $x = x_1$ определяется как

$$y_1' = \frac{y_1 - y_0}{h} + O(h) \quad (16)$$

Это выражение совпадает с ранее полученным выражением аппроксимации первой левосторонней производной первого порядка $(r = 1)$. Также можно получить выражение для правосторонней производной.

Рассмотрим выражения для первой и второй производных второго порядка погрешности.

$$\begin{aligned} y_2 &= y_1 + y_1' h + \frac{1}{2!} y_1'' h^2 + \frac{1}{3!} y_1''' h^3 + O(h^4) \\ y_0 &= y_1 - y_1' h + \frac{1}{2!} y_1'' h^2 - \frac{1}{3!} y_1''' h^3 + O(h^4) \end{aligned} \quad (17)$$

Вычитая из первого уравнения второе, получим

$$y_1' = \frac{y_2 - y_0}{2h} + O(h^2) \quad (18)$$

что совпадает с аппроксимацией центральными разностями. Погрешность имеет второй порядок.

Складывая уравнения, оценим погрешность аппроксимации производной второго порядка

$$y_1'' = \frac{y_2 - 2y_1 + y_0}{h^2} + O(h^2) \quad (19)$$

Данное выражение также совпадает с аппроксимацией центральными разностями.

Для численного определения производных можно использовать различные интерполяционные полиномы с различным количеством членов. На практике более удобным оказывается использование полиномов Лагранжа, т.к. в них в явном виде присутствуют значения табличной функции в узлах. В зависимости от количества используемых узлов получаем выражения для производных функции $f(x)$ с различной погрешностью вычислений.

Рассмотренный источник погрешностей – погрешность аппроксимации, которая определяется величиной остаточного члена. При уменьшении шага таблиц h эта погрешность, как правило, уменьшается.

Другие погрешности – неточность значений функции $y = f(x)$ в узлах и погрешность округления вычислений. Эти погрешности возрастают при уменьшении шага h.

Полином Лагранжа имеет вид (9)

$$L_n(x) = \sum_{i=0}^{n} y_i \frac{(x - x_0)(x - x_1)(x - x_2)....(x - x_{i-1})(x - x_{i+1})....(x - x_n)}{(x_i - x_0)(x_i - x_1)(x_i - x_2)....(x_i - x_{i-1})(x_i - x_{i+1})....(x_i - x_n)}$$

Рассмотрим случай трёх узлов интерполирования ($n = 2$).

Расписав многочлен как сумму трёх (соответствующих) слагаемых и приведя их к общему знаменателю, для таблиц с постоянным шагом получим следующие выражения для многочлена и первой производной

$$L(x)=\frac{1}{2h^2}\left[(x-x_1)(x-x_2)y_0-2(x-x_0)(x-x_2)y_1+(x-x_0)(x-x_1)y_2\right]$$
$$L'(x)=\frac{1}{2h^2}\left[(2x-x_1-x_2)y_0-2(2x-x_0-x_2)y_1+(2x-x_0-x_1)y_2\right] \quad (20)$$

где $y'''(x_*)$ - значение производной третьего порядка в некоторой внутренней точке $x_* \in [x_0, x_n]$

Запишем значения производных в точках x_0, x_1, x_2

$$y_0'=\frac{1}{2h}(-3y_0+4y_1-y_2)+\frac{h^2}{3}y'''(x_*)$$
$$y_1'=\frac{1}{2h}(y_2-y_0)-\frac{h^2}{6}y'''(x_*) \quad (21)$$
$$y_2'=\frac{1}{2h}(y_0-4y_1+3y_2)+\frac{h^2}{3}y'''(x_*)$$

Подобным образом можно вычислить производные для любого количества узлов аппроксимации. При четных значениях n наиболее простые выражения получаются для центральных точек. Вариант с чётным значением n - *аппроксимация с помощью центральных разностей.* Он более удобен и при вычислении вторых производных. Для четырёх узлов производные

$$y_0'=\frac{1}{6h}(-11y_0+18y_1-9y_2+2y_3)$$
$$y_1'=\frac{1}{6h}(-2y_0-3y_1+6y_2-y_3)$$
$$y_2'=\frac{1}{6h}(y_0-6y_1+3y_2+2y_3) \quad (22)$$
$$y_3'=\frac{1}{6h}(-2y_0+9y_1-18y_2+11y_3)$$

Для пяти узлов производные

$$y_0'=\frac{1}{12h}(-25y_0+48y_1-36y_2+16y_3-3y_4)$$
$$y_1'=\frac{1}{12h}(-3y_0-10y_1+18y_2-6y_3+y_4)$$
$$y_2'=\frac{1}{12h}(y_0-8y_1+8y_3-2y_4) \quad (23)$$
$$y_3'=\frac{1}{12h}(-y_0+6y_1-18y_2+10y_3+3y_4)$$
$$y_4'=\frac{1}{12h}(3y_0-16y_1+36y_2-48y_3+25y_4)$$

Аналогичным образом можно получить выражения для вторых производных, дважды продифференцировав соответствующий многочлен Лагранжа.

