ПЛОВДИВСКИ УНИВЕРСИТЕТ

„ПАЙСИИ ХИЛЕНДАРСКИ”

Факултет по Математика и Информатика

Катедра „Компютърни технологии”

ДИПЛОМНА РАБОТА

Тема:

Използване на някои компютърни програми при решаването на математически задачи

Дипломант:

Гергана Димова

Математика

Фак. номер 0701051014

Научен ръководител:

доц. д-р Коста Гъров

Пловдив 2011

Съдържание:

Увод…….....…3

1.Програма за изчисляване стойност на полином и операция с полиноми…….....5

1.1.Прогрма за пресмятане на стойност на полином Pn(x)…….....….5

1.2.Операция с полиноми…….....…8

2.Интерполационен полином. Програмиране на формулите на Лагранж и Нютон.Обратна интерполяация.Екстраполация.....……..12

2.1 Приближаване на функция.Интеролация. Интерполационен полином…….....12

2.2 Интерполационна формула на Лагранж…….....16

2.3. Крайни разлики. Интерполационна формула на Нютон…….....…….....19

2.4. Понятие за обратна интерполация и екстраполация……...……24

3. Табулиране в с++…….....28

3.1. Табулиране на функция…….....…….28

4. Двумерни масиви и работа с тях…….....…….....28

4.1. Операция с матрици…….....…….28

4.2. Линеарнизация на двумерни масиви…….....…….36

5. Методи за решаване на системи…….....……39

5.1. Метод на Гаус за последователно изключване на неизвестните….....…….40

5.2. Метод на Гаус- Жордан…….....…45

5.3. Итерационни методи…….....……48

6. Изчисляване на лице на крволинеен трапец…….....56

6.1. Лице на криволинеен трапец…….....….56

6.2. Формула на правоъгълниците…….....58

6.3. Формула на трапеците…….....58

6.4. Формула на Симпсон…….....…….59

Заключение…….....…….....60

Използвана литература …….....….61

УВОД

Обучението по информатика в училище е насочено към овладяване на базисни знания, умения и отношения свързани с учебните дисциплини. Тези базисни компетенции са задължителна част от техническата грамотност на съвременния млад човек и създават условия за пълноценното му реализиране в живота. Учебното съдържание е представено чрез очакваните резултати по определени чрез държавните образователни изисквания. Обучението се осъществява на спираловиден принцип. Заложеното в учебното съдържание се надгражда, като се акцентира върху изграждане на знания и умения за създаване на документи с графични обекти. В останалите теми акцентът е върху придобиване на начални представи и умения за използване на приложен софтуер и експериментиране при решаване на задачи, свързани с изучаването в учебния план. Основните компоненти на този план за действие са заложени и в стратегията на министерството на образуването и науката за въвеждане на информационните и комуникативните технологии в българските училища. Главна цел при обучението на учениците е стратегията за ефективното използване на съвременните информационни и мрежови технологии, за повишаване качеството на образователните технологии и методи за в учебния процес а именно:

1. Да придобият увереност в собствените си компетенции.
2. Да се ориентират в съдържанието на информация, представена на различни носители или в Интернет.
3. Да умеят да експериментират с информация в текстов, табличен и графичен формат.
4. Да придобият престава за възможност на компютърните системи да обработват и представят разнородна информация.

Основните акценти в оценяване постиженията на учениците е върху придобиване на практически умения за реализирането на определени задачи на

компютъра и наличието на първоначални знания за същността на изучаваните предмети.

Самият процес на реализация на поставената задача е труден за оценяване при групова работа в клас, поради невъзможността учителят да следи едновременно работата на всички ученици. Ето защо оценяването на постиженията на учениците става на база завършени проекти (учебни задачи) по отделните теми, а не изпълнението на една или друга задача, която е елемент от работа по проект. Наличието на краен резултат, който отговаря на първоначално поставената задача, е достатъчно основание да се счита, че ученикът притежава необходимите знания и умения. Като писмена форма на оценяване могат да бъдат използвани чек листове за проверка на знанията по предметите. Наличието на изброени твърдения и отговори на задачи, с които ученикът трябва да се съгласи или отхвърли, лесно за разбиране от обучаемите и позволява реализирането на самооценка от тяхна страна. За самооценка могат да се използват и електронни варианти.

