Оглавление

Введение

Данное пособие представляет собой вторую часть пособия, посвященного изложению типовых алгоритмов программирования и решения основных инженерных задач. В данной части продолжается рассмотрение принципов построения алгоритмов циклической структуры. Авторы знакомят читателя с приемами вычисления сумм, произведений, наибольшего и наименьшего значений функции. Решение этих задач требует от программиста умения организовывать циклы с заранее известным числом повторений.

Однако существует целый ряд задач, решение которых требует организации циклов с заранее неизвестным числом повторений. К задачам этого типа относятся задачи вычисления суммы бесконечного ряда и уточнения корней уравнений с заданной точностью. В пособии подробно рассмотрены вопросы решения каждой из этих задач. В частности, приводятся примеры вычисления очередного слагаемого ряда в соответствии с рекуррентной формулой, непосредственно по общей формуле члена ряда и с использованием смешанной схемы.

Решение задачи уточнения корней уравнений иллюстрируется реализацией трех наиболее популярных методов: простой итерации, половинного деления, касательных.

Отдельный раздел пособия посвящен рассмотрению вопросов организации вложенных циклов. Авторы уделили изложению этого материала достаточно много внимания, так как решение многих практических задач требует умения реализовывать алгоритмы как раз подобной структуры. Рассматривается решение задачи вычисления значения определенного интеграла с заданной точностью. При этом авторы затронули также вопрос сокращения количества выполняемых операций.

В этом же разделе рассмотрены вопросы нахождения наибольшего (наименьшего) значения функции на заданном интервале с заданной точностью.

Большое внимание уделено вопросу обработки матриц, так как инженеру часто в своей повседневной практике приходится сталкиваться с этим математическим объектом. Приводятся примеры программной реализации решения различных задач с матрицами.

Заканчивается раздел рассмотрением алгоритмов сортировки элементов массивов. Авторы уделили внимание изложению наиболее простых для понимания алгоритмов, с которыми знакомятся студенты при изучении дисциплины Информатика. К числу таких алгоритмов относятся алгоритмы сортировки 1)методом прямого включения (вставки), 2)прямого обмена (пузырька), 3)выбора.

Отличительной особенностью данного пособия является краткое изложение теоретического материала, сопровождаемое достаточным количеством примеров, что должно позволить студенту за ограниченное время освоить правила и алгоритмы программирования.

Для закрепления изучаемого материала и приобретению необходимого опыта программирования инженерных задач авторами предлагаются по каждой теме комплекты индивидуальных заданий, выполнение которых послужит решению поставленных задач. Задания имеют индивидуальный характер, могут использоваться при проведении лабораторных работ, подготовке к рубежным контролям знаний, а также в качестве заданий при проведении контрольных мероприятий.

4. Программы циклической структуры (продолжение, см. предыдущую часть [1])

4.9. Приёмы вычисления сумм, произведений и экстремальных значений

Вычисление суммы и произведения

При решении многих научно-технических задач часто приходится вычислять суммы и произведения. Например, при вычислении статистических параметров необходимо вычислить среднее арифметическое и среднее геометрическое значений некоторой функции. Чтобы вычислить указанные величины, надо сначала определить сумму и произведение значений функции. Сформулируем условие этой задачи. Пусть задана некоторая функция y=f(x). Вычислить среднее арифметическое и среднее геометрическое значений функции yi=f(xi), вычисляемых в равноотстоящих точках интервала изменения аргумента [a,b]. Шаг изменения аргумента известен и равен dx.

Интересующие нас величины вычисляются согласно следующим формулам:

$$y_{ср.ар.} = \frac{\sum_{i=1}^{n} y_i}{n}$$

$$y_{ср.геом.} = \sqrt[n]{\prod_{i=1}^{n} y_i}$$

где $y_{ср.ар.}$ – среднее арифметическое значение,
$y_{ср.геом.}$ – среднее геометрическое значение,
n – количество значений функции.

Сумму значений функции $s = \sum_{i=1}^{n} y_i$ будем вычислять согласно следующему правилу: вычислим очередное значение функции y_i и прибавим его к сумме предыдущих значений этой функции, т.е. последовательно будем получать все промежуточные суммы.

$s_1 = s_0 + y_1 = y_1;$

$s_2 = s_1 + y_2 = y_1 + y_2;$

$s_3 = s_2 + y_3 = y_1 + y_2 + y_3;$

. . . . . .

$s_n = s_{n-1} + y_n = y_1 + y_2 + y_3 + \ldots + y_n;$

Поскольку запоминать все промежуточные значения сумм по условию задачи не требуется, то для их сохранения достаточно выделить простую переменную. Таким образом, для вычисления суммы будем использовать прием накопления суммы, состоящий в том, что в цикле каждый раз к промежуточной сумме прибавляем очередное

слагаемое, т.е. реализуем рекуррентную формулу: s=s+y, где y- очередное слагаемое. Выполнив описанные вычисления n раз, получим в переменной s требуемое значение суммы.

После первого выполнения цикла переменная s должна хранить значение первого слагаемого, откуда следует, что s_0 должно быть равно 0. Таким образом, перед циклом, в котором вычисляется сумма, переменной s надо присвоить начальное значение, равное нулю.

Произведение $p=\prod_{i=1}^{n} y_i$ может быть найдено с использованием аналогичного приема накопления произведения, состоящего в том, что в цикле каждый раз произведение предшествующих сомножителей p умножается на очередной сомножитель y_i, т.е. используется рекуррентная формула p=p·y, где y - очередной сомножитель. Начальное значение переменной p перед циклом, в котором оно вычисляется, должно быть задано равным единице.

Пример программы, определяющей среднее арифметическое и среднее геометрическое значений функции $y = x^2 e^{\sin x+\cos x}$, вычисляемых на заданном интервале с заданным шагом изменения аргумента, приведен ниже:

```
#include "stdafx.h"
#include "math.h"
#include "conio.h"
int _tmain(int argc, _TCHAR* argv[])
{
      float  x       //текущее значение аргумента функции
            ,xn      //нижняя граница интервала
            ,xk      //верхняя граница интервала
            ,dx      //шаг изменения аргумента
            ,y       //текущее значение функции
            ,s       //сумма значений функции
            ,p       //произведение значений функции
            ,srarif  //среднее арифметическое значений функции
            ,srgeom; //среднее геометрическое значений функции
      int i,n;
      printf("wwedite xn,xk,dx");
      scanf("%f%f%f",&xn,&xk,&dx);
      n=(xk-xn)/dx+1; ;//количество значений аргумента,
                       //при которых будут вычислены значения функции
      s=0;
      p=1;
```

```
	for (i=0;i<n;i++)
	{
		x=xn+i*dx;
		y=x*x*exp(sin(x)+cos(x));
		s+=y;
		p*=y;
	}
	printf("summa=%6.2f  prois=%6.2f\n",s,p);
	srarif=s/n;
	srgeom=pow(p,(float)1.0/n);
	printf("srarifm=%6.2f  srgeom=%6.2f",srarif,srgeom);
	getch();
	return 0;
}
```

Рассмотрим вариант задачи, в которой необходимо вычислить среднее арифметическое и среднее геометрическое элементов одномерного массива. В этом случае все элементы заранее известны, поэтому нет необходимости в цикле вычислять предварительно их значения.

Пример программы:

```
#include "stdafx.h"
#include "math.h"
#include "conio.h"
int _tmain(int argc, _TCHAR* argv[])
{
	const int nn=30;
	float  y[nn]		//массив данных
		,s	//
		,p	//
		,srarif //
		,srgeom;//
	int   i
		,n;
	printf("wwedite kol-wo n");
	scanf("%d",&n);
	printf("\nwwedite massiv\n");
	for (i=0;i<n;i++)
		scanf("%f",&y[i]);
	s=0;
	p=1;
	for (i=0;i<n;i++)
	{
```

```
7
            s+=y[i];
            p*=y[i];
      }
      printf("summa=%6.2f  prois=%6.2f\n",s,p);
      srarif=s/n;
      srgeom=pow(p,(float)1.0/n);
      printf("srarifm=%6.2f  srgeom=%6.2f",srarif,srgeom);
      getch();
      return 0;
}
```

Нахождение наибольшего или наименьшего значения

При поиске наибольшего или наименьшего из множества значений необходимо также рассмотреть два случая, когда это множество значений заранее неизвестно (интересующие значения представляют собой значения функции, вычисляемые в определенных точках) и когда это множество заранее известно (множество значений представляет собой массив элементов).

При поиске наибольшего значения каждое значение y_i сравнивается с максимальным из всех ранее вычисленных значений y_{max}, и большее из этих двух сравниваемых значений принимается за максимум. Данная схема вычислений может быть представлена следующим выражением:

$$y_{\max} = \begin{cases} y_i, & если\ y_i > y_{\max} \\ y_{\max}, & если\ y_i \le y_{\max} \end{cases}$$

В качестве начального значения переменной y_{max} можно присвоить любое из вычисляемых значений функции. Удобно до цикла вычислить и присвоить y_{max} первое значение функции, а в цикле продолжить вычисления со второго значения. Можно поступить иначе, взяв в качестве начального значения переменной y_{max} очень маленькое число, для которого будет выполняться неравенство $y_1 > y_{max}$, например, $y_{max} = -10^{30}$. Строго говоря, в качестве начального значения надо выбирать наименьшее значение для используемого типа данных.

Повторив описанный процесс n раз, получим значение ymax, равное наибольшему значению функции.

Если вычисляемые значения функции не требуется сохранять, то очередное значение функции следует вычислять как значение простой переменной.

Вычисление наименьшего значения функции проводят по следующей схеме:

$$y_{\min} = \begin{cases} y_i, & если\ y_i < y_{\min} \\ y_{\min}, & если\ y_i \geq y_{\min} \end{cases}$$

В качестве начального значения переменной ymin следует выбирать первое значение функции или очень большое число, например, ymin = 10^{30} . Строго говоря, следует выбирать самое большое число для используемого типа данных.

Пример программы определения наибольшего и наименьшего значений функции $y = x \cdot \sin x - \frac{x}{2}\cos\frac{x}{2}$ на заданном интервале.

```
#include "stdafx.h"
#include "math.h"
#include "conio.h"
int _tmain(int argc, _TCHAR* argv[])
{
      float  x    //текущее значение аргумента функции
            ,xn   //нижняя граница интервала
            ,xk   //верхняя граница интервала
            ,dx   //шаг изменения аргумента
            ,y    //текущее значение функции
            ,ymax //максимальное значение функции
            ,ymin;//минимальное значение функции
      int i,n;
      printf("wwedite xn,xk,dx");
      scanf("%f%f%f",&xn,&xk,&dx);
      n=(xk-xn)/dx+1;//количество значений аргумента,
                     //при которых будут вычислены значения функции
      y=xn*sin(xn)-xn/2*cos(xn/2);
      ymax=ymin=y;
      for (i=1;i<n;i++)
      {
            x=xn+i*dx;
            y=x*sin(x)-x/2*cos(x/2);
            if(y>ymax)
                  ymax=y;
            else
                  if(y<ymin)
                        ymin=y;
      }
      printf("ymax=%6.2f  ymin=%6.2f\n",s,p);
```

```
	getch();
	return 0;
}
```

При поиске наибольшего или наименьшего значения среди элементов массива следует принять во внимание, что все элементы заранее известны. Поэтому в качестве начального значения переменной *Xmax*, которая после вычислений должна представлять наибольшее значение, и переменной *Xmin*, которая после вычислений должна представлять наименьшее значение, присваивается значение первого элемента массива. Затем в цикле для очередного элемента массива X_i, i=2, 3,…,n, производится проверка: если $X_i > Xmax$ ($X_i < Xmin$), то переменной *Xmax* (переменной *Xmin*) присваивается значение X_i, иначе значение *Xmax* (*Xmin*) не изменяется.

Поиск максимального и минимального значений в массиве и их индексов представляют следующие фрагменты программ:

```
//Поиск максимального
Xmax= X[0];
imax=0;
for (i=1; i<n; i++)
	if (X[i] >Xmax)
	{
		Xmax=X[i];
		imax=i;
	}
printf("Xmax=%6.2f   imax=%d", Xmax,imax);

//Поиск минимального
Xmin = X[0];
imin=0;
for (i=1; i<n; i++)
	if (X[i] < Xmin)
	{
		Xmin=X[i];
		imin=i;
	}
printf("Xmin=%6.2f    imin=%d", Xmin,imin);
```

Как видно из представленных фрагментов программ, их алгоритмы можно получить один из другого заменой одного знака отношения на другой. Например, заменив в алгоритме поиска максимального знак «больше» (>) на «меньше» (<), получим алгоритм поиска минимального.

Рассмотренный выше приёмы нахождения на заданном интервале максимума/минимума функции и точек (значений аргумента), на которых они достигаются, не следует путать с нахождением экстремумов функции, так как искомый максимум/минимум может находиться на границе интервала, где производная функции не равна нулю.

Наряду с поиском приближенных значений точек экстремумов функции F(X) и её значений в этих точках, могут рассматриваться близкие по смыслу задачи поиска точек перегибов (экстремумов F’(X)), точек экстремальной кривизны (экстремума |F’’(X)|/(1+(F’(X))2)3/2. Напомним, что точка перегиба может находиться только там, где вторая производная равна нулю, бесконечна или не существует. Чтобы такая точка действительно являлась точкой перегиба, первая производная должна изменять свой знак.

Эти задачи можно решать ранее рассмотренными методами, используя аналитические выражения для вычисления значений производных. Однако аналитические выражения производных не всегда известны, поэтому приходится вычислять значения производных по разностным схемам или использовать иные приёмы.

Например, приближенное значение точки экстремума-максимума (локального максимума) будет найдено, если на очередном значении аргумента, изменяемом с заданным шагом, окажется, что слева и справа от него функция имеет меньшие значения. Проверку можно ещё упростить: если заранее известно, что в начале исследуемого интервала функция растет. В этом случае точкой экстремума будет значение аргумента, после которой значение функции начнёт уменьшаться.

4.10. Пример выполнения задания А

Составить программу, которая в массиве *X(N), N*≤20 наибольший отрицательный элемент заменяет суммой отрицательных элементов этого массива, а наименьший положительный элемент заменяет произведением положительных элементов. Если замена невозможна, то выдать соответствующее сообщение. Вывести исходный и преобразованный массивы, найденные наибольшее и наименьшее, их индексы, сумму и произведение.

```
#include "stdafx.h"
#include "conio.h"
int _tmain(int argc, _TCHAR* argv[])
{
      const int   Nmax=20;
      const float MMin=1e30;
      float       X[Nmax],max,min,S,P;
```

```
int         N,iMax,iMin,i;
//ВВОД ИСХОДНЫХ ДАННЫХ
printf( "wwedite kolichestwo elementow:\n ");
scanf("%d",&N);
printf("\n wwedite elementy massiwa");
for (i=0; i<N;i++)
      scanf("%f",&X[i]);
printf("\n Ischodnij massiw:\n");
for (i=0; i<N; i++)
      printf("%6.2f ",X[i]);
//Нахождение максимального, минимального, суммы и произведения
S=0;        //Начальное значение для вычисления  суммы
P=1;        //Начальное значение для произведения
max=-1e30;  //Начальное значение для наибольшего отрицательного
min=MMin;   //Начальное значение для наименьшего положительного
for (i=0; i<N; i++)
      if (X[i]<0)
      {
            S+=X[i];
            if (X[i]>max)
            {
                  max=X[i];
                  iMax=i;
            }
      }
      else if (X[i]>0)
      {
            P*=X[i];
            if (X[i]<min)
            {
                  min=X[i];
                  iMin=i;
            }
      }
//Проверка наличия отрицательных
if (S==0)
      printf("\n otrizatelnix elementow net");
else
{
      X[iMax]=S;
      printf("\n S=%6.2f  max=%6.2f   iMax=%d",S,max,iMax);
}
```

```
	//Проверка наличия положительных
	if (min==MMin)
		printf("\n poloshitelnix elementow net");
	else
	{
		X[iMin]=P;
		printf("\n P=%6.2f  min=%6.2f   iMin=%d",P,min,iMin);
	}
	printf("\n Preobrasowannij massiw:\n");
	for (i=0; i<N; i++)
		printf("%6.2f ",X[i]);
	getch();
	return 0;
}
```

4.11. Задания А для самостоятельной работы

В программе организовать ввод-вывод с поясняющими текстами, а при отладке использовать начальные значения переменных, временно закомментировав ввод исходных данных. Выполнение программы прекратить и вывести соответствующие сообщения, если в массиве не будут найдены требуемые по условию задания значения.

1. Вычислить среднее геометрическое и наименьшее значение среди положительных элементов, сумму и наибольшее значение среди отрицательных элементов в массиве D(n), n≤25. Вывести массив, среднее геометрическое, сумму, наименьшее и наибольшее значения.

2. Найти наибольшее и наименьшее значения, их индексы и среднее арифметическое отрицательных элементов, расположенных между ними, в массиве D(n), n≤25. Вывести массив, среднее арифметическое, наименьшее и наибольшее значения и их индексы.

3. Найти наибольшее и наименьшее значения, их индексы и среднее геометрическое положительных, расположенных между ними, в массиве D(n), n≤25.. Вывести массив, наименьшее и наибольшее значения, их индексы, среднее геометрическое.

4, Вычислить среднее арифметическое значение элементов, удовлетворяющих условию a≤ Di ≤b, найти наибольшее значение среди отрицательных элементов и его индекс в массиве D(n), n≤25. Вывести массив, среднее арифметическое, наибольшее значение и его индекс.

5. Вычислить среднее арифметическое элементов массива D(n), n≤25, а также найти элемент, отличающийся от среднего на наибольшую величину. Вывести массив, среднее, и найденный элемент.

6. Вычислить среднее арифметическое элементов массива D(n), n≤25, без учета максимального и минимального элементов. Вывести массив, среднее, наибольший и наименьший элементы.

7. Из массива D(n), n≤25 переписать элементы, расположенные между средним геометрическим и средним арифметическим модулей элементов, подряд в массив F. Вывести массивы, среднее арифметическое и среднее геометрическое.

8. Вычислить среднее геометрическое положительных элементов, кратных заданному числу, и сумму отрицательных элементов, кратных другому заданному числу, в массиве D(n), n≤25. Вывести массив, среднее геометрическое и сумму.

9. Найти наибольший положительный элемент среди элементов с четными индексами, и сумму элементов с нечетными индексами, больших найденного наибольшего, в массиве D(n), n≤25. Вывести массив, наибольший элемент и сумму.

10. Вычислить количество четных и нечетных элементов массива D(n), n≤25. Если число четных элементов больше, то вычислить среднее арифметическое всех элементов массива, в противном случае - среднее геометрическое положительных. Вывести массив, количество четных и нечетных чисел, среднее арифметическое или среднее геометрическое.

11. Найти наибольшее и наименьшее значения и их индексы в массиве D(n), n≤25. Наименьший элемент заменить суммой предшествующих элементов, а наибольший - произведением последующих. Вывести массив, наименьшее, наибольшее значения, их индексы, сумму и произведение.

12. Найти наибольшее и наименьшее значения и их индексы в массиве D(n), n≤25. Вычислить их среднее значение и переписать в массив B без пропусков элементы, расположенные в δ окрестности найденного среднего. Вывести массивы D и B, наименьшее, наибольшее значения, их индексы, среднее.

13. Определить в массиве D(n), n≤25 элемент, который при умножении на последующий дает максимальное произведение. Для последнего элемента последующим считать первый. Вывести массив, найденный элемент и наибольшее произведение.

14. В массиве X(n), n≤25 вычислить среднее арифметическое элементов, расположенных в интервале [A,B], и среднее арифметическое элементов, лежащих вне этого интервала. Вычислить произведение элементов, расположенных между найденными средними. Вывести массив, произведение, средние значения.

Оглавление

15. В массиве X(n), n≤25 найти первый и последний положительные элементы. Найти сумму элементов, расположенных между найденными элементами и произведение элементов, лежащих вне этого интервала. Вывести массив, найденные элементы, сумму и произведение.

16. Вычислить отношение C=A/B, где A – сумма элементов, больших полусуммы наименьшего и наибольшего элементов массива D(n), n≤25, а B – сумма элементов по абсолютному значению меньших найденной полусуммы. Вывести массив, наибольший, наименьший элементы, суммы и их отношение.

17. Определить в массиве D(n), n≤25 элемент, который при сложении с последующим дает максимальную сумму. Для последнего элемента последующим считать первый. Вывести массив, найденный элемент и наибольшую сумму.

18. В массиве X(n), n≤25, все элементы которого расположены по неубыванию, найти среднее арифметическое всех элементов. Найти в массиве элемент, замена которого средним арифметическим не нарушит упорядоченность, и провести замену. Вывести массив, найденное среднее.

19. В массиве D(n), n≤25. вычислить полусумму наибольшего и наименьшего элементов. Переписать в массив C сначала элементы, не большие полусуммы, а затем – большие. Вычислить отношение R=A/B, где A- сумма по абсолютной величине элементов первой части массива C , а B – сумма элементов второй части массива C.

20. В массиве D(n), n≤25 найти наибольшее четное и наибольшее нечетное числа. Переписать в массив C все числа, расположенные вне интервала между найденными наибольшими. Вычислить среднее арифметическое элементов массива C. Вывести массив, наибольшие, среднее арифметическое.

21. Найти наибольшее и наименьшее значения и их индексы в массиве D(n), n≤25. Если индекс наименьшего значения меньше индекса наибольшего, то вычислить сумму элементов, стоящих после наименьшего, в противном случае произведение элементов, стоящих после наибольшего. Вывести массив, наименьшее и наибольшее значения и их индексы, сумму или произведение.

22. Вычислить среднее геометрическое элементов до первого отрицательного и среднее арифметическое до первого положительного элемента в массиве D(n), n≤25. Вывести массив, среднее геометрическое и среднее арифметическое.

23. Вычислить среднее арифметическое элементов массива X(n), n≤25. Подсчитать количество элементов, лежащих в δ окрестности найденного среднего, найти среди этих элементов наибольший и наименьший. Вывести массив, среднее, количество, наибольший и наименьший элементы.

Оглавление

24. Найти наименьший положительный и наибольший отрицательный элементы массива D(n), n≤25. Переписать в массив C элементы, расположенные в интервале между найденными элементами и вычислить их среднее арифметическое. Вывести массив, наименьшее и наибольшее значения, среднее.

25. Подсчитать в массиве X(n), n≤25 количество отрицательных и положительных элементов. Если отрицательных больше, то вычислить их среднее арифметическое, иначе – среднее геометрическое положительных. Вывести массив, количество положительных и отрицательных, среднее арифметическое или среднее геометрическое.

26. Найти наибольшее и наименьшее значения и их индексы в массиве D(n), n≤25. Наименьший элемент заменить суммой отрицательных элементов, а наибольший - произведением положительных. Вывести массив, наименьшее, наибольшее значения, их индексы, сумму и произведение.