Для трёх узлов

$$y_0''=(y_0-2y_1+y_2)/h^2$$
$$y_1''=(y_0-2y_1+y_2)/h^2$$
$$y_2''=(y_0-2y_1+y_2)/h^2$$

Для четырёх узлов

$$y_0''=(2y_0-5y_1+4y_2-y_3)/h^2$$

$$y_1'' = (y_0 - 2y_1 + y_2)/h^2$$
$$y_2'' = (y_1 - 2y_2 + y_3)/h^2$$
$$y_3'' = (-y_0 + 4y_1 - 5y_2 - 2y_3)/h^2$$

Для пяти узлов

$$y_0'' = (35y_0 - 104y_1 + 114y_2 - 56y_3 + 11y_4)/12h^2$$
$$y_1'' = (11y_0 - 20y_1 + 6y_2 + 4y_3 - y_4)/12h^2$$
$$y_2'' = (-y_0 + 16y_1 - 30y_2 + 16y_3 - y_4)/12h^2$$
$$y_3'' = (-y_0 + 4y_1 + 6y_2 - 20y_3 + 11y_4)/12h^2$$
$$y_4'' = (11y_0 - 56y_1 + 114y_2 - 104y_3 + 35y_4)/12h^2$$

Задание 1.

Используя интерполяционную формулу Лагранжа построить интерполяционный многочлен для таблично заданной функции.

Алгоритм решения.

Для тестовой функции $f(x) = \sin(x)$ в интервале $[x_0, x_n]$ построить табличную функцию $y(x)$, где $(n+1)$ – количество узлов, отстоящих друг от друга на величину h. В тестовых расчётах положить $x_0 = 0, x_n = 1,\ n = 10$.

- Сформировать массив значений аргумента x_i. Индекс i от 0 до n.
- Сформировать массив значений функции $y(x_i)$.
- Используя интерполяционную формулу Лагранжа для таблиц с постоянным шагом записать выражения производных в схемах с четырьмя $(n = 3)$ и пятью $(n = 4)$ узлами. Вычислить значения производных в этих узлах. Сравнить значения производных в одноименных узлах между собой и тестовой функцией.
- Для узла с индексом «1» вычислить значения левосторонней, центральной и правосторонней производных и сравнить их со значениями, полученными из многочлена Лагранжа.
- Провести численный эксперимент с варьированием величины шага h.
- Вычислить значения вторых производных в схемах с четырьмя и пятью узлами.
- Результаты работы представить в соответствующей форме, удобной для анализа.

==== вариант =======

=0-й= =1-й= =2-й= =3-й= =4-й=

==== тест_1 первые производные

0.1000000E+01 0.9950042E+00 0.9800666E+00 0.9553365E+00 0.9210610E+00

==== 4 узла

0.1000030E+01 0.9949925E+00 0.9800798E+00 0.9552920E+00

==== 5 узлов

0.9999807E+00 0.9950089E+00 0.9800633E+00 0.9553415E+00 0.9210409E+00

==== 2_л = 2_ц = 2 _п =

0.9983342E+00

0.9933466E+00

0.9883591E+00

==== тест _2 вторые производные

0.0000000E+00 -0.9983342E-01 -0.1986693E+00 -0.2955202E+00 -0.3894183E+00

==== 4 узла

-0.1000613E-02 -0.9975135E-01 -0.1985021E+00 -0.2972528E+00

==== 5 узлов

0.8106232E-03 -0.9991601E-01 -0.1986668E+00 -0.2954416E+00 -0.3902406E+00

Лабораторная работа № 12.

Решение СЛАУ

Способы решения систем линейных алгебраических уравнений можно разделить на две группы:

1. Точные методы – представляют собой конечные алгоритмы для вычисления корней системы (правило Крамера, метод Гаусса, метод главных элементов, метод квадратных корней и др.)
2. Итерационные методы – построенные на сходящихся бесконечных процессах, обеспечивающих заданную точность вычисления корней системы (метод итераций, метод Зейделя, метод релаксации и др.)

Полученные результаты в обоих случаях являются приближёнными из-за округления численных результатов и погрешностей методов.

Рассмотрим систему n линейных уравнений с n неизвестными

$$\begin{aligned} &a_{11}x_1 + a_{12}x_2 + ... + a_{1n}x_n = b_1 \\ &a_{21}x_1 + a_{22}x_2 + ... + a_{2n}x_n = b_2 \\ &..... \\ &a_{n1}x_1 + a_{n2}x_2 + ... + a_{nn}x_n = b_n \end{aligned} \quad (24)$$

Введем обозначения:

- матрица коэффициентов $A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix}$
- вектор-столбец правых частей системы и вектор-столбец неизвестных

$$b = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}, \quad x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}.$$

Система (24) в матричном виде запишется как

$$Ax = b \quad (25)$$

Необходимым и достаточным условием существования единственного решения системы является условие неравенства нулю детерминанта матрицы A.

$$D = Det\ A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix} \neq 0 \qquad (26)$$

В противном случае система называется *вырожденной* и может иметь либо множество решений, либо не иметь ни одного решения.

При $D \approx 0$ система является *плохо обусловленной*. Это условие является необходимым, но недостаточным для определения плохо обусловленной системы.

Прямые методы – расчёт ведётся по конечным соотношениям (формулам) для определения неизвестных. Число операций заранее известно. Достоинством прямых методов является их относительная простота и универсальность. Недостатком является необходимость хранить в памяти компьютера сразу всю матрицу, что при большой размерности системы приводит к неэффективности вычислений, особенно, если матрица содержит много нулевых элементов – в этом случае производятся ненужные операции с нулями. Другим недостатком является накопление погрешности результатов, т.к. последующие операции используют данные, полученные на предшествующих операциях (это особенно опасно для плохо обусловленных матриц).