Проверката на знания в устна форма може да бъде направено чрез диалог, в който се поставят въпроси от учителя или други ученици, свързани с реализирането на поставената задача, а не върху обяснението на основни принципи и понятия. Тук е препоръчително оценката на преподавателя да бъде направена в качествена, а не в количествена форма.

Обучението в клас се извършва основно на практическа основа. Провеждането на часовете в изцяло лекционна форма не е препоръчително. Когато е необходимо наличието на такъв учебен час, учителят трябва да осигури наличието нас активни фази, в които учениците да споделят своите знания или виждания по разглежданата тема.

Акцентът при работата трябва да бъде върху разработването на учебните задачи, под формата на тематични проекти, чрез които се усвоява трайно и осъзнава предназначението на инструментариума в използваното програмно средство, е не механичното изпълнение на последователност от стъпки.

Представянето на един или друг инструмент, команда или функция от изучаваното софтуерно средство става на базата необходимостта от неговото използване при решаване на конкретната задача, по която се работи. Ето защо подборът на задача за реализация в клас трябва да бъде направена, така че да осигури възможността за представянето на новото средство.

Материалът е насочен към успешното използване на тези технологии в подпомагане усвояването на учебното съдържание от предметите, изучавани в задължителна подготовка. Това от своя страна определя избора на теми за реализация да бъде тясно свързан с изучавания в момента материал по различните учебни дисциплини.

1. Програма за изчисляване на стойност на полином и операции с полиноми

Алгебричните полиноми са едни от най-често използваните в практиката математически функции при решаването на редица научно-технически задачи с помощта на компютър се изготвят програми за изчисляване стойността на даден полином, за намиране на сума, разлика, произведение на частно на полиноми. Тук ние ще имаме предвид следното представяне на произволен полином на една променлива от *n* –та степен:

(1) $P_n(x) = a_0x^n + a_1x^{n-1} + \ldots + a_ix^{n-i} + \ldots + a_{n-1}x + a_n.$

При съставяне на програми на C++ за полиноми е удобно коефициентите a_0, $a_1, \ldots, a_n$ (n+1 на брой) да се представят като елементи на едномерен масив.

1.1. Програма за пресмятане на стойност на полином $P_n(x)$

Стойността на един полином $P_n(x)$ може да се пресметне непосредствено, като се използва представянето (1). Първо се пресмята стойността на всяко отделно събираемо, след което се извършва сумирането. Съществува обаче и друг начин на пресмятане на стойността на един полином - по т.н. правило на Хорнер. Ето как изглежда това правило например за полином от трета степен:

$$P_3(x) = a_0x^3 + a_1x^2 + a_2x + a_3$$

Този полином може да се запише и така:

$$P_3(x) = ((a_0x + a_{1)}\,x + a_{2)}\,x + a_3$$

При правилото на Хорнер най-напред се пресмята изразът $a_0x + a_1$ от вътрешните скоби. Получената стойност се умножава с x и се прибавя a_2 Стойността на $P_3(x)$ се получава, като се умножи последния резултат с x и към полученото се прибавя a_3 . Очевидно тези разсъждения се пренасят без изменения, когато полиномът е от степен *n.* В този случай стойността на полинома се пресмята по формулата:

$$(2)\quad P_n(x) = (\ldots((a_0x + a_1)\,x + a_2)x + \ldots.. + a_{n-1})x + a_n.$$

Правилото на Хорнер е много удобно за реализация в програма на алгоритмичен език, тъй като многократно се повтаря едно и също действие - определена стойност се умножава по x и към получения резултат се прибавя съответен коефициент на полинома. Освен това, чрез него пресмятането на стойността на полинома става с изпълнение на по-малък брой аритметични операции, отколкото при непосредствено пресмятане .