4.12. Пример выполнения задания Б

Известно, что функция y=x·sin(3·x)·cos(x) на интервале значений аргумента [xn,xk] имеет несколько локальных экстремумов. Требуется найти все точки локальных экстремумов и наименьшее расстояние между двумя соседними экстремумами, изменяя значение аргумента с заданным шагом. Вывести найденные значения точек локального экстремума, значения функции в этих точках, указав вид экстремума, а также две самые близко расположенные точки экстремумов.

```
#include "stdafx.h"
#include "math.h"
#include "conio.h"

int _tmain(int argc, _TCHAR* argv[])
{
      double x    //текущее значение аргумента функции
            ,xn   //начальное значение аргумента
            ,xk   //конечное значение аргумента
            ,dx   //приращение аргумента
            ,f1   //значение функции в первой из трех соседних точек
            ,f2   //значение функции во второй из трех смежных точек
            ,f3   //значение функции в третьей из трех смежных точек
            ,min  //минимальное рапсстояние между экстремальными точками
```

```
        ,x1    //первое из двух соседних экстремальных значений
        ,x2    //второе из двух соседних экстремальных значений
        ,xe1   //первая точка искомого экстремума
        ,xe2   //вторая точка искомого экстремума
        ,dxe;  //расстояние между экстремальными точками
int     i,     //параметр цикла
        n,     //количество точек для вычисления функции
        k;     //количество экстремумов
printf("wwedite xn,xk,dx  ");
scanf("%lf%lf%lf",&xn,&xk,&dx); //ввод исходных данных
n=(xk-xn)/dx+1;
x=xn;
f2=x*sin(3*x)*cos(x);
printf("\n  x=%8.4f   f=%8.4f  ",x,f2);
x+=dx;
f3=x*sin(3*x)*cos(x);
printf("\n  x=%8.4f   f=%8.4f  ",x,f3);
k=0;
min=xk-xn;  //начальное значение минимального расстояния между
            //экстремумами
for (i=2;i<n;i++)//цикл вычисления значений функции
{
      f1=f2;      // переприсваивание значений функции
      f2=f3;
      x=xn+i*dx;  // очередное значение аргумента
      f3=x*sin(3*x)*cos(x);
      printf("\n  x=%8.4f   f=%8.4f  ",x,f3);
      if (f2>f1&&f2>f3) //проверка на локальный максимум
      {
            k++;
            printf("\n k=%4d  x=%8.4f   f=%8.4f  max",k,x-dx,f2);
            if (k==1)
                  x1=x-dx; //запоминание первого экстремума (максимума)
            if (k>1)
            {
                  x2=x-dx; //запоминание второго соседнего экстремума
                           //(максимума)
                  dxe=x2-x1;  //расстояние между двумя текущими
                              //между экстремумами
                  if(dxe<min) //определение минимального расстояние
```

```
                                    //между экстремумами
                        {
                              min=dxe;xe1=x1;xe2=x2;
                        }
                        x1=x2;
                  }
            }
            if (f2<f1&&f2<f3) // проверка на локальный минимум
            {
                  k++;
                  printf("\n k=%4d  x=%8.4f   f=%8.4f  min",k,x-dx,f2);
                  if (k==1) )// запоминание первого экстремума (минимума)
                        x1=x-dx;
                  if (k>1)
                  {
                        x2=x-dx; //запоминание второго соседнего экстремума
                                 //(минимума)
                        dxe=x2-x1; //расстояние между двумя текущими
                                   //экстремальными точками
                        if(dxe<min) //определение минимального расстояние
                                    //между экстремумами
                        {
                              min=dxe;xe1=x1;xe2=x2;
                        }
                        x1=x2;
                  }
            }
      }// конец цикла for (i=2;i<n;i++)

      printf("\n min=%8.4f  xe1=%8.4f  xe2=%8.4f",min,xe1,xe2);
      getch();
      return 0;
}
```

4.13. Задания Б для самостоятельной работы

Во всех заданиях не использовать аналитических формул производных заданных функций. Ввод исходных данных выполнить с применением переменных $X_{нач}$, $X_{кон}$, h_x. Найденные значения выводить с поясняющими текстами.

1. Составить программу вычисления максимального и минимального значений функции $Y=3X^3-15X^2-12X+8$ и соответствующих значений аргумента при его изменении на интервале от –5 до 15 с шагом 0,01.

2. Составить программу нахождения локальных минимумов функции $X^2\sin^3(3X)$ при изменении аргумента от -3 до 2,5 с шагом 0,001. Среди найденных минимумов найти точку с наибольшим значением функции.

3. Составить программу вычисления локальных максимумов функции $X^3\sin^5(4X)$ при изменении аргумента на интервале от –1,6 до 3,2 с шагом 0,001. Среди найденных максимумов найти точку с наименьшим значением функции.

4. Составить программу вычисления минимального расстояния между точками экстремумов-минимумов функции $(1+6\sin^2 X)^{1/2}\cos(3X)$ и соответствующих значений функции при изменении X на интервале -5 до 12 с шагом 0.001.

5. Произвольные значения аргумента функции $Y=2X^4-12X^3-4X^2+2$ находятся в массиве X(n), n≤20. Составить программу вычисления максимального и минимального значений функции, а также соответствующих значений элементов массива X и их индексов.

6. Известно, что в интервале -7 до 7 уравнение cos(2,5X)sin(3X) -0,2=0 имеет несколько корней. Составить программу нахождения корней уравнения, а также корня, в котором производная функции имеет максимальное значение.

7. Известно, что в интервале –10 до 20 функция $Y = 4^{-(x-1)^2}\cos 2X$ имеет несколько точек перегиба. Составить программу нахождения точек перегиба, а также той из них, где первая производная функции имеет максимальное значение.

8. Составить программу вычисления минимального расстояния между точками локальных максимумов функции (X+1)sin(1,5X)+Xcos(1,5X)) при изменении X на интервале -3,2 до 8 с шагом 0,001.

9. Известно, что в интервале -7 до 7 уравнение sin(2,5X)cos(3X) +0,2=0 имеет несколько корней. Составить программу нахождения корней уравнения, а также корня, в котором производная функции имеет минимальное значение.

10. Найти на кривой $Y(X)=\sin^3(3X)\cos(2X)$ все точки, принадлежащие интервалу [a,b] и расположенные на минимальном расстоянии от прямой Y=2.

11.В массивах X(N), Y(N), N≤20, заданы координаты точек на плоскости. Найти такие точки, для которых расстояние $d=|(aX_i+bY_i+c)/(a^2+b^2)^{1/2}|$ от точки (Xi,Yi) до прямой aX+bY+c=0 находится в интервале [r1,r2]. Среди найденных точек найти наименее и наиболее удаленные от плоскости.

12. Найти на кривой $Y(X)=\sin^2(2X)\cos(4X)$ точку, абсцисса которой принадлежит интервалу [r1,r2] и сумма расстояний от которой до прямых $a_1x+b_1y+c_1=0$ и $a_2x+b_2y+c_2=0$ минимальна.

13. Составить программу вычисления значения аргумента, изменяя его на интервале от -4 до 10 с шагом 0,001, при котором производная функции $Y=X^2\sin^2X\cos(3X)$ имеет минимальное и максимальное значение.

14.В массиве X(N), $N\leq30$, заданы абсциссы точек, принадлежащих кривой $Y=X^2\sin(2X)\cos(3X)$. Среди этих точек найти точки наиболее и наименее удаленные от прямой ax+by+c=0.

15.Составить программу вычисления максимального расстояния между экстремумами-минимумами функции Y=x(sin(3X)+cos(2x)) и соответствующих значений функции при изменении X на интервале -3,5 до 4,5 с шагом 0.001.

16. Составить программу нахождения локальных максимумов функции $X^2(\sin^3(3X)-\cos^2(2x))$ при изменении аргумента от -3,5 до 5,5 с шагом 0,001. Среди найденных максимумов найти точку с наименьшим значением производной.

17.Составить программу вычисления максимального расстояния между соседними корнями уравнения 2cos(3X)-3sin(2X)+0,1=0 , изменяя X на интервале -4 до 3 с шагом 0,001.

18. Известно, что в интервале -7 до 7 уравнение sin(2,5X)sin(X)cos(x/2) -0,1=0 имеет несколько корней. Составить программу нахождения корней уравнения, а также корня, в котором производная функции имеет минимальное значение.

19. Среди точек кривой $Y=\cos(3X)\sin^2(X)$ с абсциссами, принадлежащими интервалу [r1,r2], и расположенных в нижней полуплоскости, найти наименее и наиболее удаленные от прямой ax+by+c=0.

20. Составить программу вычисления минимального положительного и максимального отрицательного значений функции $Y=2X^3-X^2-3X$ и соответствующих значений аргумента при его изменении на интервале [r1,r2] с шагом dx.

21. Среди корней уравнения cos(3,5X)sin(2X)cos(x/2) -0,1=0, принадлежащих интервалу [r1,r2], найти ближайший к точке с координатами (Xt,Yt) и наиболее удаленный от этой же точки.

22. Составить программу вычисления минимального расстояния между соседними корнями уравнения 2sin(4X)-3sin(3X)+0,1=0 , расположенными на интервале -4 до 3.

23. Составить программу нахождения минимального и максимального значений первой производной функции $Y=X^4\sin X-12X^3\cos X-4X^2\sin(X/2)\cos(X/2)+2$ при значениях аргумента, принадлежащих интервалу [r1,r2].

Оглавление

24. На кривой Y=(|X|+4)sin(3X)cos(X/2) найти среди точек с абсциссами, принадлежащими интервалу [r1,r2], наиболее и наименее удаленные от точки с координатами (Xt,Yt).

25. Среди локальных экстремумов функции $X^2\sin^3(3X)$ с положительными значениями функции в точке экстремума найти расположенные на наибольшем расстоянии друг от друга.

26. Среди локальных экстремумов функции $X^2\sin^3(3X)$ с отрицательными значениями функции в точке экстремума найти расположенные на наименьшем расстоянии друг от друга.

4.14. Вычисление суммы бесконечного ряда с заданной точностью

Пусть задана последовательность чисел R_1, R_2, R_3,…,R_n,…. Выражение $R_1 + R_2 + R_3 +…+ R_n +…$ называют *бесконечным рядом*, или просто *рядом*, а числа R_1, R_2, R_3,… - *членами ряда*. При этом имеют в виду, что накопление суммы ряда начинается с первых его членов. Сумма $S_n = \sum_{i=1}^{n} R_i$ называется *частичной суммой ряда*: при n=1 – первой частичной суммой, при n=2 – второй частичной суммой и так далее.

Называется ряд сходящимся, если последовательность его частичных сумм имеет предел, и расходящимся – в противном случае. Понятие суммы ряда можно расширить [5], и тогда некоторые расходящиеся ряды также будут обладать суммами. Именно расширенное понимание суммы ряда будет использовано при разработке алгоритмов при следующей постановке задачи: накопление суммы следует выполнять до тех пор, пока очередной член ряда по абсолютной величине больше заданной величины ε.

В общем случае все или часть членов ряда могут быть заданы выражениями, зависящими от номера члена ряда и переменных. Например,

$$S = 1 - X + \frac{X^2}{2!} - \frac{X^3}{3!} - \frac{X^4}{4!} + \frac{X^5}{5!} + ... = 1 + \sum_{i=1}^{\infty} (-1)^i \frac{X^i}{i!}$$

Тогда возникает вопрос, как минимизировать объём вычислений - вычислять значение очередного члена ряда по общей формуле члена ряда (в приведённом примере её представляет выражение под знаком суммы), по рекуррентной формуле (её вывод представлен ниже) или использовать рекуррентные формулы лишь для частей выражения члена ряда (см. ниже).

4.15. Вывод рекуррентной формулы для вычисления члена ряда

Пусть требуется найти ряд чисел R_1, R_2, R_3,…, последовательно вычисляя их по формулам

$$R_1 = \frac{1}{2} X\ ,\ R_2 = \frac{1 \cdot 3}{2 \cdot 4} X^2,\ \ldots,\ R_N = \frac{1 \cdot 3 \cdot 5 \cdot 7 \cdot \ldots \cdot (2N-3)(2N-1)}{2 \cdot 4 \cdot 6 \cdot 8 \cdot 9 \cdot \ldots \cdot (2N-2)(2N)} X^N \qquad (4.1)$$

Для сокращения вычислений в данном случае удобно воспользоваться *рекуррентной формулой* вида $R_N = \alpha(X, N) \cdot R_{N-1}$, позволяющей вычислить значение R_N при N>1, зная значение предыдущего члена ряда R_{N-1}, где $\alpha(X, N)$ - выражение, которое можно получить после упрощения отношения выражения в формуле (4.1) для N к выражению для

N-1:

$$\alpha(X,N)=\frac{\frac{1\cdot3\cdot5\cdot7\cdot\ldots\cdot(2\cdot N-3)\cdot(2\cdot N-1)}{2\cdot4\cdot6\cdot8\cdot9\cdot\ldots\cdot(2\cdot N-2)\cdot(2\cdot N)}\cdot X^{N}}{\frac{1\cdot3\cdot5\cdot7\cdot\ldots\cdot(2\cdot(N-1)-3)\cdot(2\cdot(N-1)-1)}{2\cdot4\cdot6\cdot8\cdot9\cdot\ldots\cdot(2\cdot(N-1)-2)\cdot(2\cdot(N-1))}\cdot X^{N-1}}=$$

$$=\frac{1\cdot3\cdot5\cdot7\cdot\ldots\cdot(2\cdot N-3)\cdot(2\cdot N-1)\cdot X^{N}\cdot2\cdot4\cdot6\cdot8\cdot9\cdot\ldots\cdot(2\cdot(N-1)-2)\cdot(2\cdot(N-1))}{1\cdot3\cdot5\cdot7\cdot\ldots\cdot(2\cdot(N-1)-3)\cdot(2\cdot(N-1)-1)\cdot X^{N-1}2\cdot4\cdot6\cdot8\cdot9\cdot\ldots\cdot(2\cdot N-2)\cdot(2\cdot N)}=$$

$$=\frac{\cancel{1\cdot3\cdot5\cdot7\cdot\ldots\cdot(2\cdot N-3)}\ (2\cdot N-1)\cdot\cancel{X^{N-1}}\cdot X\cdot\cancel{2\cdot4\cdot6\cdot8\cdot9\cdot\ldots\cdot(2\cdot N-4)\cdot(2\cdot N-2)}}{\cancel{1\cdot3\cdot5\cdot7\cdot\ldots\cdot(2\cdot N-5)\cdot(2\cdot N-3)}\cdot\cancel{X^{N-1}}\cdot\cancel{2\cdot4\cdot6\cdot8\cdot9\cdot\ldots\cdot(2\cdot N-2)}\ (2\cdot N)}=$$

$$=\frac{(2\cdot N-1)\cdot X}{2\cdot N}$$

Таким образом, рекуррентная формула примет вид $R_N=\frac{(2N-1)X}{2N}R_{N-1}$ (4.2).

Из сравнения общей формулы члена ряда (4.1) и рекуррентной (3.2) видно, что рекуррентная формула значительно упрощает вычисления. Применим ее для N=2, 3 и 4 зная, что $R_1=\frac{1}{2}X$:

$$R_2=\frac{(2\cdot2-1)X}{2\cdot2}\cdot\frac{1}{2}X=\frac{1\cdot3}{2\cdot4}\cdot X^2$$

$$R_3=\frac{(2\cdot3-1)X}{2\cdot3}\cdot\frac{1\cdot3}{2\cdot4}\cdot X^2=\frac{1\cdot3\cdot5}{2\cdot4\cdot6}\cdot X^3$$

$$R_4=\frac{(2\cdot4-1)X}{2\cdot4}\cdot\frac{1\cdot3\cdot5}{2\cdot4\cdot6}\cdot X^3=\frac{1\cdot3\cdot5\cdot7}{2\cdot4\cdot6\cdot8}\cdot X^4$$

Способы вычисления значения члена ряда

Для вычисления значения члена ряда, в зависимости от его вида, может оказаться предпочтительнее использование либо общей формулы члена ряда, либо рекуррентной формулы, либо *смешанного способа вычисления значения члена ряда*, когда для одной или нескольких частей члена ряда используются рекуррентные формулы, и затем их значения подставляются в общую формулу члена ряда. Например,

- для ряда $1+\frac{1}{2\cdot3}+\frac{1}{3\cdot5}+\frac{1}{4\cdot7}+\ldots+\frac{1}{N\cdot(2N-1)}+\ldots$ проще вычислять значение члена ряда

R_N по его общей формуле $R_N = \frac{1}{N \cdot (2N-1)}$ (сравните с $R_N = \frac{(N-1)\cdot(2N-3)}{N\cdot(2N-1)} R_{N-1}$ по рекуррентной формуле);

- для ряда $1 + \frac{2}{1\cdot 2} + \frac{2^3}{1\cdot 2\cdot 3} + \ldots + \frac{2^N}{N!} + \ldots$ лучше воспользоваться рекуррентной формулой $R_N = \frac{2}{N} R_{N-1}$;

- для ряда $\frac{X^3 \ln(X^3)}{1\cdot 2\cdot 3} + \frac{X^6 \ln(X^6)}{1\cdot 2\cdot 3\cdot 4\cdot 5\cdot 6} + \ldots + \frac{X^{3N} \ln(X^{3N})}{N!} + \ldots$следует применить смешанный способ, вычисляя $A_N = X^{3N}$ по рекуррентной формуле $A_N = X^3 \cdot A_{N-1}$, N=2, 3,… при A_1=1 и B_N=N! - также по рекуррентной формуле $B_N = N \cdot B_{N-1}$, N=2, 3,… при B_1=1, а затем – член ряда R_N - по общей формуле, которая примет вид $R_N = \frac{A_N \cdot \ln(A_N)}{B_N}$.

4.16. Примеры выполнения задания

1. Вычислить с точностью ε для $0^o \le X \le 45^o$

– приближенное значения функции cos X по формуле:

$$S = 1 - \frac{X^2}{2!} + \frac{X^4}{4!} - \cdots + (-1)^N \frac{X^{2N}}{(2N)!} + \cdots,$$

используя рекуррентную формулу для вычисления члена ряда:

$$R_N = \frac{-R_{N-1} \cdot X^2}{(2N-1)\cdot(2N)};$$

– точное значение функции cos X;

– абсолютную и относительную ошибки приближенного значения.

```
// Пример1.cpp : Defines the entry point for the console application.

#include "stdafx.h"
#include "math.h"
#include "conio.h"
#define dbg //ПОСЛЕ ОТЛАДКИ ПРЕВРАТИТЬ ЭТУ СТРОКУ В КОММЕНТАРИЙ
int _tmain(int argc, _TCHAR* argv[])
{
  //Объявленные ниже переменные будут представлять:
  const double k=3.14/180;//коэффикиент перевода
                          //из градусов в радианы.
```

```
  double
        //Начальные значения при отладке:
      X=15,    // - аргумент функции,
      Eps=1e-8,// - требуемую точность.
        //Начальные значения для вычисления суммы ряда:
      R=1,  // - первого члена ряда,
      S=0;  // - суммы ряда,
  int
      N=0;  // - номера члена ряда.
        //вычисляемые в программе значения:
  double
      D,  // - часть X^2 рекуррентной
          //   формулы вычисления члена ряда,
      Y;  // - значение функции ln(1+X).

#ifndef dbg
  //Операторы, НЕ используемые при отладке.
  printf("input Eps: ");
  scanf("%lf",&Eps);
  printf("input X: ");
  scanf("%lf",&X);
#endif
  D=k*X; //Перевод X в радианы и
  D=D*D; //возведение в квадрат: D=(k*X)^2.

  //Цикл для вычисления членов ряда и накопления их суммы.
  //Выполнять, пока модуль очередного члена ряда
  //больше требуемой точности.
  while (fabs(R)>Eps)
  {
    S=S+R;//Включение очередного члена ряда в сумму.
#ifdef dbg
  //Операторы, используемые при отладке
    if (N<5)  //Вывод, используемый при отладке
            printf("N=%d\t\tR=%lf\t\tS=%lf\n",N,R,S);
#endif
    N=N+1;   //Увеличение номера члена ряда.
    R=-R*D/2/N/(2*N-1);  //Вычисление N-го члена ряда.
  }
  //Вывод результатов вычислений:
  //число шагов, за которое достигнута заданная точность,
  printf("N = %d\n",N);
```

```
  //приближенное значение функции cos(k*X),
  printf("S = %lf\n",S);
  //точное значение функции cos(k*X),
  printf("cos(X) = %lf\n",cos(k*X));
  //абсолютная ошибка,
  printf("|cos(X)-S| = %e\n"
         ,fabs(cos(k*X)-S));
  //относительная ошибка.
  printf("|(cos(X)-S)/cos(X)| = %e\n"
          ,fabs((cos(k*X)-S)/cos(k*X)));
      getch();
      return 0;
}
```

2. Вычислить с точностью ε для $0 \le X \le 1$:

– приближенное значения функции ln(1+ X) по формуле

$$S = X - \frac{X^2}{2} + \frac{X^3}{3} - \frac{X^4}{4} + \cdots + (-1)^{N+1}\frac{X^N}{N} + \cdots$$

используя смешанный способ вычисления члена ряда;

– точное значение функции ln(1+X);

– абсолютную и относительную ошибки приближенного значения.

Предусмотреть обработку исключения, возникающего при вычислениях вследствие ошибок в исходных данных, обеспечивающую вывод типа исключения и числа шагов вычисления суммы ряда, на котором оно возникло.

```
//Пример2.cpp : Defines the entry point for the console application.

#include "stdafx.h"
#include "math.h"
#include "conio.h"
#define dbg //ПОСЛЕ ОТЛАДКИ ПРЕВРАТИТЬ ЭТУ СТРОКУ В КОММЕНТАРИЙ
int _tmain(int argc, _TCHAR* argv[])
{
  //Объявленные ниже переменные будут представлять:
  double
        //Начальные значения при отладке:
      X=0.95,     // - аргумент функции,
      Eps=1e-18,  // - требуемую точность.
        //Начальные значения для вычисления суммы ряда:
      D=1,   // - начальное значение для вычисления числителя
```

```
                //   члена ряда по рекуррентной формуле,
      S=0;    // - суммы ряда,
  int
      N=1;  // - номера члена ряда.
        //Вычисляемые в программе значения,
  double
      R,  // - значение члена ряда,
      Y;  // - значение функции ln(1+X).

#ifndef dbg
  //Операторы, НЕ используемые при отладке
  printf("input Eps: ");
  scanf("%lf",&Eps);
  printf("input X: ");
  scanf("%lf",&X);
#endif
  R=X;//Первый член ряда.
  S=0;//Начальное значение для накопления суммы
       //членов ряда.
  //Цикл для вычисления членов ряда
  //и накопления их суммы.
  //Выполнять, пока модуль очередного члена ряда
  //больше требуемой точности.
  while (fabs(R)>Eps)
  {
    S=S+R;//Включение очередного члена ряда в сумму
#ifdef dbg
  //Операторы, используемые при отладке
    if (N<=5)  //Вывод, используемый при отладке
            printf("N=%d\t\tR=%lf\t\tS=%lf\n",N,R,S);
#endif
    N=N+1;    //Увеличение номера члена ряда.
    D=-D*X;   //Рекуррентная формула вычисления
              //числителя члена ряда.
    R=D/N;    //Вычисление N-го члена ряда.
  }
  //Вывод результатов вычислений:
  //число шагов, за которое достигнута заданная точность,
  printf("      N = %d\n",N);
  //приближенное значение функции ln(1+X),
  printf("      S = %lf\n",S);
  //точное значение функции ln(1+X),
```

```
  printf("ln(1+X) = %lf\n",log(1+X));
  //абсолютная ошибка,
  printf("|ln(1+X)| = %e\n"
         ,fabs(log(1+X)-S));
  //относительная ошибка.
  printf("|(ln(1+X)-S)/ln(1+X)| = %e\n"
          ,fabs((log(1+X)-S)/log(1+X)));
      getch();
      return 0;
}
```

4.17. Задания для самостоятельной работы

Составить программу вычисления суммы ряда с заданной точностью ε. Анализируя код программы, выявить возможные причины возникновения исключений и ввести их обработку, обеспечивающую вывод типа исключения и пояснение к причине его возникновения.