Итерационные методы – методы последовательных приближений. Для работы процедуры необходимо задать начальное приближение, т.е. задать произвольное значение вектору неизвестных. Это стартовое значение позволяет провести один цикл вычислений (итерацию). Затем вычисления повторяются до достижения нужной точности результатов. Алгоритмы итерационных методов, как правило, более сложные, чем алгоритмы прямых методов. Их достоинством является

отсутствие необходимости хранить в памяти машины целиком все матрицы, достаточно хранить несколько векторов с n компонентами. Погрешности вычислений не накапливаются, т.к. на текущей итерации используются результаты только предшествующей итерации, а не всех предшествующих вычислений. Более того, случайный сбой вычислений означает, что расчёт начинается с нового произвольного начального приближения.

Прямые методы решения СЛАУ.
Метод Гаусса

Этот метод основан на приведении матрицы к треугольному виду. Это достигается последовательным исключением неизвестных из уравнений системы. С помощью первого уравнения исключается x_1 из всех последующих уравнений. Преобразованная система содержит $(n-1)$ уравнение относительно $x_2, x_3 \ldots x_n$ неизвестных. Процесс повторяется - с помощью первого уравнения преобразованной системы исключается x_2 из всех последующих уравнений этой системы. Процесс продолжается до получения одного уравнения относительно неизвестного x_n. Таким образом, из исходной системы и преобразованных систем уравнений формируется матрица треугольного вида. Этот процесс называется *прямым ходом метода Гаусса.*

Например, для системы из трёх уравнений

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 &= b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 &= b_2 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 &= b_3 \end{aligned} \qquad (27)$$

Формирование матриц в процессе прямого хода можно отобразить следующим образом
Разделив первое уравнение системы (27) на a_{11}, получим уравнение

$$x_1 + \frac{a_{12}}{a_{11}}x_2 + \frac{a_{13}}{a_{11}}x_3 = \frac{b_1}{a_{11}} \qquad (28)$$

которое умножим на a_{21}

$$a_{21}x_1 + a_{21}\frac{a_{12}}{a_{11}}x_2 + a_{21}\frac{a_{13}}{a_{11}}x_3 = a_{21}\frac{b_1}{a_{11}} \qquad (29)$$

Вычтем из второго уравнения (27) уравнение (29), получим

$$\left(a_{22} - a_{21}\frac{a_{12}}{a_{11}}\right)x_2 + \left(a_{23} - a_{21}\frac{a_{13}}{a_{11}}\right)x_3 = b_2 - a_{21}\frac{b_1}{a_{11}} \qquad (30)$$

Умножим уравнение (28) на a_{31} и вычтем это уравнение из третьего уравнения (27)

$$\left(a_{32} - a_{31}\frac{a_{12}}{a_{11}}\right)x_2 + \left(a_{33} - a_{31}\frac{a_{13}}{a_{11}}\right)x_3 = b_3 - a_{31}\frac{b_1}{a_{11}} \qquad (31)$$

В результате этих действий пришли к системе из двух уравнений, не содержащих переменную x_1

$$\begin{aligned} a_{22}^{(1)}x_2 + a_{23}^{(1)}x_3 &= b_2^{(1)} \\ a_{32}^{(1)}x_2 + a_{33}^{(1)}x_3 &= b_3^{(1)} \end{aligned} \qquad (32)$$

где $a_{22}^{(1)} = \left(a_{22} - a_{21}\frac{a_{12}}{a_{11}}\right)$, $a_{23}^{(1)} = \left(a_{23} - a_{21}\frac{a_{13}}{a_{11}}\right)$, (33)

$$a_{32}^{(1)} = \left(a_{32} - a_{31}\frac{a_{12}}{a_{11}}\right), \quad a_{33}^{(1)} = \left(a_{33} - a_{31}\frac{a_{13}}{a_{11}}\right),$$

Проведём подобное преобразование для системы из двух уравнений (32) и получим одно уравнение с одним неизвестным x_3.

$$a_{33}^{(2)}x_3 = b_3^{(2)} \qquad (34)$$

На этом шаге процедура прямого хода заканчивается, треугольная матрица сформирована.

$$\begin{matrix} a_{11} & a_{12} & a_{13} & b_1 \\ & a_{22}^{(1)} & a_{23}^{(1)} & b_2^{(1)} \\ & & a_{33}^{(2)} & b_3^{(2)} \end{matrix} \qquad (35)$$

В процессе формирования треугольной матрицы (исключение неизвестных) производится деления на коэффициенты $a_{11}, a_{22}^{(1)}$ и т.д. Поэтому они должны быть отличными от нуля. В случае необходимости нужно переставить уравнения системы местами, чтобы выполнить это требование. На этом принципе исключения переменных основаны несколько алгоритмов.

Приведённый ниже алгоритм не требует хранения промежуточных данных (пересчёта коэффициентов промежуточных этапов).В общем виде вычисление коэффициентов можно записать как

$$a_{ij}^{(k)} = a_{ij}^{(k-1)} - \frac{a_{ik}^{(k-1)}}{a_{kk}^{(k-1)}} a_{kj}^{(k-1)} \qquad (36)$$

$$b_i^{(k)} = b_i^{(k-1)} - \frac{a_{ik}^{(k-1)}}{a_{kk}^{(k-1)}} b_k^{(k-1)}$$

На следующем этапе производится вычисление неизвестных, начиная с x_n

В результате обратного хода

$$x_k = \frac{1}{a_{kk}^{(k-1)}} \left(b_k^{(k-1)} - \sum_{j=k+1}^{n} a_{kj}^{(k-1)} x_j\right) \qquad (37)$$

Последовательно вычисляются все неизвестные. Этот алгоритм часто называют схемой единственного деления, а элемент $a_{kk}^{(k-1)}$ - ведущим элементом.