При пресмятане на стойността на полином от 3-та степен по правилото на Хорнер, ще бъдат извършени следните операции:

Пресмятане на :	Събиране:	Умножение :
$A = a_0x + a_1$	1	1
$B = Ax + a_2$	1	1
$P_3(x) = B + a_3$	1	1

Или общо 3 умножения и 3 събирания .

При непосредственото правило са необходими:

Пресмятане на:	Събирания:	Умножения:

$C = a_2 x$	0	1
$D = a_1 x^2$	0	2
$E = a_0 x^3$	0	3
$P_3(x) = E + D + C + a_3$	3	3

Или общо извършват 6 умножения и 3 събирания. Очевидно при по-големи стойности на *n* броят на умноженията по правилото на Хорнер е много по-малък от броя на умноженията при непосредственото правило

.Ето защо правилото на Хорнер за пресмятането на полином е за предпочитане пред непосредственото правило. Следната програма е една реализация на формула (2) на езика C++.

```
#include <iostream.h>

#include <stdlib.h>

#include <time.h>

typedef duble vector[100];

void InputRan (vecotor a, int n)

{  for (int i=0; i<n; i++) a[i]= rand() %201-100;

}

void output( vector a, int n)

{ fot(int i=0; i<n; i++) cout<< a[i]<< " ";

cout<< endl;

}double  horner (vector a, int n, doublex)

{ double y=a[0];

for(int i=1; i<n; i++) y=y*x+a[i];

return y;
```

```
}

int main ()

{ int n; vector a;

strand (time (NULL));

cout<< "Въведи брой елементи на масива:" ; cin>>n;

if (n<1 n>100 !cin) (cout<< "Error!"; return 1;);

InputRan(a,n);

output (a,n);

double x;cout<<"x="; cin>>x;

cut<<"P("<<x<<')=" <<horner(a,n,x);
```

1.2. Операции с полиноми

а/ събиране и изваждане на полиноми

Нека $P_n(x) = a_0x^n + a_1x^{n-1} + \ldots + a_{n-1}x + a_n$

И

$Q_m(x) = b_0x^m + b_1x^{m-1} + \ldots + b_{m-1}x + b_n.$

Под сума на тези два полинома разбираме полином S (x), чиито коефициенти S_i се получават по формулите

Ако $n = m$, *то* $S_i = a_i + b_i$ *за* $i = 0,1,2,\ldots, n,$

Ако $n > m$, *то* $S_i = a_i$ *за* $i = 0,1,2,\ldots, n - m - 1$

(3) *и*, $S_i = a_i + b_{i+m-n}$ *за* $i = n-m, n-m+1,\ldots, n$

Ако $m > n$, *то* $S_i = b_i$ *за* $i = 0,1,2,\ldots, m-n - 1$

и, $S_i = b_i + a_{i+m-n}$ *за* $i = m-n, m-n+1,\ldots, m$

очевидно степента на полинома S е по- голямото от числата *m* или *n. Разлика на* полиномите $P_n(x)$ и $Q_m(x)$ ще наричаме полинома С (х), чиито коефициенти c_i се получават по формулите:

ако $n = m$, *то* $c_i = a_i + b_i$, *за* $i = 0, 1, 2,\ldots, n$,

ako $n > m$, *то* $c_i = a_i + b_i$, *за* $i = 0, 1, 2,\ldots, n, m$-1

(4) и $c_i = a_i - b_{i+m-n}$, *за* $i = n - m, n - m + 1,\ldots, n$

ako $m > n$, *то* $c_i = b_i$, *за* $i = 0, 1, 2,\ldots, m - n - 1$

и $c_i = a_{i+m-n} - b_i$ *за* $i = m - n, m - n + 1,\ldots, m$

Написването на програми на езика С++ за на суми и разлика на два полинома по формулите (3) и (4) предоставяме на читателя за самостоятелна работа.