1. Вычислить с точностью ε

* приближенное значения функции ln(1+X)/X по формуле

$$S = 1 - \frac{X}{2} + \frac{X^2}{3} - \frac{X^3}{4} + \cdots + (-1)^{N+1}\frac{X^{N-1}}{N} + \cdots \quad ,$$

используя смешанный способ вычисления члена ряда,

* точное значение функции ln(1+X)/X,
* абсолютную и относительную ошибки приближенного значения.

2. Вычислить с точностью ε

* приближенное значения функции e^X по формуле

$$S = 1 + X + \frac{X^2}{2!} + \frac{X^3}{3!} + \frac{X^4}{4!} + \frac{X^5}{5!} + \ldots \quad ,$$

используя рекуррентную формулу для вычисления члена ряда,

* точное значение функции e^X,
* абсолютную и относительную ошибки приближенного значения.

3. Вычислить с точностью ε

* приближенное значения функции sinX по формуле

$$S = X - \frac{X^3}{3!} + \frac{X^5}{5!} - \frac{X^7}{7!} + \frac{X^9}{9!} + \ldots \quad ,$$

используя рекуррентную формулу для вычисления члена ряда,

* точное значение функции sinX,
* абсолютную и относительную ошибки приближенного значения.

4. Вычислить с точностью ε

* приближенное значения функции $\sqrt{1+X}$ по формуле

$$S = 1 + \frac{X}{2} - \frac{X^2}{2 \cdot 4} + \frac{3X^3}{2 \cdot 4 \cdot 6} - \frac{3 \cdot 5X^4}{2 \cdot 4 \cdot 6 \cdot 8} + \ldots,$$

используя рекуррентную формулу для вычисления члена ряда,

* точное значение функции $\sqrt{1+X}$,
* абсолютную и относительную ошибки приближенного значения.

5. Вычислить с точностью ε

* приближенное значения функции arcsinX по формуле

$$S = X + \frac{X^3}{2 \cdot 3} + \frac{3X^5}{2 \cdot 4 \cdot 5} + \frac{3 \cdot 5X^7}{2 \cdot 4 \cdot 6 \cdot 7} + \frac{3 \cdot 5 \cdot 7X^9}{2 \cdot 4 \cdot 6 \cdot 8 \cdot 9} + \ldots \quad ,$$

используя рекуррентную формулу для вычисления члена ряда,

* точное значение функции arcsinX,
* абсолютную и относительную ошибки приближенного значения.

6. Вычислить с точностью ε

* приближенное значения функции arctgX по формуле

$$S = X - \frac{X^3}{3} + \frac{X^5}{5} - \frac{X^7}{7} + \frac{X^9}{9} + \ldots \qquad ,$$

используя смешанный способ вычисления члена ряда,

* точное значение функции arctgX,
* абсолютную и относительную ошибки приближенного значения.

7. Вычислить с точностью ε

* приближенное значения функции $\frac{e^X - e^{-X}}{2}$ по формуле

$$S = X + \frac{X^3}{3!} + \frac{X^5}{5!} + \frac{X^7}{7!} + \frac{X^9}{9!} + \ldots ,$$

используя рекуррентную формулу для вычисления члена ряда,

* точное значение функции $\frac{e^X - e^{-X}}{2}$,
* абсолютную и относительную ошибки приближенного значения.

8. Вычислить с точностью ε

* приближенное значения функции $\frac{e^X + e^{-X}}{2}$ по формуле

$$S = 1 + \frac{X^2}{2!} + \frac{X^4}{4!} + \frac{X^6}{6!} + \frac{X^8}{8!} + \dots ,$$

используя рекуррентную формулу для вычисления члена ряда,

* точное значение функции $\frac{e^X + e^{-X}}{2}$,

* абсолютную и относительную ошибки приближенного значения.

9. Вычислить с точностью ε

* приближенное значения функции ln(1-X) по формуле

$$S = -X - \frac{X^2}{2} - \frac{X^3}{3} - \frac{X^4}{4} - \frac{X^5}{5} + \dots ,$$

используя смешанный способ вычисления члена ряда,

* точное значение функции ln(1-X),

* абсолютную и относительную ошибки приближенного значения.

10. Вычислить с точностью ε

* приближенное значения функции $\ln\frac{1+X}{1-X}$ по формуле

$$S = 2 \cdot (X + \frac{X^3}{3} + \frac{X^5}{5} + \frac{X^7}{7} + \frac{X^9}{9} + \dots),$$

используя смешанный способ вычисления члена ряда,

* точное значение функции $\ln\frac{1+X}{1-X}$,

* абсолютную и относительную ошибки приближенного значения.

11. Вычислить с точностью ε

* приближенное значения функции $(1+X)^{-3}$ по формуле

$$S = 1 - \frac{2 \cdot 3}{2} X + \frac{3 \cdot 4 \cdot X^2}{2} - \frac{4 \cdot 5 \cdot X^3}{2} + \frac{5 \cdot 6 \cdot X^4}{2} - \frac{6 \cdot 7 \cdot X^5}{2} + \dots ,$$

используя смешанный способ вычисления члена ряда,

* точное значение функции $(1+X)^{-3}$,

* абсолютную и относительную ошибки приближенного значения.

12. Вычислить с точностью ε

* приближенное значения функции $\ln(X+\sqrt{1+X^2})$ по формуле

$$S = X - \frac{1}{2}\cdot\frac{X^3}{3} + \frac{1}{2}\cdot\frac{3}{4}\cdot\frac{X^5}{5} - \frac{1}{2}\cdot\frac{3}{4}\cdot\frac{5}{6}\cdot\frac{X^7}{7} + \frac{1}{2}\cdot\frac{3}{4}\cdot\frac{5}{6}\cdot\frac{7}{8}\cdot\frac{X^9}{9} - \ldots ,$$

используя смешанный способ вычисления члена ряда,

* точное значение функции $\ln(X+\sqrt{1+X^2})$,
* абсолютную и относительную ошибки приближенного значения.

13. Вычислить с точностью ε

* приближенное значения функции e^{-X^2} по формуле

$$S = 1 - \frac{X^2}{1!} + \frac{X^4}{2!} - \frac{X^6}{3!} + \frac{X^8}{4!} - \ldots + (-1)^N \frac{X^{2N}}{N!} + \ldots,$$

используя рекуррентную формулу для вычисления члена ряда,

* точное значение функции e^{-X^2},
* абсолютную и относительную ошибки приближенного значения.

14. Вычислить с точностью ε

* приближенное значения функции $(1+X)^{-2}$ по формуле

$$S = 1 - 2X + 3X^2 - 4X^3 + 5X^4 - \ldots,$$

используя смешанный способ вычисления члена ряда,

* точное значение функции $(1+X)^{-2}$,
* абсолютную и относительную ошибки приближенного значения.

15. Вычислить с точностью ε

* точное значение функции $\frac{1}{\sqrt{1+X}}$,
* приближенное значения функции $\frac{1}{\sqrt{1+X}}$ по формуле

$$S = 1 - \frac{1}{2}X + \frac{1\cdot 3}{2\cdot 4}X^2 - \frac{1\cdot 3\cdot 5}{2\cdot 4\cdot 6}X^3 + \frac{1\cdot 3\cdot 5\cdot 7}{2\cdot 4\cdot 6\cdot 8}X^4 - \ldots,$$

используя рекуррентную формулу для вычисления члена ряда,

* абсолютную и относительную ошибки приближенного значения.

16. Вычислить с точностью ε

* приближенное значения π по формуле

$$S = 4 \cdot (1 - \frac{1}{3} + \frac{1}{5} - ... + (-1)^{N-1} \frac{1}{2N-1} + ...) ,$$

используя смешанный способ вычисления члена ряда,

* точное значение π с помощью стандартной функции Pi,

* абсолютную и относительную ошибки приближенного значения.

17. Вычислить с точностью ε

* приближенное значения $\frac{1}{\sqrt{1-X^2}}$ по формуле

$$S = 1 + \frac{1}{2} X^2 + \frac{1 \cdot 3}{2 \cdot 4} X^4 + \frac{1 \cdot 3 \cdot 5}{2 \cdot 4 \cdot 6} X^6 + \frac{1 \cdot 3 \cdot 5 \cdot 7}{2 \cdot 4 \cdot 6 \cdot 8} X^8 + ... ,$$

используя рекуррентную формулу для вычисления члена ряда,

* точное значение $\frac{1}{\sqrt{1-X^2}}$ функции,

* абсолютную и относительную ошибки приближенного значения.

18. Вычислить с точностью ε сумму бесконечного ряда

* $$S = \sum_{N=0}^{\infty} (-1)^{N-1} \frac{X^{2N+1}}{2N+1} ,$$

* используя смешанный способ вычисления члена ряда,

* используя общую формулу для вычисления члена ряда.

19. Вычислить с точностью ε сумму бесконечного ряда

* $$S = \sum_{N=0}^{\infty} (-1)^{N} \frac{X^{2N-1}}{4N^2-1} ,$$

* используя смешанный способ вычисления члена ряда,

* используя общую формулу для вычисления члена ряда.

20. Вычислить с точностью ε сумму бесконечного ряда

* $$S = \sum_{N=1}^{\infty} (-1)^{N+1} \frac{X^{2N} \ln(1+2N)}{4N^2} ,$$

* используя смешанный способ вычисления члена ряда,

* используя общую формулу для вычисления члена ряда.

21. Вычислить с точностью ε сумму бесконечного ряда

* $$S=\sum_{N=1}^{\infty}(-1)^{N+1}\frac{X^{2N}(2^N-1)}{2^N(2N-1)},$$

* используя смешанный способ вычисления члена ряда,

* используя общую формулу для вычисления члена ряда.

22. Вычислить с точностью ε сумму бесконечного ряда

* $$S=\sum_{N=1}^{\infty}(-1)^{N+1}\frac{(2^{2N}-1)X^{-2N}}{2^{2N}2N},$$

* используя смешанный способ вычисления члена ряда,

* используя общую формулу для вычисления члена ряда.

23. Вычислить с точностью ε сумму бесконечного ряда

* $$S=\sum_{N=1}^{\infty}(-1)^{N}\frac{(2N-1)X^{-N}}{(N+5)2N},$$

* используя смешанный способ вычисления члена ряда,

* используя общую формулу для вычисления члена ряда.

24. Вычислить с точностью ε сумму бесконечного ряда

* $$S=\sum_{N=1}^{\infty}\frac{X^{3N}(3N+1)}{N!},$$

* используя смешанный способ вычисления члена ряда,

* используя рекуррентную формулу для вычисления члена ряда.

25. Вычислить с точностью ε сумму бесконечного ряда

$$S=\frac{1\cdot 2\cdot X}{1}+\frac{3\cdot 4\cdot X^3}{3}+\frac{5\cdot 6\cdot X^5}{3\cdot 7}+\frac{7\cdot 8\cdot X^7}{3\cdot 7\cdot 11}+\frac{9\cdot 10\cdot X^9}{3\cdot 7\cdot 11\cdot 15}+\ldots+$$
$$+\frac{11\cdot 12\cdot X^{11}}{3\cdot 7\cdot 11\cdot 15\cdot 20}+\ldots+\frac{(2N-1)(2N)X^{2N-1}}{1\cdot 3\cdot 7\cdot 11\cdot 15\cdot\ldots\cdot(4N-5)}+\ldots\ ,$$

* используя смешанный способ вычисления члена ряда,

* используя рекуррентную формулу для вычисления члена ряда.

* Вычислить с точностью ε сумму бесконечного ряда

$$S=\frac{1}{3\lg X}+\frac{1+2}{(3\lg X)^2}+\frac{1+2+3}{(3\lg X)^3}+\ldots+\frac{1+2+\ldots+N}{(3\lg X)^N}+\ldots\ ,$$

* используя смешанный способ вычисления члена ряда.

26. Вычислить с точностью ε сумму бесконечного ряда

$$S = X + \frac{X^3}{Y^3 + X} + \frac{X^5}{Y^6 + X^2} + \ldots + \frac{X^{2N+1}}{Y^{3N} + X^N} + \ldots,$$

* используя рекуррентную формулу для вычисления члена ряда,
* используя смешанный способ вычисления члена ряда.

4.18. Уточнение корней уравнений

Для численного решения алгебраических уравнений разработано множество *итерационных методов* (методов последовательного приближения к точному значению) уточнения корня. Задача ставится так: при заданном одном или двух (зависит от метода) начальных приближениях корня уравнения F(X)=0 получить приближение корня с заданной точностью ε.

Требуемая точность определяет условие завершения итерационного процесса, которое задаётся отношением $|X_n-X_{n-1}| < \varepsilon$, где X_n и X_{n-1} – соседние приближения корня, полученные на (n-1) –м и n–м шагах его уточнения, а начальные (грубые) приближения корней можно найти, например, по результатам табулирования функции F(X).

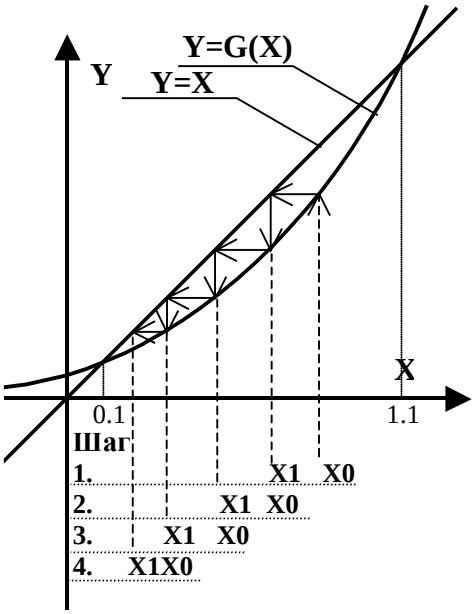

Рис. 3.1
Метод простых итераций

Метод простых итераций

Для уточнения корня уравнения вида F(X)=0 его следует преобразовать к уравнению X=G(X). Исходными данными для уточнения корня являются требуемая точность ε и начальное приближение X0. Очередное приближение X1 корня вычисляется на основе текущего приближения X0 по формуле X1=G(X0) (на первом шаге уточнения корня X0 представляет начальное приближение), после чего X0 получает значение X1 и процесс повторяется, пока модуль разности между X0 и X1 больше ε. Применение метода приводит к решению, если $|G'(X0)|<1$ внутри интервала, содержащем корень уравнения.

Пример. Пусть известно, что при заданном начальном приближении корня X0 метод простых итераций обеспечит получение решения уравнения $X=(X-0,1)^4+0,1$. Тогда для нахождения корня с заданной точностью ε можно использовать следующий фрагмент программы.

```
#include "stdafx.h"
#include "math.h"
. . . . .
  double X0, X1, Eps, dX;
  scanf("%lf%lf",&X0,&Eps);
  do
  {
```

```
    X1=pow(X0-0.1, 4)+0.1;
    dX=fabs(X0-X1);
    X0=X1;
  }
  while(dX>Eps);
  printf("X0 = %lf\n",X0);
  . . .
```

Метод не всегда обеспечивает нахождение корня. Так, при $\varepsilon = 10^{-3}$ и X0<1,1, где в окрестности корня 0,1 $|G'(X)| = |4(X-0,1)^3| < 1$ будет найден этот корень (как будет проходить уточнение корня, показано стрелками на рис.3.1). Но в окрестности корня 1,1 (1,1 – второй корень уравнения), где $|G'(X)| > 1$ каждый шаг процесса будет приводить к удалению от корня (при X0>1,1), что, в конечном счете, приведет к переполнению разрядной сетки машины и результатом будет значение 1.#INF00, обозначающее бесконечность.

Чтобы найти корень уравнения X=G(X) при условии |G'(X)|>1, его следует преобразовать к виду X=H(X), где H(X) - обратная относительно G(X) функция, и использовать для поиска корня.

Пример. С учетом сказанного, изменим текст фрагмента программы, чтобы была возможность находить все корни уравнения $X=(X-0,1)^4 +0,1$, при любых начальных приближениях, когда $|G'(X)| \neq 0$.

```
#include "math.h"
. . . . . .
double X0=1.01, X1, Eps=1e-3, dX, P;
scanf("%lf%lf",&X0,&Eps);
P=fabs(4*pow(X0-0.1,3));
if (P<1)
      //Использование исходной функции
            do
            {
                  X1=pow(X0-0.1, 4)+0.1;
                  dX=fabs(X0-X1);
                  X0=X1;
            }
            while(dX>Eps);
else if (P>1)
            //Использование обратной функции
            do
            {
```

```
                X1=pow(X0-0.1,0.25)+0.1 ;
                dX=fabs(X0-X1);
                X0=X1;
            }
            while(dX>Eps);
        printf("X0 = %lf\n",X0);
```

В некоторых случаях возможно зацикливание – бесконечное выполнение цикла программы (например, для уравнения X =1/X) или медленная сходимость процесса (например, для уравнения $X = 1/(X-10^{-6})$).

Чтобы обеспечить информативность программ при зацикливании или очень медленной сходимости, в программах вводят ограничения на число итераций, и если за заданное число шагов заданная точность не достигается, то процесс останавливается и выдается соответствующее сообщение.

Пример. Составить фрагмент программы для решения следующей задачи. Найти корень уравнения $X = 1/(X-10^{-6})$ с заданной точностью ε. Если за заданное число N шагов точность не будет достигнута, то прекратить выполнение программы и вывести с пояснениями модуль разности между двумя последними приближениями, значение и значение N, иначе вывести найденное значение корня и число шагов, за которое оно было найдено.

```
#include "math.h"
    . . . . .
    double X0, X1, Eps;
    int I, N;
    scanf("%lf%lf%d",&X1,&Eps,&N);
    for (I=1;I<N;I++)
    {
        X0=X1;
        X1=1/(X0-1E-6);
        if (fabs(X0-X1)<Eps)
            //Решение найдено
            break;//Выход из цикла
        }
    if (fabs(X0-X1)<Eps)
        printf(" X0 = %lf  I = %d\n",X0,I);
    else
        printf(" Error: I = %d\n",I);
    . . . . .
```

Выполнив эту программу при X0=0,5, $\varepsilon = 10^{-3}$, N=5000000 можно убедиться, что после 5000000 итераций заданная точность достигнута не будет.

Метод половинного деления

Исходными данными для уточнения корня уравнения вида F(X)=0 являются требуемая точность и два начальных приближения: XL и XR, между которыми должен находиться корень. Поэтому необходимым условием применения метода является истинность отношения F(XL)·F(XR)<0, то есть метод не пригоден в тех случаях, когда график F(X) лишь касается оси абсцисс, не пересекая её, например, в случае уравнения $X^2=0$.

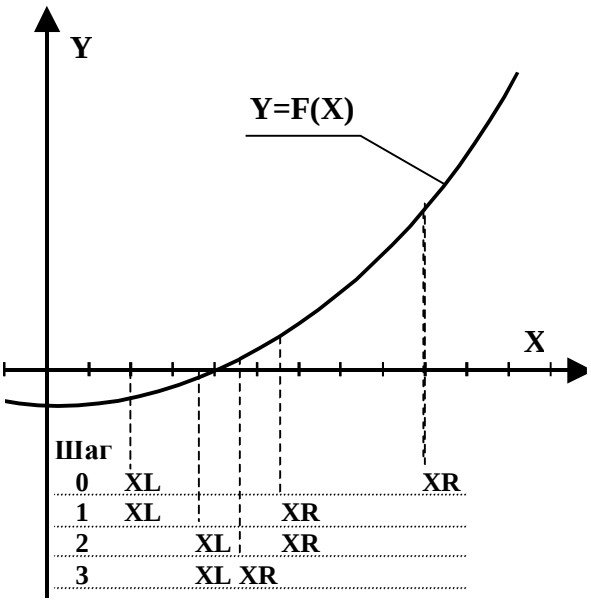

Рис. 3.2
Метод половинного деления

Один шаг итерационного процесса уточнения корня состоит в перемещении правой (XR) или левой (XL) границы отрезка (XL,XR) в его середину в соответствии со следующим правилом: если знак F((XR+XL)/2) совпадает со знаком F(XL), то XL получит значение (XR+XL)/2, иначе это значение получит XR (см. рис. 3.2). Процесс повторяется, пока модуль разности между XR и XL больше ε.

Пример. Составить фрагмент программы уточнения корня уравнения arctg(X)-X=0 с заданной точностью ε при начальных приближениях корня XL и XR методом половинного деления.

```
#include "math.h"
      . . . . .
      double X, XL, XR, Y, YL, Eps;
      scanf("%lf%lf%lf",XL,XR,Eps);
      YL=atan(XL)-XL;
      do
      {
            X=(XL+XR)/2;
            Y= atan(X)-X;
            if (Y*YL>0)
                  XL=X;
            else
                  XR=X;
```

```
}
while(fabs(XR-XL)>Eps);
printf("X = %lf\n",X);
. . . . .
```

Метод касательных

Исходными данными для уточнения корня уравнения вида F(X)=0 являются требуемая точность ε и начальное приближение X0. Необходимым условием применения метода является истинность отношения F(X0)·F"(X0)>0.

Один шаг итерационного процесса уточнения корня состоит в вычислении очередного приближения по формуле X1= X0 - F(X0)/F'(X0), после чего X0 получает значение X1 (см. рис. 3.3). Процесс повторяется, пока модуль разности между X0 и X1 больше ε.

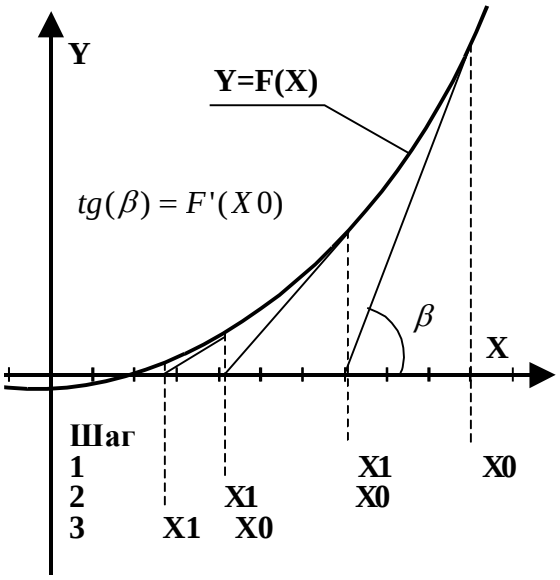

Рис. 3.3
Метод касательных

Пример. Составить фрагмент программы уточнения корня уравнения

$$(X-0,1)^4 - X + 0,1=0$$

с заданной точностью ε при начальном приближении корня X0.

```
#include "math.h"
. . . . .
double X1, X0, dX, Eps;
scanf("%lf%lf",&X0, &Eps);
do
{
      dX=(pow(X0-0.1, 4)-X0+0.1)/(4*pow(X0-0.1, 3)-1);
      X1=X0-dX;
      X0=X1;
}
while (fabs(dX)>Eps);
printf("X = %lf\n",X0);
```

В этом фрагменте использовалась найденное заранее выражение $4(X-0,1)^3 - 1$ первой производной для $(X-0,1)^4 - X + 0,1$. С точки зрения объема и точности вычислений

такое решение предпочтительнее использования для этих целей разностного отношения, как, например, в следующем операторе

```
dX1=(pow(X0-0.1, 4) - X0+0.1)
    /(( (pow(X0+1e-8-0.1, 4) - X0+1e-8+0.1)
       -( pow(X0-0.1,4) - X0+0.1) )/1e-8);
```

где для вычисления приближенного значения производной использовалась формула $F'(X0) \approx \frac{F(X0+\Delta X)-F(X0)}{\Delta X}$ и $\Delta X=10^{-8}$.