Задание .

Используя зависимости (36) и (37) найти решение тестовой системы уравнений

$$\begin{aligned} 1.000x_1 + 0.001x_2 + 0.001x_3 + 0.001x_4 + 0.001x_5 &= 1.00 \\ 0.001x_1 + 2.000x_2 + 0.001x_3 + 0.001x_4 + 0.001x_5 &= 2.00 \\ 0.001x_1 + 0.001x_2 + 3.000x_3 + 0.001x_4 + 0.001x_5 &= 3.00 \\ 0.001x_1 + 0.001x_2 + 0.001x_3 + 4.000x_4 + 0.001x_5 &= 4.00 \\ 0.001x_1 + 0.001x_2 + 0.001x_3 + 0.001x_4 + 5.000x_5 &= 5.00 \end{aligned}$$

Алгоритм решения. (псевдокод).

Для $k = 1, 2, \ldots, n-1$ (*цикл последовательного исключения переменных*)

  Для $i = k+1, \ldots, n$

    $t_{ik} := a_{ik}/a_{kk}$

    $b_i := b_i - t_{ik} b_k$

    Для $j = k+1, \ldots, n$

      $a_{ij} := a_{ij} - t_{ik} a_{kj}$

$x_n = b_n/a_{nn}$

  Для $k = n-1, \ldots, 2, 1$

$$x_k = \left(b_k - \sum_{j=k+1}^{n} a_{kj} x_j\right)/a_{kk}$$

Студентам: Постарайтесь сначала написать собственный алгоритм.

Если это вызывает сложности, воспользуйтесь приведённым кодом.

Код программы

```
real a(5,5),t(5,5),b(5),x(5)
write(*,*) 'vvesti n'
read(*,*) n
do i=1,n
   do j=1,n
```

```
            if (i.eq.j)  then
              a(i,j)=i
            else
              a(i,j)=0.1
            endif
          enddo
          b(i)=i
        enddo
        write(*,*) 'исходная матрица a(n*n)  и вектор b(n)'
        write(*,10)  ((a(i,j),j=1,5),b(i),i=1,5)
        do k=1,n-1
          do i=k+1,n
            t(i,k)=a(i,k)/a(k,k)
            b(i)=b(i)-t(i,k)*b(k)
            do j=k+1,n
              a(i,j)=a(i,j)-t(i,k)*a(k,j)
            enddo
          enddo
c       промежуточная печать на k-ой итерации
        write(*,*) ' матрица a(n*n)  и вектор b(n)  k=',k
        write(*,10)  ((a(i,j),j=1,5),b(i),i=1,5)
        enddo
c       прямой ход
        write(*,*) 'результирующая матрица и вектор'
        write(*,10)  ((a(i,j),j=1,5),b(i),i=1,5)
c       обратный ход
        x(n)=b(n)/a(n,n)
        sum=0
        do k=n-1,1,-1
          do j=k+1,n
            sum=sum+a(k,j)*x(j)
          enddo
          x(k)=(b(k)-sum)/a(k,k)
        enddo
        write(*,*) 'результат'
        write(*,10)  (x(k),k=1,n)
10      format (5e12.5,4x,e12.5)
        end
```

Результаты расчёта

```
Ввести n
5
        'исходная матрица a(n*n)  и вектор b(n)'
0.10000E+01  0.10000E+00 0.10000E+00 0.10000E+00 0.10000E+00    0.10000E+01
 0.10000E+00 0.20000E+01 0.10000E+00 0.10000E+00 0.10000E+00    0.20000E+01
 0.10000E+00 0.10000E+00 0.30000E+01 0.10000E+00 0.10000E+00    0.30000E+01
 0.10000E+00 0.10000E+00 0.10000E+00 0.40000E+01 0.10000E+00    0.40000E+01
 0.10000E+00 0.10000E+00 0.10000E+00 0.10000E+00 0.50000E+01    0.50000E+01

матрица a(n*n)  и вектор b(n)  k=1
 0.10000E+01 0.10000E+00 0.10000E+00 0.10000E+00 0.10000E+00   0.10000E+01
 0.10000E+00 0.19900E+01 0.90000E-01  0.90000E-01 0.90000E-01    0.19000E+01
 0.10000E+00 0.90000E-01 0.29900E+01  0.90000E-01 0.90000E-01    0.29000E+01
 0.10000E+00 0.90000E-01 0.90000E-01  0.39900E+01 0.90000E-01    0.39000E+01
```

0.10000E+00 0.90000E-01 0.90000E-01 0.90000E-01 0.49900E+01 0.49000E+01

матрица a(n*n) и вектор b(n) k=2
0.10000E+01 0.10000E+00 0.10000E+00 0.10000E+00 0.10000E+00 0.10000E+01
0.10000E+00 0.19900E+01 0.90000E-01 0.90000E-01 0.90000E-01 0.19000E+01
0.10000E+00 0.90000E-01 0.29859E+01 0.85930E-01 0.85930E-01 0.28141E+01
0.10000E+00 0.90000E-01 0.85930E-01 0.39859E+01 0.85930E-01 0.38141E+01
0.10000E+00 0.90000E-01 0.85930E-01 0.85930E-01 0.49859E+01 0.48141E+01