б/ Умножение на полиноми

Операцията умножение на полиноми се извършва съгласно познатото ни от алгебрата правило за умножение на суми, като се използва свойството:

$$X^p . X^q = X^{p+q}$$

От тук в частност следва фактът, че полиномът - произведение на полиномите $P_n(x)$ и $Q_m(x)$ *е* полином от степен $n+m$. Получаването на коефициентите на полинома произведение ще илюстрираме със следния пример. Нека $P_2(x) = a_0 x^2 + a_1 x + a_2$ и $Q_3(x) = b_0 x^3 + b_1 x^2 + b_2 x + b_3$. За да получим коефициентите на полинома произведение P R_5 (х) образуваме таблицата

	b_0	b_1	b_2	b_3		
a_0	a_0b_0	a_0b_1	a_0b_2	a_0b_3		
a_1		a_1b_0	a_1b_1	a_1b_2	a_1b_3	
a_2			a_2b_0	a_2b_1	a_2b_2	a_2b_3

Чрез сумиране на вертикала ще получим:

$PR_5(x) = P_2 . Q_3(x) = a_0b_0 x^5 + (a_0b_1 + a_1b_0) x^4 + (a_0b_2 + a_1b_1 + a_2b_0) x^3 + (a_0b_3 + a_1b_2 + a_2b_1) x^2 + (a_1b_3 + a_2b_2) x + a_2b_3$.

Интересното е, че коефициентите PR_5 се получават, като сумата на произведенията на коефициентите $a_i\ b_j$, чийто индекси i и j удовлетворяват условието $i+ j = k$. Това свойство на коефициентите на полинома произведение, което лесно се пренася за произведение на произволни полиноми от степен n и m, съществено се използва при изготвянето на следната подпрограма на C++ за умножение на полиномите $P_n(x)$ и $Q_m(x)$.

```
#include <iostream>
using namespace std;
int main()
{
        cout << "Programa za umnojenie na polinomi" << endl << endl;
        int M, N, K;
   cout << "M = "; cin >> M;
        cout << "N = "; cin >> N;
        NP = M + N
        int A[M];
        int B[N];
        int PR[NP]l;
        for(int i = 0; i < M; i++) {
                cout << "A[" << i+1 << "] = ";
                cin >> A[i];
        }
        for(int i = 0; i < N; i++) {
                cout << "B[" << i+1 << "] = ";
```

```
            cin >> B[i];
        }
        for(int i = 0; i < NP; i++) {
            PR[i] = 0;
        }
        for(int i = 0; i < M; i++) {
            for(int j = 0; j < N; j++) {
                K = i + j;
                PR[K] = PR[K] + A[i] * B[j]
            }
        }
}
```

Ще отбележим че в главната програма, ползваща описаната подпрограма за умножение на полиноми, трябва да се въведат степените на полиномите *m* и *n* и съответните стойности на коефициентите a_i и b_j, както и да се отпечатат коефициентите на полученото произведение.

Упражнение

1. Да се състави програма за непосредствено пресмятане на стойността на полинома от *n*- та степен.
2. Да се състави програма за намиране полинома – сума на два полинома $P_n(x)$ и $Q_m(x)$.
3. Да се състави програма за намиране полинома- разлика на два полинома $P_n(x)$ и $Q_m(x)$.
4. Нека P (x) е полином от степен *m* и Q (x) е полином от степен *n*, като $m \geq n$. Съществува полином T (x) от степен $m - n$ и полином *R(x)* от степен $n - 1$, такива, че е в сила равенството:
   $P(x) = T(x).\ Q(x) + R(x)$

Полинома Т (х) наричаме частно на полиномите *P (x) и Q (x)*, а *R (x)*- остатък.

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

$Q(x) = b_0 x^n + b_1 x^{n-1} + \ldots + b_n$,

то коефициентите c_i на полинома- частно Т (х) се изчислява по формулата:

$c_i = (a_i - \sum_{j=0}^{m-n} c_j\, b_{i-j}) / b_0$, $1 \le i \le m-n$ като $c_0 = a_0 / b_0$.