4.19. Пример выполнения задания

Программа составлена по условию варианта задания №30 (см. ниже). В реализации метода касательных используются выражения производной $9X^2$-10X+1 и второй производной 18X-10 выражения, входящего в уравнение $3X^3$-$5X^2$+X+0,4=0.

```
#include "stdafx.h"
#include "math.h"
#include "conio.h"

int _tmain(int argc, _TCHAR* argv[])
{
      //Переменные будут представлять во обоих методах
      int    i    //число итераций,
            ,N;   //допустимое число итераций,
      Double X    //значение очередного приближения корня
                  //и результат вычислений,
            ,Eps  //требуемую точность вычисления корня уравнения
            ,dX   //разности между последним и предыдущим
                  //приближениями и её абсолютное значение,
                  //в методе касательных
            ,X0   //начальное и текущее приближение корня,
                  //в методе половинного деления
            ,XL, XR //начальные и текущие приближения корня,
            ,YL //значение функции в точке XL начального приближения,
            ,Y; //значение функции в точке X найденного приближения.
      printf("Metod kasatelnyh\n");
      printf("Wwedite X0, Eps, N: ");
      scanf("%lf%lf%d",&X0,&Eps,&N);
      //Проверка применимости метода касательных
```

```
            if  ((3*pow(X0,3)-5*X0*X0+X0+0.4)*(18*X0-10) <= 0)
                  printf("Metod kasatelnyh ne primenim!\n");
            else
            {
                  i=0;
                  do
                  {
                        dX = (3*pow(X0,3)-5*X0*X0+X0+0.4)
                                    /(9*X0*X0-10*X0+1);
                        X=X0-dX;
                        X0=X;
                        i++;
                        dX=fabs(dX);
                  }
                  while (dX>Eps && i<N);
                  if (dX<Eps)
                        printf("Koren X=%lf nayden za %d\
iteraciy,\nY(X)=%lf\n"
                              ,X,i,3*pow(X0,3)-5*X0*X0+X0+0.4);
                  else
                        printf("Koren ne nayden za %d iteraciy\n",N);
            }

        //Метод половинного деления
            printf("\nMetod polovinnogo delenia\n");
            printf("Wwedite XL, XR, Eps, N: ");
            scanf("%lf%lf%lf%d",&XL,&XR,&Eps,&N);
        //Проверка применимости метода
            if ((3*pow(XL,3)-5*XL*XL+XL+0.4)*(3*pow(XR,3)-5*XR*XR+XR+0.4) > 0)
                  printf("Metod polovinnogo delenia ne primenim!\n");
            else
            {
                  YL=3*pow(XL,3)-5*XL*XL+XL+0.4;
                  i=0;
                  do
                  {
                        X=(XL+XR)/2;
                        Y= 3*pow(X,3)-5*X*X+X+0.4;
                        if (Y*YL>0)
                              XL=X;
                        else
                              XR=X;
```

```
                        i++;
                        dX=fabs(XR-XL);
                  }
                  while (dX>Eps && i<N);
                  if (dX<Eps)
                        printf("Koren X=%lf nayden za %d\
iteraciy,\nY(X)=%lf\n"
                              ,X,i,Y);
                  else
                        printf("Koren ne nayden za %d iteraciy\n",N);
            }
      getch();
      return 0;
      }
```

4.20. Задания для самостоятельной работы

Составить программу нахождения корня уравнения (см. ниже в таблице) с заданной точностью ε двумя указанными методами. Если за заданное число N шагов точность не будет достигнута, то вывести соответствующее сообщение, иначе – вывести найденное значение корня, число шагов, за которое оно было найдено, и значение функции в корне.

Перед выполнением метода проверить возможность его использования при введённом начальном приближении (для метода половинного деления – для приближения слева и справа от корня). В правом столбце таблицы для каждого уравнения приведены приближенные значения корней, на которых требуется проверить работу программы (начальное приближение для поиска корня следует брать несколько меньше и/или несколько больше такого значения). Следует иметь в виду, что не каждый метод и не при каждом начальном приближении приводит к ближайшему корню, что некоторые корни вообще не могут быть найдены методом, что разные методы при одинаковых начальных приближениях могут приводить к разным результатам, что возможны исключения, которые следует обработать не прерывая работы программы.

	Методы	Уравнение	Начальные приближения
	итераций и касательных	$0{,}75 \cdot X - \sqrt[3]{X} = 0$	-1,5; 0; 1,5
	итераций и половинного деления	$\frac{X}{2} - \sqrt[5]{X} + 0{,}2 = 0$	-2,9; 0; 2,2
	касательных и половинного деления	$X - \ln X - 2 = 0$	0,15; 3,2
	итераций и касательных	$X^3 - X = 0$	-1; 0; 1
	итераций и половинного деления	$X^5 - X + 0{,}2 = 0$	-1; 0,2; 0,95
	касательных и половинного деления	$tg\frac{X}{3} - X + 0{,}5 = 0$	-4; 0,76
	итераций и касательных	$tg\frac{X}{2} - X = 0$	-2,3; 0; 2,3
	итераций и половинного деления	$\cos(4X) - X\,0.5 = 0$	1,4; 1,7
	касательных и половинного деления	$X^5 \cdot \cos X - X + 1 = 0$	-4,7; 1,5; 4,7
0	итераций и касательных	$\frac{X^4}{8} - X\sin(14X) = 0$	0,9; 1,1; 1,38
1	итераций и половинного деления	$\sin X - \sqrt[4]{\frac{X}{2} + 2} + 2 = 0$	-1,8; -1,15
2	касательных и половинного	$X^2 - X - 2 = 0$	-1; 2

	деления		
3	итераций и касательных	$e^{X/5} - X = 0$	1,3; 12,7
4	итераций и половинного деления	$e^{X/2} - X - 3 = 0$	-2,75; 3,8
5	касательных и половинного деления	$X + \ln X = 0$	0,57
6	итераций и касательных	$X + \ln\frac{X}{3} = 0$	1
7	итераций и половинного деления	$2X - \ln(X + 2) = 0$	-1,98; 0,45
8	касательных и половинного деления	$e^{-X} - X = 0$	0,57
9	итераций и касательных	$e^{-X} - 2 + X = 0$	-1,15; 1,84
0	итераций и половинного деления	$\sin(2X) - X = 0$	-0,95; 0; 0,95
1	касательных и половинного деления	$\sin X - X + 2 = 0$	2,55
2	итераций и касательных	$e^{X} + \ln\frac{X}{10} + 2 = 0$	0,33
3	итераций и половинного деления	$e^{-3X} - \sin X - 1{,}5 = 0$	-0,17
4	касательных и половинного деления	$arctgX - 0{,}5 + (X - 1)^3 =$	0,61
	итераций и	$\arcsin X - \sqrt{X + 0{,}5} = 0$	0,93

5	касательных		
6	итераций и половинного деления	$5\sin(2X) - \ln(X+1) = 0$	-0,98; 0; 1,47
7	касательных и половинного деления	$e^{-X} - X^2 + 2 = 0$	1,49
8	итераций и касательных	$(X^2 - 1)^{-1} - 2^{1-X} = 0$	-1,1; 1,57; 6,25
9	итераций и половинного деления	$2X^2 + 5X - 10 = 0$	-3,8; 1,3
0	касательных и половинного деления	$3X^3 - 5X^2 + X + 0.4 = 0$	-0,19; 0,51; 1,3

4.21. Вычисление определённых интегралов

Для вычисления значений определённых интегралов существует множество методов. Рассмотрим три из них: – *прямоугольников*, *трапеций* и *парабол* (*метод Симпсона*) на примерах при следующей постановке задачи. Составить программу для вычисления приближенного значения определённого интеграла

$$z = \int_a^b f(x)dx$$

при заданных подынтегральной функции $f(x)$, пределах интегрирования a и b и числе N разбиений интервала на подынтервалы. При этом шаг изменения аргумента x следует найти по формуле $\Delta x = (b-a)/N$.

Суть этих методов в накоплении, с учетом знаков, сумм площадей прямоугольников, трапеций или параболических трапеций, заменяющих на каждом подынтервале в общем случае криволинейную трапецию.

Замену криволинейной трапеции прямоугольником можно осуществлять одним из трех способов. В первом случае (рис. 4.10) построение прямоугольников начинается с левой границы интервала интегрирования, при этом основание каждого прямоугольника равно Δx, а высота прямоугольника численно равна значению подынтегральной функции на левой границе подынтервала (левые прямоугольники).

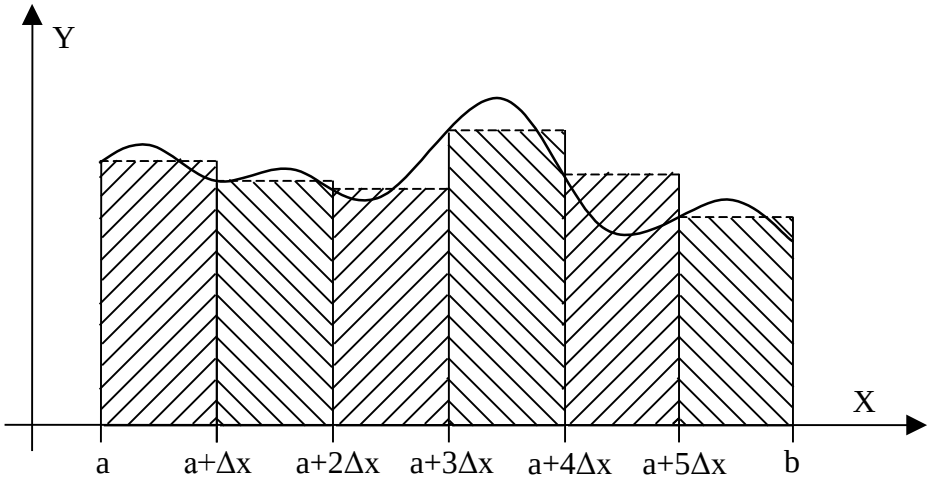

Рис.4.10

Во втором случае (рис. 4.11) построение прямоугольников начинается с правой границы интервала интегрирования, при этом основание каждого прямоугольника равно

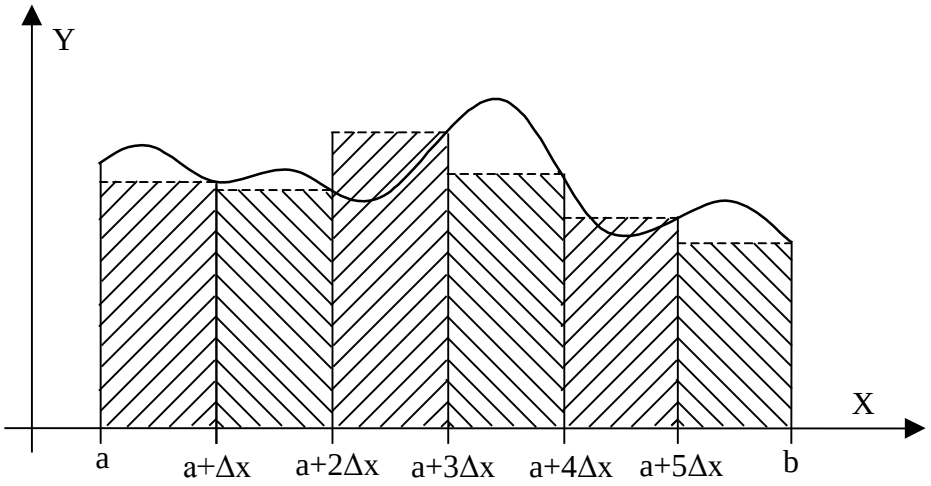

Рис.4.11

Δx, а высота прям оугольника численно равна значению подынтегральной функции на правой границе подынтервала (правые прямоугольники). Оба этих способа дают одинаковую погрешность при вычислении интеграла.

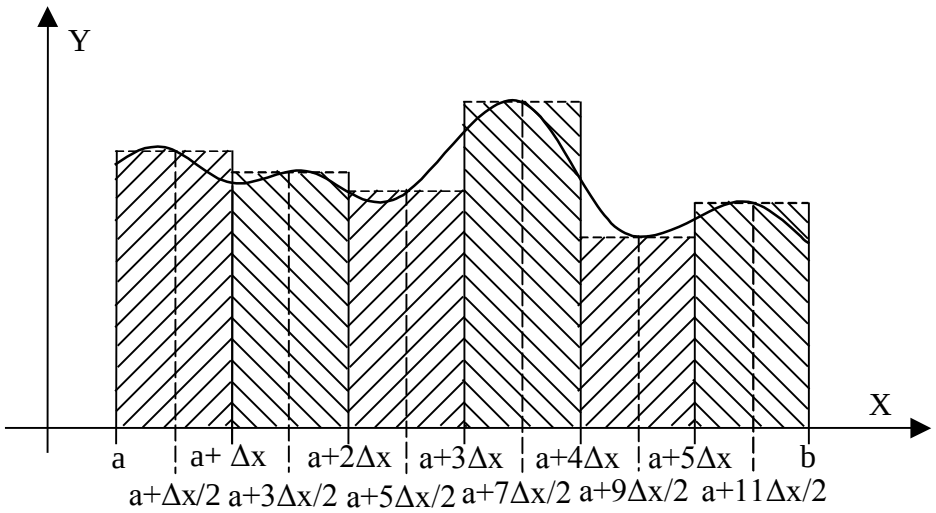

Рис. 4.12

В третьем случае (рис.4.12) верхнее основание прямоугольника проводится через точку пересечения перпендикуляра к оси абсцисс, проведенного через середину подынтервала, с кривой графика подынтегральной функции. При этом основание каждого прямоугольника равно Δx, а высота прямоугольника численно равна значению

подынтегральной функции в середине подынтервала (средние прямоугольники). Этот способ дает более точный результат и обычно применяется на практике.

В методе трапеций (рис.4.13) основания каждой трапеции образуют перпендикуляры к оси абсцисс, проведенные на концах подынтервала и заключенные между точкой

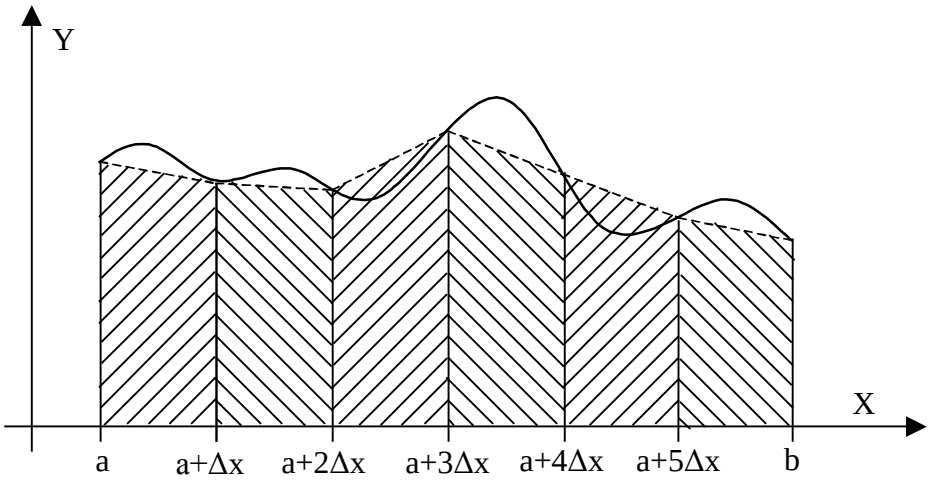

Рис. 4.13

пересечения с осью абсцисс и точкой пересечения с кривой графика подынтегральной функции. Одну боковую сторону образует отрезок оси абсцисс, а другую боковую сторону – отрезок, соединяющий точки пересечения оснований с кривой графика функции.

Метод парабол (рис.4.14) во многом совпадает с методом трапеций, отличие состоит в том, что точки пересечения перпендикуляров с кривой графика подынтегральной функции

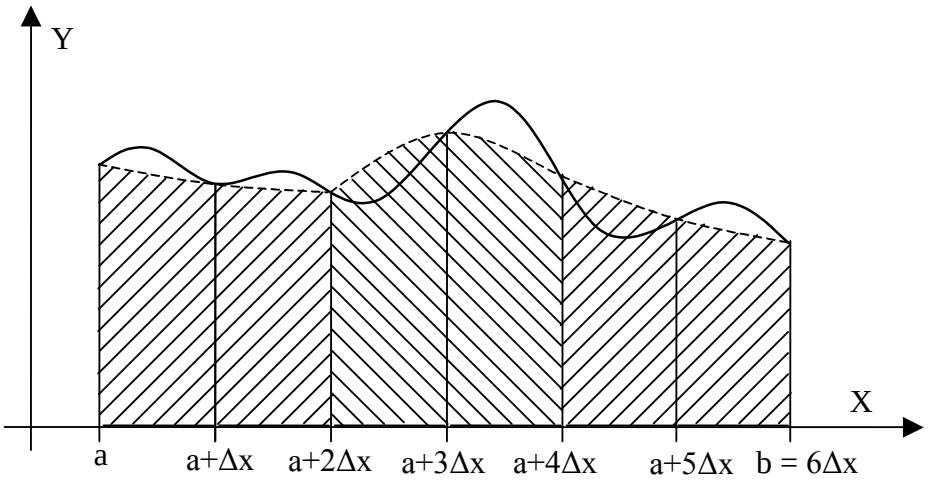

Рис. 4.14

соединяются не отрезком прямой, а дугой параболы, которая проходит через три точки, являющиеся точками пересечения перпендикуляров к оси абсцисс, проведенных через концы и середину подынтервала, с кривой графика функции.

Пример 1. Использование метода прямоугольников с вычислением высот прямоугольников в серединах подынтервалов.

В этом методе формула приближенного значения определённого интеграла представляется в виде

$$z = \sum_{i=1}^{N} f(x_i)\Delta x,$$

$$где\ x_i = a + \Delta x/2 + (i-1)\cdot\Delta x.$$

Для уменьшения объёма вычислений множитель Δx следует вынести за знак суммы:

$z = \Delta x \sum_{i=1}^{N} f(x_i)$, а для вычисления текущих значений центров x_i подынтервалов будем использовать приём накопления суммы:

```
#include "stdafx.h"
#include "math.h"
#include "conio.h"

int _tmain(int argc, _TCHAR* argv[])
{
	double const pi = 3.1415926535;
	//Переменные программы представляют:
	double	 a=0	//нижний предел интегрирования,
		,b=2*pi	//верхний предел интегрирования,
		,x	//текущее значение аргумента,
		,dx	//приращение аргумента,
		,z;	//вычисленное значение интеграла,
	int	 i	//параметр цикла
		,N=100;	//количество подынтервалов.
	//Начальные значения переменных использовать для отладки,
	//закомментировав ввод их значений:
	//printf("wwedite a, b, N: ");
	//scanf("%lf%lf%d", &a, &b,&N);
	z=0;
	dx=(b-a)/N;
```

```
	x=a+dx/2;  //Середина первого подынтервала
	for (i=1;i<=N;i++)
	{
		z=z+sin(x);
		x=x+dx;
	}
	z=z*dx;
	printf("z=%lf\n",z);
	getch();
	return 0;
}
```

Пример 2. Использование метода трапеций.

В этом методе формула приближенного значения определённого интеграла представляется в виде

$$z=\sum_{i=0}^{N-1}\frac{f(x_i)+f(x_{i+1})}{2}\Delta x,$$

$$где\quad x_i=a+i\cdot\Delta x,\quad x_{i+1}=a+(i+1)\cdot\Delta x.$$

Преобразование её к виду

$$z=\sum_{i=0}^{N-1}\frac{f(x_i)+f(x_{i+1})}{2}\Delta x=\left[\sum_{i=0}^{N-1}\frac{f(x_i)}{2}+\sum_{i=1}^{N}\frac{f(x_i)}{2}\right]\Delta x=\left[\frac{f(a)+f(b)}{2}+\sum_{i=1}^{N-1}f(x_i)\right]\Delta x$$

позволяет исключить повторные вычисления высот трапеций на внутренних подынтервалах и таким образом сократить объём вычислений:

```
#include "stdafx.h"
#include "math.h"
#include "conio.h"

int _tmain(int argc, _TCHAR* argv[])
{
	double const pi = 3.1415926535;
	double a, b, x, dx, z;
	int    i, N;
	printf("wwedite a, b, N: ");
```

```
	scanf("%lf%lf%d", &a, &b,&N);
	z=(sin(a)+sin(b))/2;
	dx=(b-a)/N;
	x=a+dx;
	for (i=1;i<N;i++)
	{
		z=z+sin(x);
		x=x+dx;
	}
	z=z*dx;

	printf("z=%lf\n",z);
	getch();
	return 0;
}
```

Пример 3. Использование метода параболических трапеций (Симпсона).