матрица a(n*n) и вектор b(n) k=3
0.10000E+01 0.10000E+00 0.10000E+00 0.10000E+00 0.10000E+00 0.10000E+01
0.10000E+00 0.19900E+01 0.90000E-01 0.90000E-01 0.90000E-01 0.19000E+01
0.10000E+00 0.90000E-01 0.29859E+01 0.85930E-01 0.85930E-01 0.28141E+01
0.10000E+00 0.90000E-01 0.85930E-01 0.39835E+01 0.83457E-01 0.37331E+01
0.10000E+00 0.90000E-01 0.85930E-01 0.83457E-01 0.49835E+01 0.47331E+01

матрица a(n*n) и вектор b(n) k=4
0.10000E+01 0.10000E+00 0.10000E+00 0.10000E+00 0.10000E+00 0.10000E+01
0.10000E+00 0.19900E+01 0.90000E-01 0.90000E-01 0.90000E-01 0.19000E+01
0.10000E+00 0.90000E-01 0.29859E+01 0.85930E-01 0.85930E-01 0.28141E+01
0.10000E+00 0.90000E-01 0.85930E-01 0.39835E+01 0.83457E-01 0.37331E+01
0.10000E+00 0.90000E-01 0.85930E-01 0.83457E-01 0.49817E+01 0.46549E+01

результирующая матрица и вектор
0.10000E+01 0.10000E+00 0.10000E+00 0.10000E+00 0.10000E+00 0.10000E+01
0.10000E+00 0.19900E+01 0.90000E-01 0.90000E-01 0.90000E-01 0.19000E+01
0.10000E+00 0.90000E-01 0.29859E+01 0.85930E-01 0.85930E-01 0.28141E+01
0.10000E+00 0.90000E-01 0.85930E-01 0.39835E+01 0.83457E-01 0.37331E+01
0.10000E+00 0.90000E-01 0.85930E-01 0.83457E-01 0.49817E+01 0.46549E+01
результат
0.17575E+00 0.71283E+00 0.86303E+00 0.91757E+00 0.93439E+00

Комментарии к результатам расчёта по данному алгоритму

- Алгоритм не требует дополнительной памяти, т.к. не сохраняет матрицы промежуточных преобразований матриц и векторов (используется область исходной матрицы и вектора).
- На очередной итерации преобразуются элементы матриц, выделенные зеленым цветом. Строки, полученные на предшествующей итерации, не изменяют своих значений.
- В результирующей матрице для вычисления значений x_i используются только элементы верхней диагональной части (чёрный цвет). Элементы нижней части (красный цвет) – это «отходы производства».

Итерационные методы.
Метод простой итерации.

При большом числе уравнений метод Гаусса может оказаться достаточно сложным. В этом случае более удобно пользоваться не точными, а приближёнными численными методами. Систему уравнений $Ax = b$ представим в развёрнутом виде

$$
\begin{aligned}
&a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1\\
&a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n = b_2\\
&\quad\ldots\ldots\ldots\ldots\ldots\ldots\\
&a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n = b_n
\end{aligned}
\qquad A = \begin{bmatrix} a_{11}a_{12}\cdots a_{1n} \\ a_{21}a_{22}\cdots a_{2n} \\ \ldots\ldots\ldots\ldots \\ a_{n1}a_{nn}\cdots a_{nn} \end{bmatrix} \quad b = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} \quad x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \qquad (38)
$$

Полагая $a_{ii} \neq 0 \quad (i = 1,2,\ldots,n)$ решим первое уравнение относительно x_1, второе — относительно x_2 и т. д. Получаем эквивалентную систему

$$
\begin{aligned}
x_1 &= \beta_1 + \alpha_{12}x_2 + \alpha_{13}x_3 \ldots + \alpha_{1n}x_n\\
x_2 &= \beta_2 + \alpha_{21}x_1 + \alpha_{23}x_3 \ldots + \alpha_{2n}x_n\\
&\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\\
x_n &= \beta_n + \alpha_{n1}x_1 + \alpha_{n2}x_2 \ldots + \alpha_{n,n-1}x_{n-1}
\end{aligned}
\qquad (39)
$$

где $\beta_i = \dfrac{b_i}{a_{ii}}, \quad \alpha_{ij} = -\dfrac{a_{ij}}{a_{ii}} \qquad i \neq j \qquad (40)$

и

$\alpha_{ij} = 0$ при $i = j, \quad (i, j = 1,2,..n)$

Введя матрицы

$$
\alpha = \begin{bmatrix} \alpha_{11}\alpha_{12}\cdots\alpha_{1n} \\ \alpha_{21}\alpha_{22}\cdots\alpha_{2n} \\ \ldots\ldots\ldots\ldots \\ \alpha_{n1}\alpha_{nn}\cdots\alpha_{nn} \end{bmatrix} \quad \text{и} \quad \beta = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_n \end{bmatrix} \qquad (41)
$$

и систему уравнений можно записать в матричном виде

$$x^{(0)} = \beta + \alpha x \qquad (42)$$

Эта система решается методом последовательных приближений. За нулевое приближение принимается, например, столбец свободных членов $x^{(0)} = \beta$.

Затем строится первое приближение

$$x^{(1)} = \beta + \alpha x^{(0)} \qquad (43)$$

потом — второе приближение

$$x^{(2)} = \beta + \alpha x^{(1)} \qquad (44)$$

и т. д. Общий алгоритм решения

$$x^{(k+1)} = \beta + \alpha x^{(k)} \qquad (45)$$

В результате получаем последовательность $x^{(0)}, x^{(1)}, \ldots, x^{(k)} \ldots$, предел которой (если он существует)

$$x = \lim_{k\to\infty} x^{(k)}$$

является решением системы $\lim\limits_{k\to\infty} x^{(k+1)} = \beta + \lim\limits_{k\to\infty} x^{(k)}$

Формулы приближений можно представить в виде

$$x_i^{(0)} = \beta_i$$

$$x_i^{(k+1)} = \beta_i + \sum_{j=1}^{n} \alpha_{ij} x_j^{(k)} \qquad (46)$$

$(\alpha_{ii} = 0; i = 1,\ldots,n; k = 0,1,2,\ldots)$

Задание .
Систему уравнений из предшествующего задания решить итерационным методом и сравнить результаты.
Алгоритм.