Коефициентите на остатъка R (x) също ще означим със c_i , като $m-n+1 \le i \le m$. Тях ще изчисляваме по формулата:

$c_i = a_i - \sum_{j=0}^{m-n} c_j\, b_{i-j}$, $m - n + 1 \le i \le m$

и $b_k = 0$, за $m - n + 1 \le \kappa \le m$

по този начин в първите $m - n + 1$ елемента на масива С ще получим коефициентите на полинома частно, а в останалите – коефициентите на остатъка. Съставете подпрограма на С++, реализираща операцията деление на полиноми.

5. *K* и *L* се съставено от m и n- цифрени цели положителни числа. Всяко от тях е представено в едномерен масив, като на всяка цифра съществува по един елемент от масива.

   а / Да се състави подпрограма, която да умножава числата и да записва цифрите на полученото произведение в нов масив.

   б / Да се състави програма, която

   - Да въвежда стойността на *m* и *n,* цифрите *K* и *L.*
   - Да пресмята произведението *K.L* чрез подпрограмата от подусловие *а* /
   - Да отпечатва *K* , *L* и произведението им.

2. Интерполационен полином. Програмиране на формулите на Лагранж и Нютон. Обратна интерполация. Екстраполация

2.1. Приближаване на функции. Интерполация. Интерполационен полином

Известно е, че съществуват различни начини за задаване на математическите функции- аналитичен, табличен, графичен, словесен. Аналитичния начин, във вид на формули, създава най- големи удобства за работа с функцията. В практиката обаче, много често функциите се

задават таблично. Нека аналитичния вид на функцията $f(x)$ не е известен и са познати само стойностите й в краен брой точките на даден интервал $[а, в]$:

x_i	x_0	x_1	x_2	…	x_n
$f(x_i)$	$f(x_0)$	$f(x_1)$	$f(x_2)$	…	$f(x_n)$

Възниква въпросът- може ли да се намерят и други точки на $f(x)$ и в други точки на този интервал?

Такава задача решават, например, учениците в часовете по алгебра, когато се търси логаритъм на число, което не фигурира в логаритмичната таблица, а се намира между две съседни числа в нея. Задачи от този тип възникват ежедневно в практиката. За решението им в математика се възприема следния подход:

Функцията $f(x)$ се заменя с друга функция $F(x)$, чийто израз е познат и чийто стойности в интервала $[а, в]$ са „ близки" в някакъв смисъл до стойностите на $f(x)$. Тогава за приближени стойности на $f(x)$ в интервала $[а, в]$ може да се приемат стойностите на $F(x)$: $f(x) \approx F(x)$, $x \in [а, в]$.

В такъв случай се казва, че се извършва приближаване на функцията $f(x)$ в интервала $[а, в]$ в указания смисъл на „близост" . Важна роля при приближаване играе класа функции, към които принадлежи функцията-приближение $F(x)$. обикновено се избира клас функции, чийто аналитичен вид е удобен и лек за пресмятане, а също така намирането на негов представител $F(x)$, удовлетворяващ критерия за близост, да е възможно и не много трудно. Такъв клас функции са например алгебричните полиноми и най- често приближаването се извършва с тях.

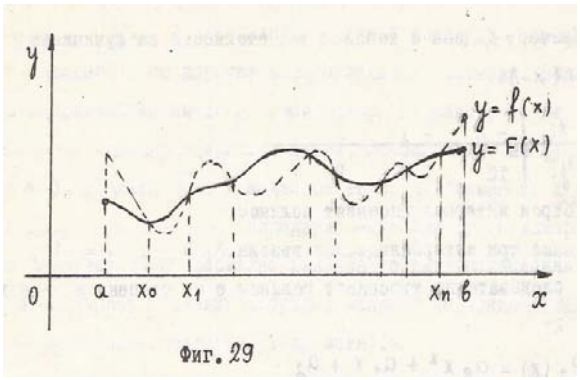
Съществуват много критерии за „близост" на две функции в даден интервал. Всеки един от тях поражда определен вид приближаване. Един от най- често използваните видове приближаване на функции е интерполацията. При интерполацията функцията $F(x)$ се подбира така, че

стойностите й в точките x_0, x_1, x_2, …, x_n да съвпадат със стойностите на функцията *f (x)*:

$$F(x_i) = f(x_i), \quad i = 0, 1, 2, \ldots, n$$

В останалите точки на интервала *[a, в]* стойностите на двете функции могат да се различават произволни. Геометрически (фиг. 29) това означава, че трябва да се построи кривата $y = F(x)$, минаваща през точките $(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)$, където $y_i = f(x_i)$, $i = 0, 1, 2, \ldots, n$

Функцията *F (x)* се нарича интерполираща функция , а точките $x_0, x_1, x_2, \ldots, x_n$ се наричат интерполационни възли.