В этом методе формула приближенного значения определённого интеграла представляется в виде

$$z=\frac{\Delta x}{3}\left[\frac{f(a)+f(b)}{2}+\sum_{i=1}^{N-1}f(x_i)+2\sum_{i=1}^{N}f(x_i-\Delta x/2)\right],\ x_i=a+i\cdot\Delta x$$

или, взяв N в 2 раза большим, то есть разбив весь интервал на четное количество участков, в 2 раза меньшей длины:

$$z=\frac{\Delta x}{3}\left[f(a)+f(b)+4\sum_{i=1}^{N/2}f(x_{2i-1})+2\sum_{i=1}^{N/2-1}f(x_{2i})\right],$$

$$\text{где}\quad x_{2i-1}=a+(2i-1)\cdot\Delta x;\quad x_{2i}=a+2i\Delta x$$

Используем вторую формулу в следующем программе.

```
#include "stdafx.h"
#include "math.h"
#include "conio.h"

int _tmain(int argc, _TCHAR* argv[])
{
	double const pi = 3.1415926535;
	double a, b, x, dx, z;
	int    i, N;
```

```
	printf("wwedite a, b, N: ");
	scanf("%lf%lf%d", &a, &b,&N);
	z=sin(a);
	dx=(b-a)/N;
	for (i=1;i<=N/2;i++)
	{
		x=a+2*i*dx;
		z=z+2*sin(x)+4*sin(x-dx);
	}
	z=(z-sin(b))*dx/3;

	printf("z=%lf\n",z);
	getch();
	return 0;
}
```

4.22. Пример выполнения задания

Составить программу вычисления приближенных значений определённого интеграла $\int_a^b e^{q\cdot x}(q^2\cos x - 2q\cdot\sin x - \cos x)dx$ методом прямоугольников и методом Симпсона, а также точное его значение по первообразной $e^{qx}(q\cdot\cos x - \sin x)$. Вычислить также абсолютную и относительную ошибки для каждого приближенного метода. Пределы интегрирования a и b, а также число N подынтервалов задавать при вводе. Для метода Симпсона используем первую формулу из предыдущего примера.

```
#include "stdafx.h"
#include "math.h"
#include "conio.h"

int _tmain(int argc, _TCHAR* argv[])
{
	//Переменные программы представляют:
	double	A		//нижний предел интегрирования,
		,B		//верхний предел интегрирования,
		,Q		//коэффициет,
		,X		//текущее значение аргумента,
		,dX		//приращение аргумента,
```

```
                ,dX2        //половина приращения аргумента,
                ,Z          //приближенное значение интеграла,
                ,Z0;        //точное значение интеграла,
      int       i           //параметр цикла
               ,N;          //количество подынтервалов.
      //Ввод исходных данных
      printf("wwedite A, B, Q и N : ");
      scanf("%lf%lf%lf%d", &A, &B, &Q, &N);
      //Вывод исходных данных для контроля
      printf("A=%lf     B=%lf Q=%lf N=%d\n" ,A, B, Q ,N );
//МЕТОД ПРЯМОУГОЛЬНИКОВ
      Z=0;
      dX=(B-A)/N;
      X=A+dX/2;
      for (i=1;i<N;i++)
      {
            Z=Z+exp(Q*X)*(Q*Q*cos(X)-2*sin(X)*Q-cos(X));
            X=X+dX;
      }
      Z=Z*dX;
      printf("METOD PRAMOUGOLNIKOV\n");
      printf("%lf - pribligennoe znachenie inegrala\n",Z);
      Z0=exp(Q*B)*(Q*cos(B)-sin(B))
         -exp(Q*A)*(Q*cos(A)-sin(A));
      printf("%lf - tochnoe znachenie inegrala\n",Z0);
      printf("%e - absolutnaia oshibka\n",fabs(Z0-Z));
      printf("%e - otnositelnaia oshibka\n",fabs((Z0-Z)/Z0));
//МЕТОД ПАРАБОЛ
      dX=(B-A)/N;
      dX2=dX/2;
      Z=( exp(Q*A)*(Q*Q*cos(A)-2*sin(A)*Q-cos(A))
         +exp(Q*B)*(Q*Q*cos(B)-2*sin(B)*Q-cos(B)))/2
        +2*exp(Q*(B-dX2))
        *(Q*Q*cos(B-dX2)-2*sin(B-dX2)*Q-cos(B-dX2));
      X=A+dX;
      for (i=1;i<=N-1;i++)
      {
            Z=Z+exp(Q*X)
                    *(Q*Q*cos(X)-2*sin(X)*Q-cos(X))
               +2*exp(Q*(X-dX2))
                  *(Q*Q*cos(X-dX2)-2*sin(X-dX2)*Q-cos(X-dX2));
            X=X+dX;
```

```
    }
    Z=Z*dX/3;
    printf("\nMETOD PARABOL\n");
    printf("%lf - pribligennoe znachenie inegrala\n",Z);
    printf("%lf - tochnoe znachenie inegrala\n",Z0);
    printf("%e - absolutnaia oshibka\n",fabs(Z0-Z));
    printf("%e - otnositelnaia oshibka\n",fabs((Z0-Z)/Z0));
    getch();
    return 0;
}
```

4.23. Задания для самостоятельной работы

Составить программу вычисления приближенных значений определённого интеграла (см. таблицу ниже) двумя предложенными методами, а также точное его значение по первообразной. Вычислить также абсолютную и относительную ошибки для каждого приближенного метода. Пределы интегрирования a и b, а также число N подынтервалов задавать при вводе.

Выполняя программу при вводимых N = 10k, k=1, 2, …,8, установить зависимость величин ошибок от N.

№	Использовать методы	подынтегральная функция	Первообразная подынтегральной функции
1	парабол и прямоугольников	$\frac{1}{1-x^2}$	$\frac{1}{2}\ln\frac{x+1}{1-x}$
2	трапеций и парабол	$\sin^2 x$	$\frac{1}{2}x-\frac{1}{4}\sin 2x$
3	трапеций и прямоугольников	$\frac{1}{X}$	$\ln\lvert X\rvert$
4	парабол и прямоугольников	$\sqrt{\frac{x+1}{1-x}}$	$-\sqrt{1-x^2}+\arcsin x$

5	трапеций и парабол	$\sin 3x \cdot \cos 2x$	$-\frac{\cos 5x}{10} - \frac{\cos x}{2}$
6	трапеций и прямоугольников	$\frac{1}{x\sqrt{x^2+1}}$	$\ln \frac{x}{1+\sqrt{x^2+1}}$
7	парабол и прямоугольников	$\ln^2 x / x$	$\ln^3 x / 3$
8	трапеций и парабол	$e^x \sin x$	$e^x(\sin x - \cos x)/2$
9	трапеций и прямоугольников	$\frac{1}{1+x^2}$	arctg x
10	парабол и прямоугольников	$\frac{x}{(1+x)^3}$	$\frac{1}{2(1+x)^2} - \frac{1}{1+x}$
11	парабол и прямоугольников	$\sin^3 x \cos x$	$\frac{\sin^4 x}{4}$
12	трапеций и парабол	$(e^x - e^{-x})/2$	$(e^x + e^{-x})/2$
13	трапеций и прямоугольников	$tg\ x$	$-\ln \cos x$
14	парабол и прямоугольников	$x\sqrt{x^2+1}$	$\sqrt{(x^2+1)^3}/3$
15	трапеций и парабол	$\frac{1}{x(1+x^2)}$	$\frac{1}{2}\ln \frac{x^2}{1+x^2}$
16	трапеций и прямоугольников	$\sqrt{1+x}$	$2\sqrt{(1+x)^3}/3$

17	парабол и прямоугольников	$\sin x$	$-\cos x$
18	трапеций и парабол	8^x	$8^x / \ln 8$
19	парабол и прямоугольников	$\frac{1}{x(1+x)}$	$-\ln\frac{1+x}{x}$
20	парабол и прямоугольников	$1/\sqrt{1-x^2}$	$\arcsin x$
21	трапеций и парабол	$\ln x$	$x\ln x - x$
22	трапеций и прямоугольников	$\frac{e^x - e^{-x}}{e^x + e^{-x}}$	$\ln\frac{e^x + e^{-x}}{2}$
23	парабол и прямоугольников	$\frac{1}{x^2\sqrt{x^2-1}}$	$\frac{\sqrt{x^2-1}}{x}$
24	трапеций и парабол	$x^3 \ln x$	$x^4\left(\frac{\ln x}{4} - \frac{1}{16}\right)$
25	трапеций и прямоугольников	$\frac{x}{1+x}$	$1 + x - \ln(1+x)$
26	трапеций и парабол	$\frac{\sin x}{1+\cos x}$	-ln|1+cosx|
27	парабол и прямоугольников	$\ln(x+\sqrt{x^2+4})$	$-\sqrt{x^2+4} + x\ln(x+\sqrt{x^2+4})$)
28	трапеций и прямоугольников	$\frac{1}{X \ln X}$	ln(lnX)
29	трапеций и	$\frac{1}{\sin x}$	$\ln\left\|\mathrm{tg}\frac{x}{2}\right\|$

	парабол		
30	парабол и прямоугольников	$-2e^x \sin x$	$e^x(\cos x - \sin x)$

5. Организация программ со структурой вложенных циклов

Структурой с вложенным циклом называют такую структуру, в которой внутри одного цикла находится один или несколько других циклов. Цикл, расположенный внутри другого цикла, называют внутренним. Цикл, внутри которого находятся другие циклы, называют внешним. Таким образом, один и тот же цикл может выступать и в роли внешнего (если он содержит внутри себя другие циклы), и в роли внутреннего (если он расположен внутри другого цикла). Правильная организация *вложенного цикла* состоит в том, что *внутренний цикл* должен целиком располагаться внутри *внешнего цикла*.

Допустимыми являются следующие варианты организации вложенных циклов.

Первый вариант вложенного цикла – внутри внешнего цикла последовательно расположено несколько внутренних циклов:

```
for (i=1;i<=n;i++)  // внешний цикл
{
        . . . . . .
      for (j=1;j<=m;j++) // первый внутренний цикл
      {
           . . . . . .
      }// конец первого внутреннего цикла
        . . . . . .
      for (k=1;k<=l;k++)  // второй внутренний цикл
      {
           . . . . . .
      }// конец второго внутреннего цикла
        . . . . . .
}// конец внешнего цикла
```

Второй вариант организации вложенного цикла – иерархическое расположение циклов (каждый внутренний цикл расположен внутри предыдущего).

```
for (i=1;i<=n;i++) // внешний цикл
{
      . . . . . .
      for (j=1;j<=m;j++) // первый внутренний цикл
      {
            . . . . . .
            for (k=1;k<=l;k++) // второй внутренний цикл
            {
                  . . . . . .
            }// конец второго внутреннего цикла
            . . . . . .
      }// конец первого внутреннего цикла
      . . . . . .
}// конец внешнего цикла
```

Частным случаем второго варианта организации вложенных циклов является следующий

```
for (i=0; i<n;i++)
      for (j=0;j<m;j++)
            for (k=0;k<l;k++)
                  a[i][j][k]=i*j*k;
```

В приведенном примере все три цикла оканчиваются в одном и том же месте.

Следует иметь в виду, что во вложенном цикле параметры каждого из циклов изменяются разновременно, то есть при очередном (фиксированном) значении параметра внешнего цикла параметр внутреннего цикла последовательно принимает все свои возможные значения. Затем параметр внешнего цикла принимает следующее по порядку новое значение и при этом значении параметр внутреннего цикла вновь принимает все свои возможные значения. Поэтому вложенный цикл часто называют циклом с разновременно изменяющимися параметрами. Таким образом, если внутренний цикл должен выполняться M раз, а внешний – N раз, то операторы, стоящие во внутреннем цикле выполнятся в общей сложности M·N раз, т.е. трудоемкость выполнения вложенных циклов может быть весьма высокой, что надо учитывать при разработке программ. Например, если M=N=100, то операторы внутреннего цикла будут выполняться 10000 раз:

```
//Вычисление и вывод таблицы умножения чисел от 1 до 100
for (i=1;i<=100;i++)
      for (j=1;j<=100;j++)
      {
            k=i*j;      // умножение выполняется 10000 раз
            printf("K=%6d",k);
      }
```

Во вложенных циклах допустимо передавать управление из внутреннего цикла в любую точку внешнего цикла. Передача управления из внешнего цикла в произвольную точку внутреннего цикла запрещена, так как в этом случае вход во внутренний цикл осуществляется не через его начало. Такая передача управления в языке С возможна только с использованием оператора безусловного перехода. Поскольку применение этого оператора не отвечает принципам структурного программирования, то его, как правило, в программах не используют и, следовательно, подобные структуры встречаться не будут.

При организации вложенных циклов необходимо обращать внимание на правильное задание вложенности циклов. В некоторых программах порядок вложенности

циклов не оказывает влияния на правильность получаемого результата. Например, при вычислении суммы всех элементов матрицы порядок следования циклов может быть произвольным:

Вариант 1

```
s=0;
for (i=0;i<m;i++)
      for (j=0;j<n;j++)
            s+=a[i][j];
```

Вариант 2

```
s=0;
for (j=0;j<n;j++)
      for (i=0;i<m;i++)
            s+=a[i][j];
```

Однако при решении многих задач изменение порядка вложенности циклов приводит к неверному результату. Например, при вычислении суммы элементов каждой строки матрицы изменение порядка расположения циклов приведет к тому, что будут вычисляться суммы элементов столбцов.

Вариант 1 (правильный)

```
for (i=0;i<m;i++)
{
      s=0;
      for (j=0;j<n;j++)
            s+=a[i][j];
      printf("S=%7.2f",s);
}
```

Вариант 2 (неправильный)

```
for (j=0;j<n;j++)
{
      s=0;
      for (i=0;i<m;i++)
            s+=a[i][j];
      printf("S=%7.2f",s);
}
```

Необходимость организации вложенных циклов возникает при решении таких широко распространенных задач, как вычисление определенного интеграла с заданной точностью, вычисление экстремума функции на заданном интервале с заданной точностью. Рассмотрим решение этих задач.

5.1.Вычисление определенного интеграла с заданной точностью

Задача вычисления определенного интеграла формулируется следующим образом: вычислить $I=\int_a^b f(x)dx$ с точностью ε при известных значения пределов интегрирования a, b, известной точности ε и заданной подынтегральной функции f(x). При вычислении интеграла с точностью будут использоваться изложенные ранее методы прямоугольников, трапеций и парабол для вычисления интеграла при заданном числе разбиений интервала интегрирования. Для оценки погрешности вычисления интеграла на практике используют правило [4]. Суть правила состоит в том, что выполняют вычисление интеграла с двумя разными шагами изменения переменной x, а затем сравнивают результаты вычислений и

получают оценку точности. Наиболее часто используемое правило связано с вычислением интеграла дважды: с шагом h и шагом h/2.

Для методов прямоугольников и трапеций погрешность $R_{h/2}$ вычисления интеграла с шагом h/2 оценивается следующей формулой:

$$|R_{h/2}| = \frac{|I_{h/2} - I_h|}{3},$$

где $I_{h/2}$ – значение интеграла, вычисленное с шагом h/2;

I_h – значение интеграла, вычисленное с шагом h.

Для метода Симпсона погрешность оценивается в соответствии со следующим выражением:

$$|R_{h/2}| = \frac{|I_{h/2} - I_h|}{15}.$$

Шаг интегрирования для методов прямоугольников и трапеций пропорционален величине $\sqrt{\varepsilon}$, поэтому в [2] рекомендовано начальное количество разбиений выбирать согласно следующему выражению n=] $\frac{b-a}{\sqrt{\varepsilon}}$ [+ 1, где][- целая часть. Для метода парабол шаг интегрирования пропорционален величине $\sqrt[4]{\varepsilon}$, поэтому начальное количество шагов рекомендовано выбирать из следующего выражения n=] $\frac{b-a}{2\sqrt[4]{\varepsilon}}$ [+ 1.

В программе вычисления интеграла с точностью во внутреннем цикле находят значение определенного интеграла при заданном (фиксированном) числе разбиений интервала интегрирования. Во внешнем цикле производится сравнение значений интегралов, вычисленных при числе шагов, равных n и 2n соответственно. Если требуемая точность не достигнута, то производится удвоение числа разбиений, а в качестве предыдущего значения интеграла берется текущее; процесс вычисления интеграла выполняется при новом числе разбиений.

В дальнейшем в программах при вычислении значений аргумента используется формула арифметической прогрессии $X=X_н +(i-1)*dx$, а не формула накопления суммы X=X+dx, так как в первом случае получается меньшая погрешность вычислений [5].

Пример программы вычисления определенного интеграла $\int_a^b xe^x dx$ с точностью ε методом трапеций ($(x-1)e^x$ - первообразная подынтегральной функции).

```
//Вычисление интеграла с точностью.
//Метод трапеций.
```

Оглавление

```
#include "stdafx.h"
#include <conio.h>
#include <math.h>

int _tmain(int argc, _TCHAR* argv[])
{
	double
		i1	//предыдущее значение интеграла (число разбиений равно n)
		,i2	//текущее значение интеграла (число разбиений равно 2*n)
		,x	//текущее значение аргумента
		,xn	//нижний предел интегрирования
		,xk	//верхний предел интегрирования
		,dx	//шаг изменения аргумента
		,s1	//полусумма значений функции, вычисленных на нижнем и
			//верхнем пределах интегрирования
		,eps	//точность вычисления интеграла
		,itoch; //точное значение интеграла, вычисленное на основе
			//первообразной
	int
		i	//параметр цикла
		,n	//количество разбиений интервала интегрирования
		nn;	//начальное количество разбиений
	printf("wwedite nach, kon znach, tochnost\n");
	scanf("%lf %lf %lf",&xn,&xk,&eps); //ввод исходных данных
	printf("Metod trapezij\n");
	printf("Isxodnie dannie xn=%5.2f,   xk=%5.2f",xn,xk);
	printf("    eps=%7.1e\n",eps);
	//вычисление начального количества разбиений
	nn=(xk-xn)/sqrt(eps)+1;
	n=nn;
	dx=(xk-xn)/n;
	//полусумма значений функции на нижнем и
	//верхнем пределах интегрирования
	s1=(xn*exp(xn) + xk*exp(xk))/2.0;
	//начальное значение интеграла, вычисленное
	//при начальном количестве разбиений
	i2=0;
	for (i=1;i<n;i++)
	{
```

```
            x=xn+i*dx;       // текущее значение аргумента
            //вычисление суммы значений функции в узлах интегрирования
            i2+=x*exp(x);
      }
      //значение интеграла при начальном количестве разбиений
      i2=(s1+i2)*dx;
      printf("Pri nachalnom kolichestve razbienij n=%d\n",n);
      printf("inegral raven %8.4f\n\n",i2);
      //вложенный цикл вычисления интеграла с точностью
      do
      {
            n*=2; //удвоение количества разбиений
            i1=i2; //переменной i1присваивается  текущее
          //приближенное значение интеграла
            dx=(xk-xn)/n; //шаг изменения переменной интегрирования
            //начальному значению суммы присваивается полусумма
            //значений функции на концах интервала интегрирования
            i2=s1;
            //внутренний цикл для вычисления суммы значений
            //функции при фиксированном количестве шагов
            for (i=1;i<n;i++)
            {
                  x=i*dx+xn; // текущее значение аргумента
                  //накопление суммы значений функции
                  //в узлах интегрирования
                  i2+=x*exp(x);
            }
            i2*=dx; //текущее значение интеграла
      } while (fabs(i2-i1)/3>=eps); //анализ точности вычисления

      printf("inegral s tochnostu %7.1e raven %8.4f \n",eps, i2);
      printf("Pri kolichestve razbienij = %d\n",n);
      itoch=(xk-1)*exp(xk)-(xn-1)*exp(xn);//точное значение интеграла
      printf("\nTochnoe znachenie integrala ravno %8.4f",itoch);
      getch();
      return 0;
}
```

Метод трапеций удобен для вычисления интеграла по правилу Рунге, так как при увеличении количества разбиений в два раза каждый второй узел представляет собой ранее рассматривавшийся узел, поэтому вновь значения функции в этих точках вычислять

не следует. Для этого следует сумму значений функции в этих узлах сохранять в переменной `s2`.

Модифицированная программа приведена ниже. В отличие от первого варианта программы до вложенного цикла необходимо написать цикл вычисления сумм значений функции при начальном разбиении интервала интегрирования для всех узлов, кроме первого и последнего.

```
scanf("%lf%lf%lf",&xn,&xk,&eps);
n=(xk-xn)/sqrt(eps)+1;
s1=(xn*exp(xn) + xk*exp(xk))/2.0;
s=0;
dx=(xk-xn)/n;
//Цикл вычисления суммы значений функции в узлах интегриро-
//вания при начальном разбиении интервала интегрирования
for (i=1; i<n;i++)
{
  x=xn+i*dx;
  s+=x*exp(x);
}
i2=dx*(s+s1);
s2=s;
do
 {
  i1=i2;
  n*=2;
  dx=(xk-xn)/n;
  x1=xn+dx; //Координата первого нового узла  интегрирования
  s=0;
  //Цикл вычисления суммы значений функции
  //в новых узлах интегрирования
  for (i=1;i<=n/2;i++)
  {
    x=x1+2*(i-1)*dx;
    s+=x*exp(x);
  }
```

```
        //Сумма значений функции в узлах интегрирования
        //при новом числе разбиений
        s2+=s;
        i2=(s1+s2)*dx;
        }
        while (fabs(i2-i1)/3>eps);
```

Полный текст программы приведен ниже:

```
#include "stdafx.h"
#include <conio.h>
#include <math.h>
int _tmain(int argc, _TCHAR* argv[])
{
      double i1  //предыдущее значение интеграла (число разбиений равно n)
             ,i2 //текущее значение интеграла (число разбиений равно 2*n)
             ,x  //текущее значение аргумента
             ,x1 // абсцисса первого нового узла интегрирования
             ,xn //нижний предел интегрирования
             ,xk //верхний предел интегрирования
             ,dx //шаг изменения аргумента
             ,s  //сумма значений функции, вычисленных в старых узлах
                 //интегрирования
             ,s1 //полусумма значений функции, вычисленных на нижнем и
                 //верхнем пределах интегрирования
             ,s2 //сумма значений функции, вычисленных в старых и новых узлах
                 //интегрирования
            ,eps //точность вычисления интеграла
            ,itoch; //точное значение интеграла, вычисленное на основе
                    //первообразной
      int
            i    //параметр цикла
          ,n     //количество разбиений интервала интегрирования
          ,nn;   //начальное количество разбиений
      printf("wwedite nach, kon znach, tochnost\n");
      //scanf("%lf %lf %lf",&xn,&xk,&eps);
      printf("Metod trapezij\n");
      printf("Isxodnie dannie xn=%5.2f,   xk=%5.2f",xn,xk);
      printf("    eps=%7.1e\n",eps);
      //вычисление начального количества разбиений
      n=(xk-xn)/sqrt(eps)+1;
      s1=(xn*exp(xn) + xk*exp(xk))/2.0;
      s=0;
```

```
	dx=(xk-xn)/n;
	//Цикл вычисления суммы значений функции в узлах интегриро-
	//вания при начальном разбиении интервала интегрирования
	for (i=1; i<n;i++)
	{
		x=xn+i*dx;
		s+=x*exp(x);
	}
	i2=dx*(s+s1);
	printf("\ninegral s tochnostu %7.1e raven %8.4f \n",eps, i2);
	printf("Pri nachalnom kolichestve razbienij = %d\n",n);
	s2=s;
	do
	{
		i1=i2;
		n*=2;
		dx=(xk-xn)/n;
		x1=xn+dx; //Координата первого нового узла  интегрирования
		s=0;
		//Цикл вычисления суммы значений функции
		//в новых узлах интегрирования
		for (i=1;i<=n/2;i++)
		{
			x=x1+2*(i-1)*dx;
			s+=x*exp(x);
		}
		//Сумма значений функции в узлах интегрирования
		//при новом числе разбиений
		s2+=s;
		i2=(s1+s2)*dx;
	}
	while (fabs(i2-i1)/3>eps);

	printf("\ninegral s tochnostu %7.1e raven %8.4f \n",eps, i2);
	printf("Pri poslednem kolichestve razbienij = %d\n",n);
	itoch=(xk-1)*exp(xk)-(xn-1)*exp(xn);//точное значение интеграла
	printf("\nTochnoe znachenie integrala ravno %8.4f",i2);
	getch();
	return 0;
}
```

Таким же образом можно сократить количество выполняемых операций при вычислении интеграла по методу левых или правых прямоугольников. Однако на практике при использовании метода прямоугольников значение функции вычисляют не на конце интервала (левом или правом), а в его середине. В этом случае при удвоении числа разбиений ранее вычисленные значения функции (и их сумму) использовать не удается. Для использования результатов ранее выполнявшихся вычислений при применении метода прямоугольников следует количество разбиений увеличивать в три раза и вычислять значения функции в точках с абсциссами $X_i = X_{нач} + \Delta X \cdot i/6 + \Delta X (i-1)$, $i=1,n$; $X_j = X_{нач} + 5\ \Delta X\ j/6 + \Delta X\ (j-1)$, $j=1,n$. В приведенных выражениях $X_{нач}$ – нижний предел интегрирования, ΔX – шаг интегрирования, n – количество разбиений на предыдущей итерации.

При вычислении интеграла по методу Симпсона можно непосредственно использовать суммы значений функции, вычислявшиеся на предыдущей итерации. При этом надо иметь в виду, что узлы с нечетными номерами получают на очередной итерации четные индексы и сумма значений функции для этих узлов должна браться с коэффициентом 2 (на предыдущей итерации с коэффициентом 4). Узлы с четными номерами и при новой итерации останутся четными, поэтому сумма значений в этих узлах берется с тем же коэффициентом 2.

5.2. Задания для самостоятельной работы

В таблице приведены подынтегральные функции. Вычислить значение интеграла каждым из трех методов (прямоугольников, трапеций, парабол) с заданной точностью. Пределы интегрирования задавать из интервала, заданного в таблице. Сравнить скорость сходимости используемых методов. Обеспечить в программе ввод точности, пределов интегрирования, коэффициентов, присутствующих в функциях, вывод с поясняющим текстом вычисленных значений интеграла каждым методом, количество разбиений при использовании каждого метода.