- Сформировать матрицы $\boldsymbol{A}$ и b.
- Сформировать матрицы $\boldsymbol{\alpha}$ и β.
- За нулевое приближение принять столбец свободных членов $x^{(0)} = \beta$.
- В цикле организовать процесс (46). Завершение процесса – значения вектора x на двух соседних итерациях отличаются на малую величину ε (точность вычислений)

Лабораторная работа № 13.

Нахождения минимума функции.

Задание 1.

Алгоритм реализуется в задачах оптимизации тремя способами – методом дихотомии, методом деления отрезка пополам и золотым сечением. Различие заключается в быстродействии, которое в данном случае определяется количеством обращений к вычислению минимизируемой функции F_{min}.

Сравнить методы по быстродействию – количеству обращений к вычислению функции. Произвести вычисления при различных значениях точности вычисления. В последующих заданиях значения варьируемых параметров (границы интервала поиска минимума функции и точность вычисления) задавать с клавиатуры, используя операторы ввода данных.

Дихотомия

Наиболее простой алгоритм, в котором на каждой итерации интервал поиска минимума функции *сокращается ровно вдвое*. Вычисление значения минимизируемой функции на каждой итерации производится дважды. Суть алгоритма заключается в следующем – на текущей итерации определяется x_{cp} середина отрезка $[a,b]$ и вычисляются значения двух близко лежащих точек $x_1 = x_{cp} - \varepsilon$ и $x_2 = x_{cp} + \varepsilon$, где ε наперёд заданное малое значение аргумента функции (Рис.7). Далее сравниваются значения функции в точках x_1 и x_2.

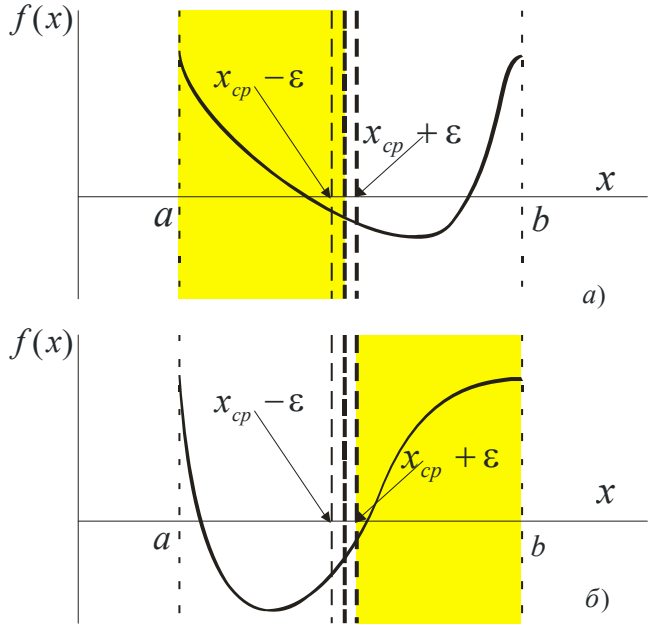


Рис.7. Сокращение интервала поиска методом дихотомии

Пусть функция $f(x)$ задана на отрезке $[a,b]$ и имеет только один минимум. Параметры ε, δ – малые значения.

Шаг1: Определить середину интервала $x_{cp}=(a+b)/2$, выделить две точки на оси аргументов $x_1 = x_{cp} - \varepsilon$ и $x_2 = x_{cp} + \varepsilon$.

Шаг2: Вычислить значения функции $f(x_1)$ и $f(x_2)$.

Шаг3: Сравнить значения функции $f(x_1)$ и $f(x_2)$.

Если $f(x_1) < f(x_2)$, то присваиваем $b = x_{cp}$, иначе $a = x_{cp}$.

Шаг4: Признак завершения счета. Если $abs(b-a)< \delta$, то перейти к шагу 5, иначе перейти к шагу 1.
Шаг5: Вычислить $x_{min}=(a+b)/2$ и $f(x_{min})$

Для тестирования программы найдите минимум параболы $y=(x-1)^2+3$ на интервале [-1, 3]. Результат - $x_{min}=1$.

Варианты функций

1. $f(x)=-x^3+3x^2+9x+10 \qquad x\in[-3,3]$
2. $f(x)=(2x+1)^2+(x-4) \qquad x\in[-2,4]$
3. $f(x)=(10-x)^2 \qquad x\in[-10,20]$
4. $f(x)=x^3-12x+3 \qquad x\in[-2,5]$
5. $f(x)=(10x^3+3x^2+x+5)^2 \qquad x\in[-2,2]$
6. $f(x)=3x^4+(x-1)^2 \qquad x\in[-2,2]$
7. $f(x)=4x*sin(x) \qquad x\in[-\pi/4,\pi/2]$
8. $f(x)=2(x-3)^2+e^{x/2} \qquad x\in[0,4]$

Деление отрезка пополам.