Фиг. 29

Ако в задачата за интерполация се постави допълнително условие интерполиращата функция да бъде полином, тя придобива следната формулировка: по стойностите $y_0, y_1, y_2, \ldots, y_n$ на функцията *f (x)* в точките $x_0, x_1, x_2, \ldots, x_n$ на интервала *[a, в]* да се намери полином $P_n(x) = a_0 x^n + a_1 x^{n-1} + \ldots + a_{n-1} x + a_n$, от степен най-много равна на *n*, удовлетворяващ условията:

$$\begin{aligned} P_n(x_0) &= y_0 \\ P_n(x_1) &= y_1 \\ &\vdots \\ P_n(x_n) &= y_n \end{aligned} \qquad (1)$$

Полиномът $P_n(x)$ се нарича интерполационен полином на функцията $f(x)$ в точките $x_0, x_1, x_2, \ldots, x_n$. В сила е следното:

Теорема: Съществува точно един полином $P_n(x)$ от степен, най-много равна на n, за който: $P_n(x_i) = y_i \qquad i = 0, 1, 2, \ldots, n$

Интеполационният полином може да се намери непосредствено, като се реши системата (1), в която неизвестни са коефициентите му $a_0, a_1, \ldots, a_n$.

Пример. Дадена е таблица със стойности на функцията

$Y = f(x)$:

x_i	-2	-1	3
y_i	12	6	2

Да се построи интерполациония полином.

Имаме три интерполациони възела $x_0 = -2$, $x_1 = -1$, $x_2 = 3$, следователно търсеният полином е от степен най- много равна на 2:

$$P_2(x) = a_0 x^2 + a_1 x + a_2.$$

Стойностите му при x = -2, -1, 3 трябва да бъдат съответно 12, 6 и 2 . Получава се линейната система от три уравнение с три неизвестни:

$$\begin{cases} a_0(-2)^2 + a_1(-2) + a_2 = 12 \\ a_0(-1)^2 + a_1(-1) + a_2 = 6 \\ a_0 \cdot 3^2 + a_1 3 \qquad a_2 = 2 \end{cases}$$

или

$$\begin{cases} 4a_0 - 2a_1 + a_2 = 12 \\ a_0 - a_1 + a_2 = 6 \\ 9a_0 + 3a_1 + a_2 = 2 \end{cases}$$

Чието единствено решение е $a_0 = 1$, $a_1 = -3$, $a_2 = 2$. Следователно търсеният интерполационен полином е : $P_2(x) = x^2 - 3x + 2$.

Може да се направи извода, че за намиране на коефициентите на интерполационния полином $P_n(x) = a_0 x^n + a_1 x^{n-1} + a_n$ трябва да се реши система от $n+1$ линейни с $n+1$ неизвестни. При по-голяма стойност на n, това е много трудоемко и неоправдано. На практика е удобно да се използва друго представяне на интерполационния полином, което включва данните, с които се разлага – възлите $x_0, x_1, x_2, \ldots, x_n$ и стойностите $y_0, y_1, y_2, \ldots, y_n$. В математиката такива аналитични представяния са издадени и се наричат интерполационни формули. Всяко от тях обикновено носи името на своя автор. В зависимост от разположението на интерполационните възли, особеностите на функцията $f(x)$, точката $\bar{x}$, в която се търси приближената и стойност и др., е удобно да се използва една или друга интерполационна формула. Чисто формално различните интерполационни полиноми се означават с различни букви, въпреки че представляват един и същ полином (съгласно теоремата).