Номер варианта	Подынтегральная функция	Интервал для выбора пределов интегрирования
1	$X^3 e^{X^2}$	1 - 5
2	Xe^{X^2}	1 - 7
3	$X^2\ e^{aX}$	1 - 6
4	$e^{aX}\ \sin pX$	1 - 5

5	Xe^{aX}	1 - 7
6	$X^3\ (tgX + ctgX)$	2 - 3
7	$tg^3\ X$	0,9 - 1,5
8	$tg^2\ X$	1 - 1,52
9	$e^{tgX+ctgX}$	1,7 - 3
10	ctg^2X	0,1 - 0,8
11	ctg^3X	0,1 - 0,8
12	$sin(X)X^4$	1 - 2
13	$cos(X)X^4$	1 - 3
14	$tg(X)X^2$	1 - 1,52
15	$e^X + e^{-X}$	1 - 4
16	$e^X - e^{-X}$	-4 - 4
17	$e^{\sqrt{x}}$	3 - 16
18	$X^{2/3}e^{aX}$	2 - 5
19	$X^{2\sqrt{X^2+X}}$	2 - 10
20	$(tgX+lnX)X^{5/2}$	1,1 - 1,52
21	$(ctgX+X)X^{-13/4}$	0,1 - 1
22	e^{tgX}	0,9 - 1,5
23	X^X	3 - 7
24	$X^{\frac{8}{3}\sqrt{X^3+X}}$	2 - 10
25	$X^{13/2}sinX$	1,1 - 2,5
26	$X^{\frac{7}{2}}\ log_2(X^3 - X)$	2 - 8
27	$ctgX$	0,1 - 0,7
28	$X^{2\sqrt{X}}$	3 - 6
29	$X^4 - X^3 + X^2$	1 - 8

30	*tgX*	1 - 1,52

5.3.Вычисление наибольшего (наименьшего) значения функции с заданной точностью на заданном интервале

Необходимость организации вложенного цикла возникает также и при решении задачи нахождения наибольшего (наименьшего) значения функции на заданном интервале с заданной точностью. Рассмотрим сначала задачу определения с заданной точностью аргумента, при котором функция достигает своего экстремального значения.

Данная задача может быть сформулирована следующим образом. Задана некоторая функция f(x) и некоторый интервал [a,b]. Известно, что на заданном интервале функция имеет один экстремум, известен и вид экстремума (максимум или минимум). Требуется с заданной точностьюε найти значение аргумента, при котором достигается экстр емум функции.

Решить поставленную задачу можно, используя прием программирования - вычисление максимума или минимума. При этом шаг изменения аргумента следует задать равным требуемой точностиε. Однако такой подход может потребовать большого количества вычислений. Сократить количество выполняемых операций можно за счет использования следующего алгоритма. Сначала вычисляются значения функции при “грубом” значении шага изменения аргумента. При этом очередное значение функции сравнивается с ранее вычисленным максимальным значением (допустим, что определяется максимум функции) и если текущее значение превышает максимальное, т.е. $f(x_i)>f_{max}$, то $f_{max} := f(x_i)$, и производится вычисление значения функции в следующей точке. Если же на очередном шаге выполнится условие $f(x_i)<f_{max}$, то это будет означать, что максимум функции уже пройден, т.е. он находится на интервале (x_i-2h, x_i), где h – шаг изменения аргумента. В этом случае шаг изменения аргумента уменьшается (обычно в два раза) и на вновь полученном интервале вычисляются значения функции и ищется максимум при новом шаге изменения аргумента. Процесс продолжается до тех пор, пока h не станет меньше или равным ε.

Таким образом, во внутреннем цикле вычисляются значения функции и ищется максимум на заданном интервале изменения аргумента при заданном шаге изменения аргумента. Во внешнем цикле задается новый интервал поиска максимума и новый шаг изменения аргумента. Фрагмент программы вычисления максимума функции $y=2x^3+10x^2+6x-20$ в интервале [a,b] имеет вид:

```
//Вычисление максимума функции с использованием
//итерационного (вложенного) цикла
n=0; //Количество вычислений значения функции
scanf("%lf%lf%lf%lf",&a,&eps,&h);
xn=a;
while (h>eps)
{
    //Установка начального значения для максимума
  ymax=((2*xn+10)*xn+6)*xn-20;
  xmax=xn;
  //Цикл определения максимума функции
  //на очередном интервале изменения аргумента
  do
  {
    //Вычисление текущего значения аргумента
    x=xn+i*h;
    //Вычисление значения функции в очередной точке
    y=((2*x+10)*x+6)*x-20;
    n++;
    //Определение нового максимума
    //и соответствующего значения аргумента
    if (y>ymax)
    {
      ymax=y;
      xmax=x;
     }
   }  while (y>=ymax);
  //Определение левого конца нового интервала
  //вычисления максимума
  xn=xmax-h;
  //Уменьшение шага изменения аргумента в два раза
  h/=2;
}
```

Полный текст программы выглядит следующим образом:

```
#include "stdafx.h"
#include <conio.h>
#include <math.h>

int _tmain(int argc, _TCHAR* argv[])
{
	//Переменные программы представляют:
	double a=-5 //левая граница интервала поиска максимума,
		,eps=1e-8 //требуемую точность,
		,y	//текущее значение функции,
		,x	//текущее значение аргумента,
		,xn	//текущее значение аргумента начала поиска максимума
		,h=0.1		//шаг приращения аргумента,
		,xmax		//точка максимума функции,
		,ymax;		//максимум функции
	int
		i	//параметр цикла
		,n;	//количество вычислений значения функции.
	n=0; //устновка начального значения количество вычислений значения
	     //функции
	scanf("%lf%lf%lf",&a,&eps,&h);//ввод исходных данных
	xn=a;
	while (h>eps) //внешний цикл поиска экстремума (пока не достигнута
		      //требуемая точность)
	{
		i=0; //установка начального значение параметра
		//Установка начального значения для максимума
		ymax=((2*xn+10)*xn+6)*xn-20;
		xmax=xn;
		//Цикл определения максимума функции
		//на очередном интервале изменения аргумента
		do
		{
			i++;
			//Вычисление текущего значения аргумента
			x=xn+i*h;
			//Вычисление значения функции в очередной точке
			y=((2*x+10)*x+6)*x-20;
			n++;
			//Определение нового максимума
			//и соответствующего значения аргумента
			if (y>ymax)
```

```
                {
                        ymax=y;
                        xmax=x;
                }
        }
        while (y>=ymax);
        //Определение левого конца нового интервала
        //вычисления максимума
        xn=xmax-h;
        //Уменьшение шага изменения аргумента в два раза
        h/=2;
    }
    printf("\nymax = %lf,  xmax = %lf,   h = %e  n = %d\n"
                ,ymax, xmax, h, n);
    getch();
    return 0;
}
```

Поиск минимума функции можно свести к поиску максимума, если функцию f(x) заменить функцией -f(x).

Рассматриваемую задачу можно обобщить на случай, когда функция может не иметь внутри заданного интервала экстремума, а изменяется внутри него монотонно, причем заранее характер поведения функции не известен. В этом случае следует говорить о нахождении наибольшего (наименьшего) значения функции на заданном интервале. Наибольшее (наименьшее) значение функции достигается либо в критической точке (где первая производная равна нулю), либо на конце интервала.

В этом случае в программу необходимо добавить проверку аргумента на принадлежность заданному интервалу. В случае отсутствия такой проверки аргумент может принимать значения, не принадлежащие заданному интервалу (если наибольшее значение достигается на левом конце интервала, то возможен выход аргумента за левую границу, если же оно достигается на правом конце интервала, то возможен выход аргумента за правый конец). Тогда добавляют еще одно условие выхода из внутреннего цикла (см. текст последней программы) и проводят проверку нового значения для начального значения аргумента:

```
    . . . .
    while(y>=ymax && x<=b);
```

```
xn= xmax-h;//Определение левой границы нового интервала
            //вычисления наибольшего значения
if (xn<a) xn=a; //Проверка выхода нового значения аргумента
          //за пределы заданного интервала
. . . .
```

Фрагмент программы с внесенными изменениями будет иметь следующий вид:

```
while (h>eps) //внешний цикл поиска экстремума (пока не достигнута
              //требуемая точность)
{
      i=0; //установка начального значение параметра
      //Установка начального значения для максимума
      ymax=((2*xn+10)*xn+6)*xn-20;
      xmax=xn;
      //Цикл определения максимума функции
            //на очередном интервале изменения аргумента
      //Цикл определения максимума функции
      //на очередном интервале изменения аргумента
      do
      {
            i++;
            //Вычисление текущего значения аргумента
            x=xn+i*h;
            //Вычисление значения функции в очередной точке
            y=((2*x+10)*x+6)*x-20;
            n++;
            //Определение нового максимума
            //и соответствующего значения аргумента
            if (y>ymax)
            {
                  ymax=y;
                  xmax=x;
            }
      } while(y>=ymax && x<=b);
      xn=xmax-h;  //Определение левой границы нового интервала
                  //вычисления наибольшего значения
      if (xn<a) xn=a;   //Проверка выхода нового значения аргумента
                        //за пределы заданного интервала
      //Уменьшение шага изменения аргумента в два раза
      h/=2;
 }
```

Если известны аналитические выражения для первой и второй производных, то можно найти значение аргумента, при котором первая производная обращается в ноль (решить уравнение, например, методом деления отрезка пополам) и определить знак второй производной в найденной точке. Знак второй производной позволит определить вид экстремума (максимум или минимум) или его отсутствие на заданном интервале. При отсутствии экстремума пользователю программы предлагается ответить на вопрос о нахождении наибольшего или наименьшего значения функции.

Пример программы вычисления наибольшего (наименьшего) значения функции приведен ниже. В программе реализованы три рассмотренных подхода к определению наибольшего (наименьшего) значения функции на заданном интервале. Программа позволяет также оценить трудоемкость последних двух способов на основе сравнения количества вычислений значений функции.

```
#include "stdafx.h"
#include <conio.h>
#include <math.h>

/*
Вычисление наибольшего(наименьшего) значения функции
y=2x3 +10x2 +6x-20 в интервале [a,b] с заданной точностью eps
с начальным шагом изменения аргумента h.
y'=6x2 +20x+6 - первая производная функции,
y''=12x+20 - вторая производная
задаваемый интервал должен содержать только один экстремум или
на этом интервале функция должна изменяться монотонно
*/
int _tmain(int argc, _TCHAR* argv[])

{
 double
  x  // значение аргумента
,a   // левая граница интервала
,b   // правая граница интервала
,aa  // вспомогательные переменные, дублирующие
,bb  // значения границ интервала
,h   // шаг изменения аргумента
,y   // значение функции
,ymax // экстремальное значение функции
,ya   // значение функции на левой границе интервала
```

```
,yb   // значение функции на правой границе интервала
,ysr  // значение функции в середине интервала
,eps  // точность вычисления экстремума
,xmax // экстремальное значение
,dx   // шаг изменения аргумента
,xn   // левая граница интервала
,xsr; // значение аргумента в середине интервала
 int i //параметр цикла
   ,n1 //количество вычислений значений функции при использовании
       //табулирования функции
   ,n2 //количество вычислений значений функции при использовании
       //итерационного цикла
    ,k //коэффициент, учитывающий характер экстремума (максимум 1, минимум
       // -1)
    ,mon;//признак монотонности функции

  printf("Введите a,b,eps,h\n");
  scanf("%lf%lf%lf%lf",&a,&b,&eps,&h);
  mon=0;  //Признак монотонности:0 - немонотонная;1-монотонная
  //Определение точки на заданном интервале, в которой первая
  //производная функции обращается в ноль (уточнение корня
  //уравнения методом деления отрезка пополам)
  ya= 6*a*a+20*a+6;
  if (ya==0)
      xmax=a;
  else
  {
    yb=6*b*b+20*b+6;
    if (yb==0)
      xmax=b;
    else if (ya*yb>0)
   {
   printf("Функция на заданном интервале изменяется монотонно");
   printf("\nДля нахождения наибольшего значения введите 1");
   printf("\nДля нахождения наименьшего значения введите -1");
   scanf("%d",&k);
   mon=1;
   }
    else   // уточнение корня первой производной методом половинного деления
    {
      aa=a;
      bb=b;
```

```
      xsr=(aa+bb)/2;
      ysr=6*xsr*xsr+20*xsr+6;
      while (fabs(bb-aa)>eps && ysr*ya!=0)
      {
        if (ya*ysr>0)  aa=xsr; else bb=xsr;
        xsr=(aa+bb)/2;
        ysr=6*xsr*xsr+20*xsr+6;
      }
      xmax=xsr;
    }
  }
  //Определение знака второй производной в критической точке
  //с целью определения вида экстремума
  //(вторая производная отрицательна - максимум,
  //вторая производная положительна - минимум)
  if (mon==0)
  {
    if (12*xmax+20<0)
      k=1;
    else
      k=-1;//Поиск минимума сводится к поиску максимума
           //функции -f(x)
    //Вычисление значения функции в критической точке
    ymax=k*(((xmax+5)*xmax+3)*2*xmax-20);
    switch (k)
   {    case  1: printf("\nymax=%8.4f xmax=%8.4f",k*ymax,xmax);break;
   case -1: printf("\nymin=%8.4f xmin=%8.4f",k*ymax,xmax); break;
    }
    getch();
  }
//Поиск максимума функции с использованием табулирования
//значений функции при шаге изменения аргумента, равном
//требуемой точности
  n1=(b-a)/eps+1;
//Установка начального значения для максимума
  ymax=((2*a+10)*a+6)*a-20;
  xmax=a;
  dx=eps;
  for (i=1;i<n1;i++)
  {
    x=a+i*dx;
    y=k*(((2*x+10)*x+6)*x-20);
```

```
    if (y>ymax)
    {
      ymax=y;
      xmax=x;
    }
  }
  switch (k)
   {    case  1: printf("\nymax=%8.4f xmax=%8.4f",k*ymax,xmax);break;
        case -1: printf("\nymin=%8.4f xmin=%8.4f",k*ymax,xmax); break;
    }
   printf("\nn1=%3d",n1);
    getch();
  //Вычисление наибольшего значения функции с использованием
  //итерационного (вложенного) цикла
  n2=0; //Количество вычислений значения функции
  xn=a;
  while (h>eps)
  {
    //Установка начального значения для наибольшего значения
    ymax=k*{((2*xn+10)*xn+6)*xn-20);
    xmax=xn;
    i=0;
    //Цикл определения наибольшего значения функции
    //на очередном интервале изменения аргумента
    do
      {
      i++;
      x=xn+i*h;  //Вычисление текущего значения аргумента
      //Вычисление значения функции в очередной точке
      y=k*(((2*x+10)*x+6)*x-20);
      n2++;
      if (y>ymax)
      //Определение нового наибольшего значения и
      //соответствующе го значения аргумента
        {
          ymax=y;
          xmax=x;
        }
      }
    while (y>=ymax && x<b);
    xn=xmax-h; //Определение левой границы нового интервала
                //вычисления наибольшего значения
```

```
    //Проверка выхода аргумента за пределы интервала
    if (xn<a)
       xn=a;
    h/=2; //Уменьшение шага изменения аргумента в два раза
   }
   switch (k)
   {    case  1: printf("\nymax=%8.4f xmax=%8.4f",k*ymax,xmax);break;
        case -1: printf("\nymin=%8.4f xmin=%8.4f",k*ymax,xmax); break;
    }
   printf("\nn2=%3d",n2);
    getch();
      return 0;
}
```

5.4. Задания для самостоятельной работы

Составить программу для вычисления экстремума функции на заданном интервале с заданной точностью.

1) Найти аналитическое выражение для первой производной заданной в таблице функции. Одним из известных методов уточнения корня уравнения найти значение аргумента на заданном интервале, при котором первая производная обращается в ноль. Вычислить значение функции в полученной точке. С помощью второй производной определить вид экстремума (максимум или минимум)

2) Найти с помощью итерационного алгоритма экстремум той же функции и значение аргумента, при котором он достигается.

Обеспечить ввод в программе значения точности, значений, определяющих интервал поиска экстремума, начальный шаг изменения аргумента. Для обоих вариантов поиска экстремума вывести значение аргумента, при котором достигается экстремум функции, и значение функции в полученной точке.

Номер варианта	Функция	Интервал поиска экстремума
1	X^2+1/X^2	[0,2; 2]
2	$X^3 - X^2 - 6X$	[-2;0]
3	$\cos(2x)/2 - \sin(x) + 2$	$[\pi/3;2\pi/3]$
4	$X+2\sqrt{10-X}$	[0;10]

5	$3X^4 -8X^3 +6 X^2 +2$	[0,1;2]
6	cos(2x)-2cos(x)	[-π/6;π/3]
7	$5^x -4X$	[0;2]
8	$X^3 - X^2 – 6X$	[0; 3]
9	$X^4 -8X^2 -9$	[-1;1]
10	$(4^x +4^{-x})/\ln(4)$	[-1;2]
11	$2X^2 +7\|X-4\| +5$	[-1;2]
12	$X^2\sqrt{5-X}$	[1;5]
13	$X^2 +\|X-4\|$	[-3;3]
14	$1/(X^2 + X +2)$	[-2;2]
15	$2X^3 +15X^2 +36X -30$	[-4;-2,4]
16	$X^4 -6X^3 +8$	[1;6]
17	$\sqrt{X(8-X)}$	[0,6]
18	$X^3 -5X^2 +3X$	[1,5;5]
19	$2X^3 -3X^2$	[-2;0,5]
20	$\log_2(X^2 -2 X +5)$	[0,2;2]
21	$6X - X^2 -6$	[0;6]
22	$\log_3(2 X +3- X^2)$	[0;2]
23	$X^4 -8X^2 -9$	[-4;-0,5]
24	$3X^4 -8X^3 +6 X^2 +2$	[-2;0,1]
25	cos(2x)/2 – sin(x) +2	[-π/3;9π/10]
26	$2X^3 +15X^2 +36X -30$	[-2,4;0]
27	$X^3 -5X^2 +3X$	[-1;1]
28	$2X^3 -3X^2$	[0,5;3]
29	$X^4 -8X^2 -9$	[0,2;3,5]
30	$\log_2(3- 4X - 4X^2)$	[-1,2;0]
31	$0,5^{x^2 -2x}$	[0;2]

32	8^x-6*4^x-3*2^x	[0;3]
33	$\sqrt{5-2x+3x^2-x^3/3}$	[0;2,5]
34	$\sqrt{5-2x+3x^2-x^3/3}$	[2,5;4]

5.5.Обработка матриц

Матрицы представляют собой один из наиболее удобных математических объектов, обработка которых ведет к необходимости программирования вложенных циклов. При программной реализации обработки матриц будем использовать двумерные массивы (элементы которых имеют два индекса), считая, что элементы X_{ij} матрицы X, i=1, 2,...,m, j=1, 2,…,n размещаются в ячейках соответствующего массива с такими же индексами. Это позволит сделать текст программы удобным для понимания процесса обработки и уменьшить, в некоторых случаях, объём вычислений. Соответственно первый индекс массива будем называть номером строки, а второй – номером столбца.

Предполагая соответствие элементов матрицы элементам массива, им дают, как правило, одинаковые имена; иногда в пояснениях к программе вместо термина *массив* будем использовать термин *матрица*, как бы отождествлять объекты предметной области с объектами программы для лучшего понимания процессов обработки.

При постановке задач на обработку матриц с целью обеспечения применимости программ для обработки матриц, например, матрицы X, размеры которых не должны превосходить заданных значений, например, m≤12 и n≤14, будем использовать сокращенное обозначение, например, X(m,n), m≤12, n≤14. Указанные в таком обозначении максимальные количества строк и столбцов матрицы будут использоваться в *объявлении* соответствующего *двумерного массива*, а сами эти максимальные значения желательно объявить в виде именованных констант. Если также учесть, что двумерный массив можно рассматривать и объявлять как одномерный массив с базовым типом массив, то объявление типа массива для хранения матриц с размерами, не превышающими 12x14, будет следующим.

```
const int  mmax=12, nmax=14;
typedef float  tv[nmax]; // определение имени tv нового типа -
      //одномерного массива действительных элементов
typedef   tv   tmatr[mmax]; //определение имени tmatr – нового типа
// двумерного массива (одномерного массива, состоящего из одномерных
```

```
// массивов
```

(при объявлении типа двумерного массива необходимо задавать количество элементов по каждому измерению, т.е. количество строк и столбцов, и тип самих элементов массива).

Задание в разделе констант максимальных значений для количества строк и столбцов улучшает стиль программирования, так как в случае необходимости изменения количества строк или столбцов программисту не придется внимательно изучать весь текст программы, а достаточно будет изменить значения соответствующих констант.

```
float x[mmax][nmax];
```

Объявление переменной-двумерного массива может быть объявлено следующими способами

```
1) tmatr b;
2) typedef float tmas2[mmax][nmax];
      tmas2 b1;
3) float b2[mmax][nmax];
```

При работе с матрицами надо помнить, что первый индекс соответствует номеру строки, а второй индекс – номеру столбца.

Ввод-вывод матриц можно осуществить только с использованием вложенного цикла. Обычно принято вводить матрицу в виде матрицы, т.е. по строкам как это принято записывать на бумаге. В этом случае во внутреннем цикле должен обеспечиваться ввод (вывод) одной строки матрицы. Внешний цикл будет задавать ввод (вывод) всех строк матрицы и переход в начало новой строки после ввода (вывода) очередной строки.

Фрагмент программы, обеспечивающей ввод и вывод матрицы X(m,n), m≤12, n≤14, имеет следующий вид:

```
int m,n;
printf("wwedite rasmeri m i n");
scanf("%d%d",&m,&n);
printf("\n wwedite matrizu \n");
for (int i=0;i<m;i++)  //Цикл ввода всех строк матрицы
  for (int j=0;j<n;j++)//Цикл ввода очередной строки матрицы
     scanf("%f",&b[i][j]);
printf("\n matriza\n");
for (int i=0;i<m;i++)  //Цикл вывода всех строк матрицы
{
  for (int j=0;j<n;j++)//Цикл вывода очередной строки матрицы
     printf("%6.1f ",b[i][j]);
    printf("\n");  //Переход в начало новой строки файла вывода
}
```

Порядок расположения циклов при вводе-выводе имеет существенное значение. Если поменять местами заголовки циклов, то вводимые значения будут присваиваться элементам столбца, а не элементам строки. В этом случае будет выведена фактически транспонированная матрица

Также следует соблюдать аккуратность при использовании приемов программирования. Например, при вычислении сумм и произведений надо следить за тем, в каком месте программы присваиваются начальные значения соответствующим переменным.

Приведенный ниже фрагмент программы позволяет вычислить сумму и произведение всех элементов матрицы:

```
s=0; pr=1;
for (int i=0;i<m;i++)
      for (int j=0;j<n;j++)
      {
            s+=a[i][j];
            pr*=a[i][j];
      }
printf("\ns=%6.1f, pr=%6.1f",s,pr);
```

При изменении места расположения операторов присваивания `s=0; pr=1;` смысл вычислений меняется:

```
for(int i=0;i<m;i++)
{
      s=0; pr=1;
      for (int j=0;j<n;j++)
      {
            s+=a[i][j];
            pr*=a[i][j];
            printf("\ns=%6.1f, pr=%6.1f",s,pr);
      }
}
```

Приведенный фрагмент программы позволит вычислить сумму и произведение элементов каждой строки матрицы, а не всех элементов матрицы.