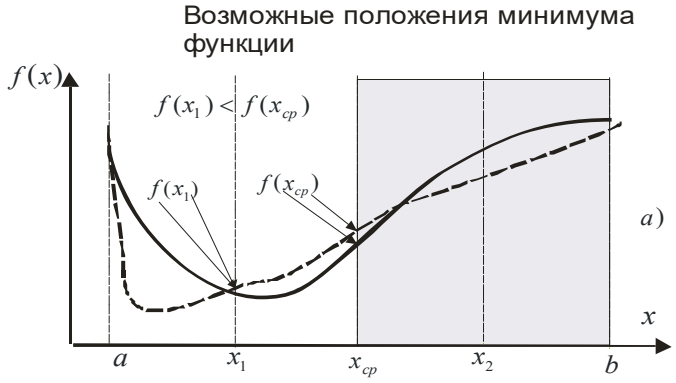
Метод, в котором на каждой итерации интервал поиска минимума функции *сокращается вдвое*. Он основан на выборе трёх пробных точек, расположенных равномерно на интервале поиска минимума целевой функции. На текущей итерации определяется x_{cp} середина отрезка $[a,b]$ и значения двух симметрично расположенных точек $x_1=x_{cp}-abs(b-a)/4$ и $x_2=x_{cp}+abs(b-a)/4$. Далее сравниваются значения функции в точках x_{cp}, x_1, и x_2. Можно определить, какую часть интервала поиска можно исключить на следующей итерации.

На Рис.8. приведены возможные положения расположения минимума целевой функции, которые могут возникнуть в процессе сокращения интервала поиска. Заштрихованные области исключаются из дальнейшего рассмотрения.

При $f(x_1)>f(x_{cp})$ – правую половину (Рис.8а.), при $f(x_1)<f(x_{cp})$ - левую половину (Рис.8б.) и при $f(x_2)>f(x_{cp})$ - левую и правую четверти (Рис.8в.). Соответствующим образом переносятся границы интервала поиска $a\rightarrow x_{cp}$ (Рис.8а.) или $b\rightarrow x_{cp}$ (Рис.8б.) или $a\rightarrow x_1$ и $b\rightarrow x_2$ (Рис.8в.) и формируется очередной интервал $[a,b]$.

Поиск завершается, когда интервал поиска $[a,b]$ сокращается до заданной малой величины δ.

На первом шаге поиска производится три обращения к вычислению целевой функции в точках x_{cp}, x_1, и x_2 на последующих шагах поиска – два обращения (в точках x_1, и x_2) Общее количество обращений к вычислению целевой функции определяется как $(2*N+1)$, где N – число шагов поиска.



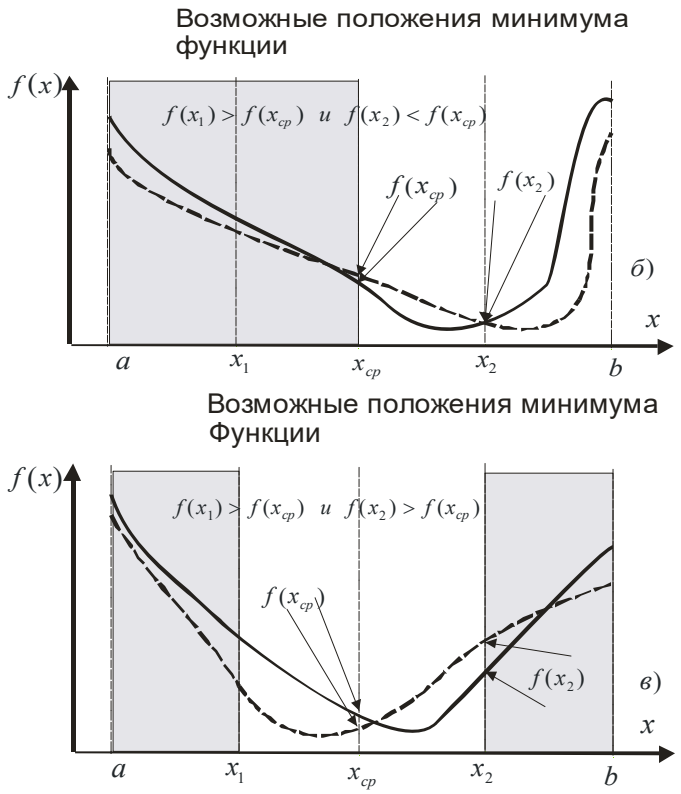


Рис.8. Возможные положения минимумов целевой функции.

Алгоритм поиска.

Пусть функция $f(x)$ задана на отрезке $[a,b]$ имеет только один минимум. δ – малое значения.

Шаг1: Определить середину интервала $x_{cp}=(a+b)/2$. Вычислить в этой точке значение функции $fx_{cp}= f(x_{cp})$.

Шаг2: Определить точки $x_1= x_{cp} – abs(b-a)/4$ и $x_2= x_{cp} + abs(b-a)/4$.

Шаг3: Вычислить значения функции $fx_1= f(x_1), fx_2 =f(x_2)$.

Шаг4: Сравнить значения функции fx_1 и fx_{cp}.

Если $fx_1 < fx_{cp}$, то присвоить $b= x_{cp}$, $x_{cp}=x_1$, $fx_{cp}= fx_1$, вычислить точки $x_1= x_{cp} – abs(b-a)/4$ и $x_2= x_{cp} + abs(a)/4$,

перейти к шагу 6. Иначе перейти к шагу 5.

Шаг5: Сравнить значения функции fx_2 и fx_{cp}.