След като намери интерполационния полином $P_n(x)$, за приближена стойност на функцията $f(x)$ във всяка точка в интервала $[a, в]$, се приема стойността на $P_n(x)$ в тази точка, т.е.

$$f(\bar{x}) \approx P_n(\bar{x})$$

2.2. Интерполационна формула на Лагранж

Една от най-разпространените интерполационни формули, носи името на големия френски математик Жозеф Луи Лагранж (1736 – 1813). За извеждането на тази формула е постъпва по следния начин: интерполационния полином се търси във вида:

$$Ln(x) = y_0 p_0(x) + y_1 p_1(x) + \cdots + y_n p_n(x)$$

където $p_0(x), p_1(x), \ldots, p_n(x)$ са полиноми от n - та степен със следните свойства:

(2) $p_i(x) = 1$, за $x = x_i$

(3) $p_i(x) = 0$, за $x = x_0, x_1, \ldots, x_{i+1}, \ldots, x_n$ при $i = 0, 1, 2, \ldots, n$.

тези свойства показват, че всеки от полиномите $p_0(x), p_1(x), \ldots, p_n(x)$ съществува на интерполационния възел със същия индекс - $p_0(x)$ на x_0, $p_1(x)$ на $x_1, \ldots, p_n(x)$ на x_n като съгласно свойство (2) стойността на

полинома за съответния възел е 1, а съгласно свойство (3) в останалите n интерполационни възли стойността му е 0, т.е. тези възли са негови корени. Ако се построят $p_0(x), p_1(x), \ldots, p_n(x)$ с исканите свойства (2) (3) то Ln (x) ще бъде търсеният интерполационен полином, защото

$$Ln(x_i) = y_0 p_0(x_i) + \ldots + y_{i-1} p_{i-1}(x_i) + y_i p_i(x_i) + y_{i+1} p_{i+1}(x) + \ldots$$
$$+ y_n p_n(x_i) = y_0 . 0 + \ldots + y_{i-1} 0 + y_i . 1 + y_{i+1} 0 + \ldots + y_n . 0 = y_i$$

за $i = 0, 1, \ldots, n$. По този начин задачата се свежда до намирането на полиномите $p_0(x), p_1(x), \ldots, p_n(x)$.

За полинома $p_0(x)$, трябва да е изпълнено:

(2') $p_0(x) = 1$

(3') $p_0(x_к) = 0, \quad к = 1, 2, \ldots, n$

От алгебрата за 8 клас е известно, че ако полиномът от втора степен (квадратният тричлен) $T_2(x) = a_0 x^2 + a_1 x + a_2$ има два различни корена x_1 и x_2, той се представя във вида :

$T_2(x) = a_0 (x - x_1)(x - x_2)$. Може да се докаже, че ако полима $T_n(x) = a_0 x^n + a_1 x^{n+1} + a_n$ има n различни корена $x_1, x_2, .., x_n$, той се представя чрез тях по следния начин:

(4) $T_n(x) = a_0 (x - x_1)(x - x_2) \ldots (x - x_n)$

Съгласно (3) възлите $x_1, x_2, .., x_n$ са n различни корена на $p_0(x)$ и според (4)

$p_n(x_0) = a_0 (x - x_1)(x - x_2) \ldots (x - x_n) = 1$,

откъдето

$$a_0 = \frac{1}{(x - x_1)(x - x_2) \ldots (x - x_n)}$$

така за $p_0(x)$ се получава

$$p_0(x) = \frac{(x - x_1)(x - x_2) \ldots (x - x_n)}{(x_0 - x_1)(x_0 - x_2) \ldots (x_0 - x_n)} .$$

в числителя участват всички разлики на променливата x с интерполационните възли с изключение на разликата $x - x_0$, а знаменателят се отличава само по това, че в разликите променливата X се заменя с интерполационния възел x_0. Постъпвайки аналогично, може да се намери представяне на всеки полином p_i , $i = 0, 1, \ldots, n$. Така

$$p_i = \frac{(x - x_0)(x - x1)\dots(x - x_{i-1})(x - x_{i+1})\dots(x - x_n)}{(x_i - x_0)(x_i - x_1)\dots(x_i - x_{i-1})(x_i - x_{i+1})\dots(x_i - x_n)}.$$