При вычислении максимального и минимального элементов матрицы также надо правильно устанавливать начальные значения вычисляемых переменных, а также правильно задавать пределы изменения параметров циклов:

```
while(y>=ymax && x<=b);
      xn=xmax-h;  //Определение левой
      amax=a[0][0]; imax=0; jmax=0;
      amin=a[0][0]; imin=0; jmin=0;
      for (int i=0;i<m;i++)
            for (int j=0;j<n;j++)
                  if (a[i][j]>amax)
                  {
                        amax=a[i][j];
                        imax=i;
                        jmax=J;
                  }
                  else if (a[i][j]<amin)
                        {
                              amin=a[I][j];
                              imin=i;
                              jmin=j;
                        }
      printf("amax=%8.2f imax=%2d jmax%2d ",amax,imax,jmax);
      printf("amin=%8.2f imin=%2d jmin%2d ",amin,imin,jmin);
```

В приведенном фрагменте программы нельзя опускать операторы присваивания начальных значений индексам минимального и максимального элементов (`imax=0; jmax=0; imin=0;  jmin=0;`). В случае отсутствия указанных операторов индексы искомых элементов будут не определены в том случае, если первый элемент `a[0][0]` является максимальным (минимальным).

При поиске минимального (максимального) элемента матрицы нельзя устанавливать начальное значение параметра цикла, равное одному, как это можно сделать при обработке одномерного массива. Следующий фрагмент программы будет приводить к неверному результату, так как будут исключены из рассмотрения все элементы нулевой строки, за исключением первого, значение которого выбирается в качестве начального значения для максимума и минимума.

```
max=a[0][0]; imax=0; jmax=0;
min=a[0][0]; imin=0; jmin=0;
```

```
for (int i=1;i<m;i++)
  for (int j=0; j<n; j++)
  ....................
```

Рассмотрим примеры обработки матриц.

5.6.Примеры выполнения задания на обработку матриц

1. В матрице A(m,n), m≤12, n≤10, поменять местами строки с наибольшей и наименьшей суммами элементов.

Для решения поставленной задачи необходимо вычислить суммы элементов каждой строки матрицы, однако не требуется запоминать все вычисленные суммы. Поэтому после вычисления суммы элементов очередной строки матрицы следует сравнить вычисленное значение с текущими значениями максимальной и минимальной сумм и изменить, в случае необходимости, текущие значения максимума и минимума на только что вычисленное, а также запомнить номер строки, для которой эта сумма минимальна или максимальна.

После нахождения номеров строк, подлежащих обмену местами выполняется собственно обмен. Поскольку строка матрицы представляет собой одномерный массив (первый индекс имеет фиксированное значение), то эта операция выполняется с использованием обычного (не вложенного цикла), при этом обмен местами ведется с использованием промежуточной простой переменной b.

```
#include "stdafx.h"
#include "conio.h"
int _tmain(int argc, _TCHAR* argv[])
{
const int  mm=12, nn=10; //задание максимального количества строк и
                         //столбцов матрицы
  float
      a[mm][nn]  //матрица
     ,s  //  сумма элементов текущей строки
     ,smax  //  максимальная сумма
     ,smin  //  минимальная сумма
     ,b; //  вспомогательная переменная для обмена
  int
      imax   //индекс строки с максимальной суммой элементов
     ,imin  //индекс строки с минимальной суммой элементов
```

```
      ,i,j   //параметры циклов
      ,m,n;  //фактическое количество строк и столбцов матрицы

  printf("wwedite kol-wo strok i stolbzow\n");
  scanf("%d%d",&m,&n);
  printf("\nwwedite matrizu\n");
  for (i=0;i<m;i++)
    for (j=0; j<n;j++)
      scanf("%f",&a[i][j]);
  printf("\nisxodnaja matriza\n");
  for (i=0; i<m;i++)
  {
    for (j=0; j<n;j++)
      printf("%6.1f",a[i][j]);
    printf("\n");
  }
  //Задание начальных значений
  smax=-1e30; //Очень маленькое число
  smin=1e30;  //Очень большое число
  //Строго говоря, самое маленькое и самое большое
  //значения для используемого типа данных
/*
//Другой вариант - нахождение суммы элементов первой строки и
//задание этого значения для максимума и минимума:
 s=0;
 for (J=0; j<n;j++)
   s+=a[0][j];
 smax=s; imax=0;
 smin=s; imin=0;
//Далее строки можно рассматривать, начиная со второй
*/
  //Цикл поиска строк с максимальной и минимальной
  //суммами элементов
  for (i=0;i<m;i++)
  {
    s=0;  //Задание начального значение суммы
    //Цикл вычисления суммы элементов очередной строки
    for (j=0;j<n;j++)
      s+=a[i][j];
    if (s>smax) //нахождение строки с максимальной суммой
      {
      smax=s;
```

```
            imax=i;
            }
          if (s<smin) //нахождение строки с минимальной суммой
            {
            smin=s;
            imin=i;
            }
        }
        for (j=0;j<n;j++)  //Цикл обмена местами найденных строк
        {
          b=a[imin][j];
          a[imin][j]=a[imax][j];
          a[imax][j]=b;
        }
        printf("\npoluchennaja matriza\n");
        for (i=0; i<m;i++)
        {
          for (j=0; j<n;j++)
            printf("%6.1f",a[i][j]);
          printf("\n");
        }
         getch();
         return 0;
      }
```

2. Удалить из матрицы B(m,n), m≤10, n≤8 строки, содержащие отрицательные элементы. Можно предложить два варианта решения этой задачи.

Если понимать условие буквально, то в случае наличия отрицательных элементов в очередной строке матрицы, следует именно удалить эту строку. Выполняется эта операция путем переписи всех строк, расположенных после удаляемой, на одну строку в сторону меньших номеров строк и уменьшения общего количества строк матрицы на единицу. При этом в ходе анализа надо учитывать, что после удаления очередной строки на ее место перепишется новая строка, которая еще не анализировалась, поэтому номер строки изменять не следует. Если же текущая строка не удалялась, для перехода к следующей строке текущий номер следует увеличить на единицу.

Внутренний цикл, в котором определяется наличие или отсутствие отрицательных элементов в очередной строке матрицы, является циклом с заранее неизвестным числом повторений. Он должен выполняться, пока не обнаружится наличие отрицательного элемента и пока не проверены все элементы строки.

Согласно второму варианту формально производится не удаление строк, содержащих отрицательные элементы, а оставление в матрице строк, не содержащих отрицательных элементов.

В представленной ниже программе использован метод, в котором очередная i-я строка, не содержащая отрицательных элементов, переписывается в начало матрицы на место k-ой строки, где k - количество строк без отрицательных элементов на данный момент времени минус единица (учитывается, что начальный индекс имеет нулевое значение), включая найденную, если i>k, иначе - остаётся на месте.

Программа имеет следующий вид.

```
#include "stdafx.h"
#include "conio.h"
int _tmain(int argc, _TCHAR* argv[])
{
  const int  mm=10, nn=8; //задание максимального количества строк и
                          //столбцов матрицы

  float b[mm][nn];   // матрица
  int i,j,  //параметры циклов
      m,n,  //количество строк и столбцов матрицы
        k;  //количество строк без отрицательных элементов минус 1
  bool pr;  // признак отсутствия отрицательных элементов в строке
  printf("wwedite kol-wo strok i stolbzow\n");
  scanf("%d%d",&m,&n);
  printf("\nwwedite matrizu\n");
  for (i=0;i<m;i++)          //ввод элементов матрицы
    for (j=0; j<n;j++)
      scanf("%f",&b[i][j]);
  printf("\nisxodnaja matriza\n");
  for (i=0; i<m;i++)
  {
    for (j=0; j<n;j++)
      printf("%6.1f",b[i][j]);
    printf("\n");
  }
  k=-1;      //установка начального значения
  for (i=0;i<m;i++)
  {
    pr=true; //Отрицательных элементов нет
    j=0;
    while (pr && j<n) //цикл проверки наличия отрицательных элементов в
```

```
                        //строке матрицы
        if (b[i][j]<0)
          pr=false;
        else
         j++;
      if (pr) //перепись строки без отрицательных элементов в начало
              //матрицы
      {
          k++;
          if (i>k)
          for (j=0; j<n;j++)
            b[k][j]=b[i][j];
      }
    }
    if (k>=0)
    {
      printf("\nPoluchennaja matriza\n");
      for (i=0; i<=k;i++)
        {
        for (j=0; j<n;j++)
          printf("%6.1f ",b[i][j]);
        printf("\n");
        }
    }
    else
      printf("\n Poluchennaja matriza pustaja");
     getch();
     return 0;
   }
```

3. В инженерной деятельности часто приходится выполнять операцию умножения матриц. Напомним, что очередной элемент результирующей матрицы C вычисляется согласно следующей формуле:

$$C_{ij} = \sum_{k=1}^{n} A_{ik} B_{kj} ,$$

где A,B умножаемые матрицы,

i, j – индексы очередного элемента матрицы C,

n – количество столбцов первой матрицы (строк второй матрицы).

Перемножение матриц возможно только в том случае, если количество столбцов первой матрицы равно количеству строк второй матрицы. Умножение матриц приводит к необходимости программирования тройного цикла. Это связано с тем, что один элемент,

представляющий собой сумму попарных произведений соответствующих элементов исходных матриц, вычисляется в цикле, а вычисление всех элементов матрицы требует организации еще вложенного цикла. Следующая программа демонстрирует решение рассматриваемой задачи для матриц A(m,n), B(n,n2), m≤10, n≤8, n2≤12.

```
#include "stdafx.h"
#include "conio.h"
int _tmain(int argc, _TCHAR* argv[])
{
  const int  mm=10, nn=8, nn2=12; //задание максимального количества
                                  //строк и столбцов матриц
  float a[mm][nn]   //первая матрица
       ,b[nn][nn2]  //вторая матрица
       ,c[mm][nn2]  //третья матрица - произведение
       ,s;          //элемент третьей матрицы
  int i,j           //параметры циклов
      ,m1,n1        //количество строк и столбцов первой матрицы
       ,m2,n2       //количество строк и столбцов второй матрицы
           ,k;      //параметр цикла
  printf("wweditr kol-wo strok i stolbzow matrizi 1\n");
  scanf("%d%d",&m1,&n1);
  printf("\nwwedite matrizu\n");
  for (i=0;i<m1;i++)
    for (j=0; j<n1;j++)
      scanf("%f",&a[i][j]);
  printf("wweditr kol-wo strok i stolbzow matrizi 2\n");
  scanf("%d%d",&m2,&n2);
  printf("\nwwedite matrizu\n");
  for (i=0;i<m2;i++)
    for (j=0; j<n2;j++)
      scanf("%f",&b[i][j]);
  printf("\nisxodnaja matriza 1\n");
  for (i=0; i<m1;i++)
  {
    for (j=0; j<n1;j++)
      printf("%6.1f",a[i][j]);
    printf("\n");
  }
  printf("\nisxodnaja matriza 2\n");
  for (i=0; i<m2;i++)
  {
```

```
    for (j=0; j<n2;j++)
      printf("%6.1f",b[i][j]);
    printf("\n");
  }
  if (n1!=m2)  //проверка возможности умножения матриц
    printf("Umnochenie newosmoshno");
  else
  {
    for (i=0; i<m1;i++)     //цикл умножения матриц
      for (j=0; j<n2; j++)
        {
        s=0;
        for (k=0; k<n1;k++)
          s+=a[i][k]*b[k][j];
        c[i][j]=s;
        }
  printf("\npoluchennaja matriza \n");
  for (i=0; i<m1;i++)
  {
    for (j=0; j<n2;j++)
      printf("%6.1f",c[i][j]);
    printf("\n");
  }
  }
   getch();
    return 0;
}
```

Наряду с прямоугольными матрицами часто приходится иметь дело и с квадратными матрицами, в которых количество строк равно количеству столбцов. При обработке квадратных матриц часто приходится обрабатывать элементы, расположенные в определенном месте матрицы (на главной диагонали, на побочной, под главной диагональю и т.д.). В этом случае, чтобы избежать ненужных проверок индексов элементов матрицы, целесообразно знать закон изменения индексов для элементов, расположенных в определенном месте матрицы. Для матрицы A, имеющей n строк и столбцов, можно записать следующее.

- Диагональные элементы имеют одинаковые индексы:

a[i,i], i=0,n-1 .

- Элементы, стоящие над главной диагональю (образующие верхнюю треугольную матрицу), не включая диагональ:

a[i,j] , i=0,n-2; j=i+1,n-1.

- Элементы, стоящие над главной диагональю, включая диагональ:

a[i,j] , i=0,n-1; j=i,n-1.

- Элементы, стоящие под главной диагональю (образующие нижнюю треугольную матрицу), не включая диагональ:

a[i,j] , i=1,n-1; j=0,i-1.

- Элементы, стоящие под главной диагональю, включая диагональ:

a[i,j] , i=1,n-1; j=0,i.

- Элементы побочной диагонали:

a[i,n-i-1], i=0,n-1.

- Элементы, стоящие над побочной диагональю, не включая диагональ:

a[i,j] , i=0,n-2; j=0,n-i-2.

- Элементы, стоящие над побочной диагональю, включая диагональ:

a[i,j] , i=0,n-1; j=0,n-i-1.

- Элементы, стоящие под побочной диагональю, не включая диагональ:

a[i,j] , i=1,n-1; j=n-i,n-1.

- Элементы, стоящие под побочной диагональю, включая диагональ:

a[i,j] , i=0,n-1; j=n-i-1,n-1.

4. В квадратной матрице F(k,k), k≤9, определить, что больше: модуль минимального отрицательного элемента, стоящего над побочной диагональю, или максимальный положительный элемент, стоящий под побочной диагональю. Ниже представлена программа.

```
#include "stdafx.h"
#include "conio.h"
#include "math.h"
int _tmain(int argc, _TCHAR* argv[])
{
 const int mm=9;  //задание максимального количества строк
 float f[mm][mm], //квадратная матрица
            min,  //минимальный отрицательный элемент
            max;  //максимальный положительный элемент
 int i,j,         //параметры циклов
       m;         //количество строк матрицы
```

```
bool pol,otr;    //признаки наличия положительных и отрицательных
                 //элементов
printf("wweditr kol-wo strok \n");
 scanf("%d",&m);
 printf("\nwwedite matrizu\n");
 for (i=0;i<m;i++)
   for (j=0; j<m;j++)
     scanf("%f",&f[i][j]);
 printf("\nisxodnaja matriza\n");
 for (i=0; i<m;i++)
 {
   for (j=0; j<m;j++)
     printf("%6.1f",f[i][j]);
   printf("\n");
 }
 min=0; otr=false; //задание начальных значений для минимума и признака
                   //наличия отрицательных элементов
 for (i=0; i<m-1;i++)  //цикл поиска минимального отрицательного
                       // элемента над побочной диагональю
   for (j=0;j<m-i-1;j++)
     if (f[i][j]<min)
       {
       min=f[i][j];
       otr=true;
       }
 max=0; pol=false; //задание начальных значений для максимума и
                   // признака наличия положительных элементов
 for (i=1; i<m; i++)    //цикл поиска максимального положительного
                       // элемента под побочной диагональю
   for (j=m-i;j<m; j++)
     if (f[i][j]>max)
       {
       max=f[i][j];
       pol=true;
       }
 if (!otr)
   printf("\notrizatelnix elementow net");
 else if (!pol)
   printf("\npoloshitelnix elementow net");
 else
 {
   if (fabs(min)>max)
```

```
            {printf("\nmodul naimenschego otrizatelnogo bolsche naibolschego\
poloshitelnogo\n");
            printf("min=%6.1f    max=%6.1f",min,max);
            }
          if (fabs(min)<max)
            {printf("\nmodul naimenschego otrizatelnogo mensche naibolschego\
poloshitelnogo\n");
             printf("min=%6.1f    max=%6.1f",min,max);
            }
          if (fabs(min)==max)
            {printf("\nmodul naimenschego otrizatelnogo rawen naibolschemu\
poloshitelnomu\n");
             printf("min=%6.1f    max=%6.1f",min,max);
            }
        }
      getch();
      return 0;
      }
```

5.7. Задания для самостоятельной работы

1. Составить программу, которая в матрице A(m,n), m≤10, n≤12, меняет местами строку, содержащую максимальный элемент со строкой, содержащей минимальный элемент. Предполагается, что искомые элементы единственные. Вывести исходную и преобразованную матрицы, минимальный и максимальный элементы, а также номера строк, в которых они расположены. Если минимальный и максимальный элементы расположены в одной строке, то поменять местами столбцы, содержащие эти элементы.

2. Составить программу, которая в каждой строке матрицы B(m,n), m≤10, n≤12, находит модуль суммы отрицательных элементов и сумму положительных элементов. Из найденных элементов сформировать матрицу C(m,3), в каждой строке которой первые два элемента - найденные суммы, а третий элемент равен -1, если первая сумма больше второй, 0, если они равны, 1, если вторая сумма больше первой. Вывести исходную и полученную матрицы так, чтобы в каждой строке сначала располагалась строка исходной матрицы, а затем строка полученной матрицы.

3.Составить программу, которая в каждой строке матрицы D(m,n), m≤10, n≤12, находит элемент, для которого модуль разности этого элемента и среднего арифметического элементов строки минимален. Вывести исходную матрицу так, чтобы после элементов

строки матрицы располагались найденный элемент, среднее арифметическое и модуль их разности.

4.Составить программу, которая в матрице K(m,n), m≤10, n≤12, меняет местами строки, содержащие максимальное количество четных и максимальное количество нечетных элементов. Если во всех строках эти количества одинаковы, то поменять местами первую и последнюю строки матрицы. Вывести исходную и преобразованную матрицы, найденные количества и номера найденных строк.

5.Составить программу, которая в квадратной матрице F(m,m), m≤10, находит сумму всех элементов верхней треугольной матрицы, которые больше всех элементов нижней треугольной матрицы. Вывести исходную матрицу и найденную сумму, если верхняя треугольная матрица не содержит нужных элементов, то выдать соответствующее сообщение.

6.Составить программу, которая в каждой строке матрицы H(m,n), m≤10, n≤12, находит максимальное из произведений вида hi1, h i1hi2, h i1hi2 hi3,…., (h i1h i2…h in). Вывести исходную матрицу и рядом с каждой строкой найденное максимальное значение из произведений.

7.Составить программу, которая в каждой строке матрицы G(m,n), m≤10, n≤12 находит сумму элементов, расположенных до максимального элемента и после максимального элемента. Если сумма не может быть вычислена (нет элементов до или после максимального элемента), то считать ее равной нулю. Вывести исходную матрицу, располагая в одной строке элементы строки матрицы, после которых вывести номер столбца максимального элемента и найденные суммы.

8.Составить программу, которая находит в каждой строке матрицы S(k,l), k≤12, l≤15, самую длинную последовательность отрицательных чисел и произведение элементов этой последовательности. Если строка не содержит отрицательных чисел, то считать произведение равным нулю. Вывести исходную матрицу, располагая в одной строке элементы строки матрицы, после которых длину найденной последовательности и ее произведение.

9.Составить программу, которая находит в каждой строке матрицы P(k,l), k≤12, l≤14, сумму элементов с нечетными номерами столбцов и сумму элементов с четными номерами столбцов. Найти максимальное значение из первых сумм и минимальное из вторых сумм. Вывести исходную матрицу, располагая в одной строке элементы строки матрицы, после которых найденные суммы, максимальное и минимальное значения.

10.Составить программу, которая находит в каждой строке матрицы Q(k,l), k≤12, l≤14, произведение элементов, расположенных между минимальным и максимальным элементами этой же строки. Если произведение вычислить нельзя (нет элементов между минимальным и максимальным), то считать его равным нулю. Вывести исходную матрицу, располагая в одной строке элементы строки матрицы, после которых найденное произведение, минимальное и максимальное значения.

11.Составить программу, которая находит в каждой строке матрицы Q(k,l), k≤12, l≤14, сумму положительных элементов, расположенных между первым и последним отрицательными элементами этой же строки. Если сумму вычислить нельзя (нет положительных элементов между первым и последним отрицательными элементами), то считать ее равной нулю. Вывести исходную матрицу, располагая в одной строке элементы строки матрицы, после которых - найденную сумму, первый и последний отрицательные элементы.

12.Составить программу, которая находит в каждой строке матрицы Q(k,l), k≤12, l≤14, среднее арифметическое максимального отрицательного и минимального положительного элементов. Найти максимальное среднее арифметическое и номер строки, для которой оно получено. Если среднее вычислено быть не может (нет отрицательных или положительных элементов в строке), то считать его равным нулю. Вывести исходную матрицу, располагая в одной строке элементы строки матрицы, после которых найденные максимальное и минимальное, их среднее арифметическое. Под матрицей вывести максимальное среднее арифметическое и номер строки.

13.Составить программу, которая в каждой строке матрицы D(m,n), m≤10, n≤12, находит элементы, для которых сумма предшествующих элементов больше суммы последующих элементов. Для первого элемента сумму предшествующих элементов считать равной нулю. Для последнего элемента сумму последующих элементов считать равной нулю. Вывести матрицу в виде матрицы, располагая рядом с каждой строкой найденные элементы.

14.Составить программу, которая в каждой строке матрицы D(m,n), m≤10, n≤12, находит элемент, для которого модуль разности этого элемента и среднего геометрического модулей всех элементов строки максимален. Предполагается, что матрица нулевых элементов не содержит. Вывести матрицу в виде матрицы, располагая рядом с каждой строкой найденный элемент и модуль искомой разности.

15.Составить программу, которая в матрице D(m,n), m≤10, n≤12, находит все элементы, для которых сумма всех элементов строки, стоящих до рассматриваемого элемента,

больше суммы элементов столбца, стоящих до рассматриваемого элемента. Сумму предшествующих элементов считать равной нулю, если элемент является первым в строке или в столбце. Сформировать из найденных элементов массив. Вывести матрицу в виде матрицы, а под ней – элементы массива.

16.Составить программу, которая в матрице D(m,n), m≤10, n≤12, находит все элементы, модуль которых располагается в интервале между средним геометрическим модулей всех элементов и средним арифметическим модулей всех элементов матрицы. Из найденных элементов сформировать одномерный массив. Вывести матрицу в виде матрицы, а под ней – элементы массива. Предполагается, что матрица нулевых элементов не содержит.

17.Составить программу, которая в матрице D(m,n), m≤10, n≤12, находит элемент, для которого сумма его четырех ближайших соседей (двух элементов, стоящих перед ним в строке и в столбце, и двух, стоящих после него в строке и в столбце) максимальна. Если соседний элемент отсутствует, то считать его равным нулю. Вывести матрицу в виде матрицы, а под ней – найденный элемент, его номера строки и столбца и сумму.

18.Составить программу, которая в матрице D(m,n), m≤10, n≤12, находит для элементов, сумма индексов которых нечетна, максимальный элемент и сумму элементов. Найти также максимальный элемент и сумму элементов, для которых сумма индексов четна. Вывести матрицу в виде матрицы, а под ней – найденные максимальные элементы и их индексы и две суммы.

19.Составить программу, которая в матрице D(m,n), m≤10, n≤12, находит все элементы, для которых максимальный элемент среди предшествующих элементов строки, в которой стоит элемент, превышает максимальный элемент среди предшествующих элементов столбца, в котором расположен элемент. Если предшествующие элементы отсутствуют, то считать максимальный равным нулю. Найденные элементы переписать в одномерный массив. Вывести матрицу в виде матрицы, а под ней – элементы сформированного массива.