Если $fx_2 < fx_{cp}$, то присвоить $a= x_{cp}$, $x_{cp}=x_2$, $fx_{cp}= fx_2$,

вычислить точки $x_1= x_{cp} – abs(b-a)/4$ и $x_2= x_{cp} + abs(b-a)/4$, перейти к шагу 6. Иначе присвоить $a= x_1$, $b= x_2$, вычислить точки $x_1= x_{cp} – abs(b-a)/4$ и $x_2= x_{cp} + abs(b-a)/4$

Шаг6: Завершения счета. Если $abs(b-a)< \delta$, то перейти к шагу 7, иначе перейти к шагу 3.

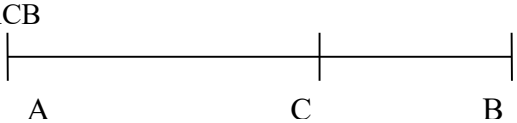
Шаг7: Вычислить $x_{min}=(a+b)/2$ и $f(x_{min})$

Золотое сечение.

Из рассмотрения выше приведенных методов можно сделать выводы позволяющие повысить эффективность поиска:

1. Если количество пробных точек равно двум, то их следует размещать на одинаковых расстояниях от середины интервала.
2. В соответствии с общей минимаксной стратегией пробные точки должны размещаться по симметричной схеме таким образом, чтобы отношение длины исключаемого подынтервала к величине интервала поиска оставалось постоянным.
3. На каждой итерации поиска должна вычисляться только одна пробная точка и значение целевой функции в этой точке.

Геометрическая интерпретация принципа метода золотого сечения заключается в следующем. Дана прямая АСВ



Отношение отрезков прямой определяется соотношением AC/AB=CB/AC, т.е. отношение большего отрезка к целому равно отношению меньшего отрезка к большему отрезку. Если принять целый отрезок AB=1, то $AC = (-1+\sqrt{5})/2$. Приближённо AC≈0.62, а CB≈0.38.

Количество обращений к вычислению целевой функции определяется как (N+1), где N – число шагов поиска.

Пусть функция *f(x)* задана на отрезке [*a,b*] и имеет только один минимум. Параметр δ – малое значение.

Шаг1: Задать коэффициенты *λ2*=0.62, *λ1*=0.38, интервал *[a,b]* и длину отрезка *l=abs(b-a)*.
Шаг2: Выделить две точки *v=a+l* λ1* и *w=a+l* λ2* .
Шаг3: Если abs(*b-a)* < *δ* , то перейти к шагу 7,
иначе перейти к шагу 4.
Шаг4: Вычислить в этих точках значения функции *fv=f(v)* и *fw=f(w)*.
Шаг5: Сравнить значения функции *fw* и *fv*.
Если *fw* < *fv*, то присвоить *a=v, v=w, fv =fw* и вычислить *w=a+abs(b-a)* λ2*, *f(w)*. Перейти к шагу 3.
иначе перейти к шагу 6.
Шаг6: Присвоить *b=w, w=v, fw =fv* и вычислить *v=a+abs(b-a)* λ1*, *f(v)*. Перейти к шагу 3.
Шаг7: Вычислить $x_{min}=(a+b)/2$ и $f(x_{min})$.

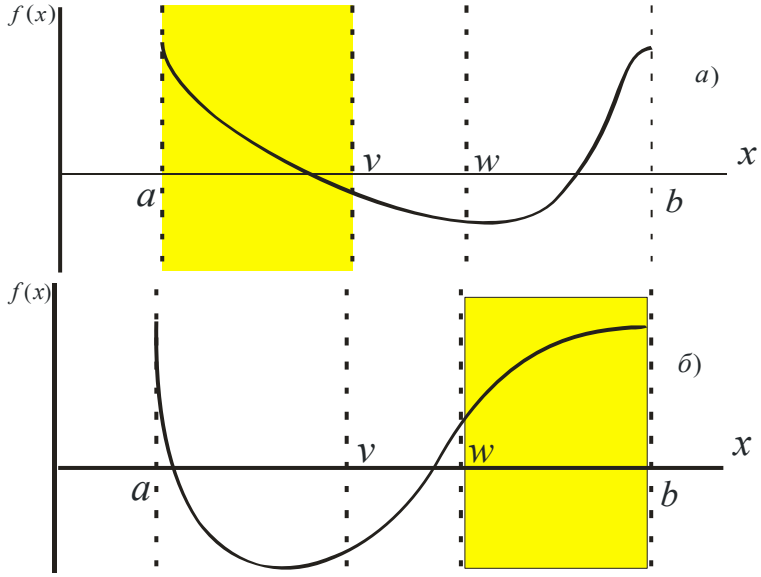


Рис.9. Возможные положения минимума целевой функции.

Пусть длина исходного интервала поиска L_1, а конечного интервала - L_n. Эффективность метода можно оценить по отношению L_n/L_1. В методах дихотомии и деления отрезка пополам длина конечного интервала составляет $L_1(0{,}5)^{N/2}$, а в методе золотого сечения - $L_1(0{,}618)^{N-1}$. При заданной точности вычислений ε количество обращений к вычислению целевой функции в двух первых методах составляет $N = 2\ln(\varepsilon)/\ln(0{,}5)$, а для метода золотого сечения $N = 1+\ln(\varepsilon)/\ln(0{,}618)$.

Задание.

- Реализовать приведённые алгоритмы и проверить для тестовой функции.
- Провести расчёт для одного из приведённых вариантов функций.
- Сравнить приведённые методы поиска минимума функции одной переменной по эффективности - количеству обращений к вычислению самой функции при одинаковой точности расчётов.