20.Составить программу, которая в матрице K(m,n), m≤10, n≤12, меняет местами строки, содержащие максимальный элемент, нацело делящийся на заданное число L, и минимальный элемент, нацело делящийся на то же число L . Если найденные элементы расположены в одной строке, то поменять местами столбцы, в которых они расположены. Если в матрице требуемых элементов нет или он единственный, то поменять местами первую и последнюю строки матрицы. Вывести исходную и преобразованную матрицы, найденные элементы и их индексы.

21.Составить программу, которая в квадратной матрице F(m,m), m≤10, находит одноименные строки и столбцы с равными суммами элементов. Номера найденных строк запомнить в массиве. Дополнительных массивов для сохранения значений сумм не использовать. Вывести исходную матрицу и номера найденных строк и их сумм. Если требуемые строки и столбцы отсутствуют, то выдать соответствующее сообщение.

22.Составить программу, которая в матрице D(m,n), m≤10, n≤12, находит номера строк, в которых максимальный элемент среди элементов с четными индексами столбцов совпадает с максимальным элементом из элементов с нечетными индексами столбцов. Вывести исходную матрицу, номера найденных строк и максимальные элементы. Если требуемых строк нет, то выдать соответствующее сообщение.

23.Составить программу, которая в квадратной матрице F(m,m), m≤10, находит произведение всех элементов нижней треугольной матрицы, которые меньше минимального элемента верхней треугольной матрицы. Вывести исходную матрицу и найденное произведение, если нижняя треугольная матрица не содержит нужных элементов, то выдать соответствующее сообщение.

24.Составить программу, которая находит в матрицы Q(k,l), k≤12, l≤14, все строки, произведение элементов которых больше суммы тех же элементов. Определить среди найденных строк строку, для которой разность произведения и суммы максимальна. Вывести исходную матрицу, располагая рядом с элементами каждой строки найденные сумму и произведение. Вывести под матрицей номера найденных строк, номер строки с максимальной разностью или сообщение об отсутствии искомых строк.

25.Составить программу, которая в матрице D(m,n), m≤10, n≤12, находит номера строк, в которых каждый элемент больше наибольшего из элементов того же столбца, расположенных до рассматриваемого элемента. Вывести исходную матрицу, номера найденных строк. Если требуемых строк нет, то выдать соответствующее сообщение.

26. Составить программу, которая в матрице D(m,n), m≤10, n≤12, находит номера строк с максимальным и минимальным значениями среднего квадратического отклонения.

Среднее квадратическое отклонение элементов i-ой строки вычисляется по следующей формуле $\sigma i = \sqrt{\frac{\sum_{j=1}^{n}(d_{ij}-d_{icc})^2}{n}}$, где dicp = $\sum_{j=1}^{n} d_{ij}$ /n. Вывести исходную матрицу, номера найденных строк и значения найденных минимального и максимального, средних квадратических. Дополнительных массивов не использовать.

5.8.Методы сортировки массивов

Для *сортировки* (*упорядочения*) по возрастанию или убыванию значений в массиве разработано множество методов [6, 8]. Рассмотрим три из них, считая, для определённости, что первые n, n=6, элементов массива X

X	X	X	X	X	X
0	1	2	3	4	5
34	21	15	18	25	40

требуется упорядочить по возрастанию, не используя дополнительных массивов.

Метод включения с сохранением упорядоченности (метод прямого включения или сортировка вставками).

Сортировка начинается со сравнения чисел в первой (X_0) и второй (X_1) ячейках массива. Если окажется, что $X_0 > X_1$, то их значения меняют местами.

X	X	X	X	X	X
0	1	2	3	4	5
21	34	15	18	25	40

На каждом следующем i-том шаге, i=1,2, 3,…,n-2, значение из (i+1)-ой ячейки массива путем обмена положением с числом из предыдущей ячейки продвигают в сторону уменьшения индекса ячейки до тех пор, пока не окажется, что в предыдущей ячейке находится не большее число.

Из сказанного следует, что при реализации метода прямого включения внешний цикл должен выполняться n-1 раз, а максимально возможное число выполнений внутреннего цикла, в теле которого должны выполняться сравнения и перестановки чисел, будет увеличиваться от 1 до n-1. Однако внутренний цикл следует организовать так, чтобы он заканчивался или вообще не выполнялся при наступлении условия:: значение в предыдущей ячейке массива не больше, чем в текущей.

В нашем примере:

– при i=1 массив не изменяется, так как число из ячейки X_0 меньше числа из ячейки X_1,

– при i=2 число 15 из ячейки X_2 последовательно обменяется местами с числом 34 из ячейки X_1, а затем с числом 21 из ячейки X_0,

X	X	X	X	X	X
0	1	2	3	4	5
15	21	34	18	25	40

- при i=3 число 18 из ячейки X_3 последовательно обменяется местами с числом 34 из ячейки X_2, а затем с числом 21 из ячейки X_1,

X	X	X	X	X	X
0	1	2	3	4	5
15	18	21	34	25	40

- при i=4 число 25 из ячейки X_4 обменяется местами с числом 34 из ячейки X_2,

X	X	X	X	X	X
0	1	2	3	4	5
15	18	21	25	34	40

- при i=5 число 40 из ячейки X_5 не изменит своего положения, так как оно больше числа в ячейке X_4,

X	X	X	X	X	X
0	1	2	3	4	5
15	18	21	25	34	40

Ниже представлен фрагмент программы упорядочения по возрастанию первых n элементов массива X *методом прямого включения* (включения с сохранением упорядоченности).

```
for (i=1;i<nn;i++)
{
	j=i;
	while((aa[j-1]>aa[j])&&(j>=1))
	{
		bb=aa[j];
		aa[j]=aa[j-1];
		aa[j-1]=bb;
		j--;
	}
}
```

Для упорядочения чисел в массиве по убыванию достаточно на каждом шаге изменить условие перестановки чисел в соседних ячейках массива на обратное, а именно, обмен значениями соседних ячеек выполнять в случае, когда предыдущее меньше текущего.

Другая реализация метода основана не на перестановке соседних, а на сдвиге данных в массиве. На очередном шаге i=1,2,3,…,n-1: 1) копируют значение X_i в дополнительную переменную, например, R; 2) начиная с k=1, пока X_{i-k}>R и k<i, копируют X_{i-k} в X_{i-k+1} и увеличивают k на единицу; 3) копируют в X_{i-k+1} значение R.

Метод прямого обмена (метод пузырька).

Этот метод, как и предыдущий, основан на обмене значениями соседних ячеек массива, но с первого же шага в последовательном анализе, при движении от одного конца массива к другому, участвуют все пары соседних ячеек массива.

На первом шаге последовательно, для j = 0,1,2, . . ., n-2 сравниваются значения соседних ячеек массива, и при выполнении условия X_{j+1}<X_j выполняется их перестановка, в результате чего наибольшее число оказывается в ячейке X_{n-1}.

Для примера возьмем следующий массив

X 0	X 1	X 2	X 3	X 4	X 5
4 0	3 4	2 1	2 5	1 8	1 5

В нашем примере после выполнения первого шага (i=0) данные в массиве расположатся так:

X 0	X 1	X 2	X 3	X 4	X 5
3 4	2 1	2 5	1 8	1 5	4 0

На втором шаге (i=1) процесс повторяется, но только для j = 0.1,2,. . ., n-3

X 0	X 1	X 2	X 3	X 4	X 5
2 1	2 5	1 8	1 5	3 4	4 0

На каждом следующем шаге число проверяемых пар ячеек будет уменьшаться на 1. В общем случае, на любом шаге i, i=0,1, 2, 3, …, n-2, процесс будет выполняться для j от 0 до n-i-2, в частности, при i= n-2 – только один раз для 0-ой и 1-вой ячеек.

Из сказанного следует, что при реализации метода прямого обмена внешний цикл должен выполняться n-1 раз, а число выполнений внутреннего цикла, в теле которого должны выполняться сравнения и перестановки чисел, будет уменьшаться от n-1 до 1.

Происхождение термина “метод пузырька” объясняется так: если представить вертикальное расположение ячеек массива с ростом индекса сверху вниз, то самое маленькое число из рассматриваемых будет подниматься вверх подобно пузырьку в воде.

В нашем примере

- при i=2 перестановки приведут к следующему состоянию массива

X	X	X	X	X	X
0	1	2	3	4	5
21	18	15	25	34	40

- при i=3

X	X	X	X	X	X
0	1	2	3	4	5
18	15	21	25	34	40

- при i=4

X	X	X	X	X	X
0	1	2	3	4	5
15	18	21	25	34	40

При использовании метода пузырька не имеет значения, в сторону увеличения или в сторону уменьшения индексов продвигается анализ пар чисел в массиве, а вид упорядочения (по возрастанию или убыванию) определяется только условием перестановки чисел (меньшее должно расположиться за большим или наоборот).

Ниже представлен фрагмент программы упорядочения по возрастанию первых n элементов массива X *методом пузырька***.**

```
for (i=1;i<n-1;i++)
{
      for (j=0;j<n-i;j++)
            if (x[j]>x[j+1])
```

```
            {
                  d=x[j];
                  x[j]=x[j+1];
                  x[j+1]=d;
            }
}
```

Модифицированный метод прямого обмена (модифицированный метод пузырька).

При использовании метода пузырька массив может оказаться упорядоченным уже после i-го шага (i<n-1), то есть возможно выполнение внешнего цикла не n-1 раз, а меньше, когда станет известно, что массив уже упорядочен. Такая проверка основывается на следующем: если при выполнении внутреннего цикла не было ни одной перестановки, значит массив уже упорядочен и можно выйти из внешнего цикла. В качестве признака, выполнялась ли перестановка, используют переменную булевского типа: до входа во внутренний цикл ей дают одно значение, например, false, а при выполнении перестановки – другое, например, true.

Очевидно, эффект при использовании модифицированного метода пузырька по сравнению с не модифицированным методом в ускорении процесса сортировки будет наблюдаться, если исходная последовательность чисел близка к упорядоченности в нужном направлении. В предельном случае, когда массив уже упорядочен нужным образом, тело внешнего цикла будет выполнено только один раз.

Ниже представлен фрагмент программы упорядочения по возрастанию первых n элементов массива X *модифицированным методом пузырька.*

```
i=1;
do
{
      flag=0;
      for (j=0;j<n-i;j++)
            if (x[j]>x[j+1])
            {
                  d=x[j];
                  x[j]=x[j+1];
                  x[j+1]=d;
                  flag=1;
            }
      i++;
}while(flag);
```

Метод прямого выбора (сортировки посредством выбора) и его модификации

Все описываемые далее методы следует рассматривать как частные варианты метода, известного под названием *метод прямого выбора* или *сортировка посредством выбора*. Общим для этих методов является нахождение (выбор) максимальных или минимальных элементов массива и размещение их в последовательных ячейках массива.

Сортировка методом поиска минимального элемента

Суть этого метода (имея в виду выбранные ограничения для рассматриваемого нами числового примера) состоит в следующем.

На первом шаге отыскивается и сохраняется в переменной, например, xmin минимальное число среди всех чисел массива и его индекс, сохраняемый в другой переменной, например, imin, а затем проводится обмен местами в массиве найденного минимального числа с нулевым элементом массива: x[imin]=x[0]; x[0]=xmin;. (при записи операторов учтено, что индекс первого элемента массива в языке С равен 0).

В нашем примере

X	X	X	X	X	X
0	1	2	3	4	5
34	21	15	18	25	40

минимальное число xmin=15 находится в ячейке imin=2, и перестановка нулевого и минимального чисел приведёт к следующему результату

X	X	X	X	X	X
0	1	2	3	4	5
15	21	34	18	25	40

В общем случае, на любом шаге i, i=0,1, 2, 3, …, n-2, отыскивается xmin - минимальное число среди ячеек массива с индексами от i до n-1 и его индекс imin, а затем проводится обмен местами в массиве найденного минимального числа с i-тым элементом массива: x`[imin]=x[i]; x[i]=xmin;.`

В нашем примере

- при i=1 получим xmin=18 и imin=3, и после перестановки

X	X	X	X	X	X
0	1	2	3	4	5
15	18	34	21	25	40

- при i=2 получим xmin=21 и imin=3, и после перестановки

X	X	X	X	X	X
0	1	2	3	4	5
15	18	21	34	25	40

- при i=3 получим xmin=25 и imin=4, и после перестановки

X	X	X	X	X	X
0	1	2	3	4	5
15	18	21	25	34	40

- при i=4 получим xmin=34 и imin=4, и после перестановки

X	X	X	X	X	X
0	1	2	3	4	5
15	18	21	25	34	40

Таким образом, внешний цикл должен выполняться n-1 раз, а число выполнений внутреннего цикла будет уменьшаться от n-1 до 1. Чтобы упорядочить массив по убыванию, следует первое найденное минимальное число обменять местами с последним, второе – с предпоследним и так далее.

Сортировка методом поиска максимального элемента

Этот метод отличается от предыдущего только тем, что отыскиваются максимальные элементы. При одинаковой организации циклов при реализации этих методов они дадут прямо противоположные результаты: если один приведёт к возрастанию чисел в массиве, то другой – убыванию, и наоборот.

Сортировка методом поиска индекса минимального элемента

Этот метод отличается от метода поиска минимального элемента и его индекса тем, что внутренний цикл используется для поиска только индекса минимального

элемента, поэтому перестановки чисел в массиве на каждом шаге i, i=0,1, 2, …,n-2 придется выполнять с привлечением дополнительной переменной, например, r: r:=x[i]; x[i]:=x[imin]; x[imin]:=r;.

Сортировка методом поиска индекса максимального элемента

Этот метод отличается от предыдущего только тем, что отыскиваются индекс максимального элемента. При одинаковой организации циклов при реализации этих методов они дадут прямо противоположные результаты: если один приведёт к возрастанию чисел в массиве, то другой – убыванию, и наоборот.

5.9.Пример выполнения задания

Программа составлена по условию варианта задания №30 (см. ниже).

```
#include "stdafx.h"
#include "conio.h"
#define dbg //после отладки программы превратите эту строку в комментарий

int _tmain(int argc, _TCHAR* argv[])
{
	const int nn=12;
	int n=10,imax,k,i;
	float x[nn]={50,23,123, 34,125, 54, 231, 3,222, 13, 5,100},y;
#ifndef dbg
	//при отладке программы ввод исходных данных
	//не использовать (замкомментировать директиву #define dbg)
	printf("Wwedite kol-wo elementow\n");
	scanf("%d",&n);
	printf("\nWwedite  elementi\n");
	for (i=0;i<n;i++)
		scanf("%f",&x[i]);
#endif dbg
	printf("\n Isxodnij massiw\n");
	for (i=0;i<n;i++)
		printf("%5.2f ",x[i]);
	printf("\n");
	i=0;
	do
	{
		//Поиск индекса imax максимального элемента
		//массива среди элементов с индексами от i до N
```

```
            imax=i;
            for (k=i+1;k<n;k++)
                  if (x[k]>x[imax])
                        imax=k;
            //Взаимная перестановка X[i] с X[iMax]
            y=x[i];
            x[i]=x[imax];
            x[imax]=y;
#ifdef dbg
            //Операторы, используемые при отладке
            //Вывод массива после перестановки элементов X[i] с X[imax]
            printf("\n Promeshutochnij massiw\n");
            for (k=0;k<n;k++)
                  printf("%5.2f ",x[k]);
            printf("\n imax=%d\n",imax);
#endif dbg
            i++;
      }while (i<n-1);
      printf("\n Otsortirowannij massiw\n");
      for (i=0;i<n;i++)
            printf("%5.2f ",x[i]);
      getch();
      return 0;
}
```

5.10. Задания для самостоятельной работы

Составить программу упорядочения первых N, N ≤12, элементов массива X. Вид сортировки, а также метод сортировки и операторы внешнего и внутреннего циклов, которые следует использовать в программе, указаны для каждого варианта в расположенной ниже таблице.

При отладке использовать начальные значения N и массива X, а также выполнять форматный вывод первых N элементов массива одной строкой в конце каждого шага выполнения работ во внешнем цикле.

№	Вид сортировки	Метод сортировки	Оператор внешнего цикла	Оператор внутреннего цикла
1	по возрастанию	прямого включения	while	while

2	по убыванию	выбора максимального	for с положительным приращенирем параметра	for с положительным приращением параметра
3	по возрастанию	выбора минимального	for с положительным приращением параметра	for с отрицательным приращением параметра
4	по убыванию	выбора индекса максимального	while	for с отрицательным приращением параметра
5	по возрастанию	выбора индекса минимального	for с отрицательным приращением параметра	while
6	по убыванию	прямого обмена (пузырька)	do-while	do-while
7	по возрастанию	модифицированного прямого обмена (пузырька)	while	for с отрицательным приращением параметра
8	по убыванию	прямого включения	for с положительным приращением параметра	while
9	по возрастанию	выбора максимального	for с положительным приращением параметра	while
10	по убыванию	выбора минимального	for с отрицательным приращением параметра	for с положительным приращением параметра

11	по возрастанию	выбора индекса максимального	for с отрицательным приращением параметра	for с положительным приращением параметра
12	по убыванию	выбора индекса минимального	for с положительным приращением параметра	do-while
13	по возрастанию	прямого обмена (пузырька)	for с отрицательным приращением параметра	while
14	по убыванию	модифицированного прямого обмена (пузырька)	do-while	for с отрицательным приращением параметра
15	по возрастанию	прямого включения	for с положительным приращением параметра	while
16	по убыванию	выбора максимального	for с отрицательным приращением параметра	for с отрицательным приращением параметра
17	по возрастанию	выбора минимального	for с положительным приращением параметра	do-while
18	по убыванию	выбора индекса максимального	for с положительным приращением параметра	for с положительным приращением параметра
19	по возрастанию	выбора индекса минимального	while	do-while

20	по убыванию	прямого обмена (пузырька)	for с отрицательным приращением параметра	for с положительным приращением параметра
21	по возрастанию	прямого включения	do-while	while
22	по убыванию	выбора максимального	for с положительным приращением параметра	for с отрицательным приращением параметра
23	по возрастанию	выбора минимального	for с отрицательным приращением параметра	while
24	по возрастанию	выбора индекса максимального	for с отрицательным приращением параметра	for с отрицательным приращением параметра
25	по убыванию	выбора индекса минимального	do-while	while
26	по возрастанию	прямого обмена (пузырька)	for с положительным приращением параметра	for с отрицательным приращением параметра
27	по убыванию	модифицированного прямого обмена (пузырька)	do-while	for с положительным приращением параметра
28	по возрастанию	выбора индекса минимального	for с положительным приращением параметра	for с положительным приращением параметра

29	по возрастанию	модифицированного прямого обмена (пузырька)	while	do-while
30	По убыванию	выбора индекса максимального	do-while	for с положительным приращением параметра

Алфавитный указатель

Вопросы для самопроверки

Приёмы вычисления сумм, произведений и экстремальных значений

1. В чем состоит прием накопления суммы, произведения?

2. В каком месте программы надо задавать начальные значения суммы и произведения и каковы должны быть эти значения?

3. Сформулируйте правило нахождения наибольшего (наименьшего) значения функции на заданном интервале изменения аргумента.

4. Какие значения целесообразно задавать в качестве начальных для максимального (минимального) значения и его индекса при его нахождении среди элементов массива?

5. Сформулируйте правило нахождения локального экстремума функции на заданном интервале изменения ее аргумента.

Вычисление суммы бесконечного ряда с заданной точностью

1. Какие формулы называют рекуррентными?

2. Как получить рекуррентную формулу для вычисления значений членов бесконечного ряда?

3. В чём суть смешанного способа вычисления значений членов бесконечного ряда?

4. В чём суть приёма вычисления суммы бесконечного ряда с точностью ε?

Уточнение корней уравнений

1. К какому виду следует привести уравнение при использовании метода простых итераций для уточнения корня?

2. В каких случаях не применим:

- метод простых итераций;

- метод касательных;

- метод половинного деления?

3. Существуют ли корни уравнений, которые невозможно уточнить методом половинного деления?

4. Выведите формулу для вычисления очередного приближения корня уравнения для метода касательных.

Вычисление определённых интегралов

1. На основе какого приёма программирования основаны все рассмотренные методы вычисления определённых интегралов?

2. В чём состоит основанное отличие всех рассмотренных методов вычисления определённых интегралов?

3. За счёт чего можно сократить объём вычислений определённого интеграла по методу трапеций?

Вложенные циклы

1. Какая алгоритмическая структура называется вложенным циклом ?

2. Дайте понятие внутреннего цикла, внешнего цикла.

3. Сформулируйте основное правило организации вложенных циклов.

4. Приведите примеры допустимых вариантов организации вложенных циклов.

5. Как определить количество повторений операторов, расположенных внутри внутреннего цикла, при использовании вложенного цикла?

6. Приведите примеры допустимой и недопустимой передачи управления во вложенных циклах.

7. Сформулируйте основное правило вычисления определенного интеграла с заданной точностью.

8. Объясните назначение внутреннего и внешнего циклов при вычислении определенного интеграла с заданной точностью.

9. Как можно уменьшить трудоемкость вычисления значения определенного интеграла с заданной точностью?

10. Как вычисляется наибольшее (наименьшее) значение функции на заданном интервале с заданной точностью?

11. Как можно объявить двумерный массив в программе?

12. Как можно обратиться к элементам массива?

13. Как организовать ввод-вывод элементов двумерного массива?

14. Как располагаются элементы двумерного массива в памяти ЭВМ?

15. Влияет ли порядок расположения циклов на результат при обработке двумерных массивов?

16. Укажите номера строк и столбцов элементов матрицы, расположенных над (под) главной диагональю.

17. Укажите номера строк и столбцов элементов матрицы, расположенных над (под) побочной диагональю.

18. Какой из рассмотренных методов сортировки массива потребует меньшего количества шагов выполнения внешнего цикла, если к уже упорядоченному по возрастанию массиву добавить ещё один элемент: 1) в начало массива, 2) в конец массива, 3) внутрь массива?

19. Что является главным результатом первого шага выполнения внешнего цикла в рассмотренных методах сортировки: выбора максимального, прямого обмена (пузырька), прямого включения?

20. Сколько раз должно выполняться тело внешнего цикла в каждом из рассмотренных методов сортировки массива при наихудшем исходном размещении данных в массиве?

Список литературы

1. Алексеев Ю.Е., Куров А.В. Практикум по программированию на языке С в среде VS C++. Часть 1. – М.: Мзд. МГТУ им. Н.Э. Баумана, 2011. -100с.

2.Блюмин А.Г., Гусев Е.В., Федотов А.А. Численные методы. Методические указания к лабораторным работам. /Под ред. Г.А.Колотушкина. – М.:Из-во МГТУ им.Н.Э.Баумана. -2002.-48с.

3. Керниган Б. И., Ритчи Д. М. Язык программирования С, 2-е издание,: Пер. с англ. – М.:Издат. дом «Вильямс» , 2006. – 304 с.: ил.

4. Лафоре Р. Объектно-ориентированное программирование в С++. Классика Computer Science. 4-е изд. – СПб.: Питер, 2008. -928 с.: ил.

5.Мак-Кракен Д., Дорн У. Численные методы и программирование на Фортране.-Пер. с англ. - М.:Мир.- 1977.-584 с.

6. Павловская Т.А. С/С++. Программирование на языке высокого уровня. – СПб:Питер, 2007.-461 с.: ил.

7. Пахомов Б.И. С/С++ и MS Visual C++ 2008 для начинающих. – СПб: БХВ-Петербург, 2008.- 624 с.: ил.