замествайки $p_0(x), p_1(x), \dots, p_n(x)$ в $Ln(x)$, получаваме

$$Ln = y_0 \frac{(x - x_1)(x - x_2)\dots(x - x_n)}{(x_0 - x_1)(x_0 - x_2)\dots(x_0 - x_n)} +$$

$$+ y_1 \frac{(x - x_0)(x - x_2)\dots(x - x_n)}{(x_i - x_0)(x_1 - x_2)\dots(x_1 - x_n)} + \dots\dots + \quad (5)$$

$$y_i \frac{(x - x_0)(x - x_1)\dots(x - x_{i-1})(x - x_{i+1})\dots(x - x_n)}{(x_i - x_0)(x_i - x_1)\dots(x_i - x_{i-1})(x_i - x_{i+1})\dots(x_i - x_n)} + \dots\dots + y_n \frac{(x - x_0)(x - x1)\dots(x - x_{n-1})}{(x_i - x_0)(x_i - x_1)\dots(x_i - x_{n-1})}.$$

Формула (5) се нарича интерполационна формула на Лагранж, а самия интерполационен полином Ln (x) се нарича интерполационен полином на Лагранж.

Представената по-долу програма на C++ изчислява стойността на интерполационния полином на Лагранж.

```
#include <iostream>
using namespace std;
int main(int argc, char **argv)
{
	int N;
	int stoinost;
	double ln = 0;
	double PR;
	cout << "Interpolirane po formulata na Lagranj." << endl << endl;
	/*	Povtarya se dokato ne stane > ot 1
		Izpulnyava se pone vednuj
	*/
	do {
		cout << "Vuvedete N: ";
		cin >> N;
	} while(N < 1);
	cout << "Vuvejdane na tablicata na funkciata" << endl;
	int X[N], Y[N];
```

```
	for(int i = 0; i < N; i++) {
		cout << "Vuvedete X(" << i+1 << "): ";
		cin >> X[i];
		cout << "Vuvedete Y(" << i+1 << "): ";
		cin >> Y[i];
	}
	cout << endl << "Izchislyavane priblijenata stoinost na funkciata." << endl;
	cout << "Vuvedete stoinost na X: ";
	cin >> stoinost;
	for(int i = 0; i < N; i++) {
		PR = 1;
		for(int j = 0; j < N; j++) {
			if(i != j) {
				PR = PR * ((stoinost - X[j]) / (X[i] - X[j]));
			}
		}

		ln = ln + Y[i] * PR;
	}

	cout << endl << "Rezultat = " << ln << endl;
}
```

2.3. Крайни разлики. Интерполационна формула на Нютон

Други две представяния на интерполационния полином носят името на големия английски математик Нютон (1643- 1727). В тези представяния се използва математическото понятие крайни разлики.

Нека е дадена произволна числова редица

(6) $y_0, y_1, \ldots, y_i, \ldots$

разлките между всеки два члена се нарича крайна разлика от първи ред на редицата и се бележат по следния начин

$\Delta y_0 = y_1 - y_2$

$\Delta y_1 = y_2 - y_3$

- - - - - - - -

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

- - - - - - - -

Аналогично разликите във всеки два съседни члена на редицата от крайните разлики от първи ред

$\Delta y_0 , \Delta y_1, \Delta y_2, \ldots, \Delta y_i , \ldots..$

Се наричат крайни разлики от втори ред на редицата (6)и се отбелязват по следния начин:

$\Delta^2 y_0 = \Delta y_1 - \Delta y_2$

$\Delta^2 y_1 = \Delta y_2 - \Delta y_1$

- - - - - - - - - - -

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

Изобщо крайна разлика от *к*-ти ред на числовата редица (6) се наричат разликите между всеки два съседни члена на редицата от крайните разлики от $(к - 1)$- ви ред:

$\Delta^к y_0 = \Delta^{к-1} y_1 - \Delta^{к-1} y_0$

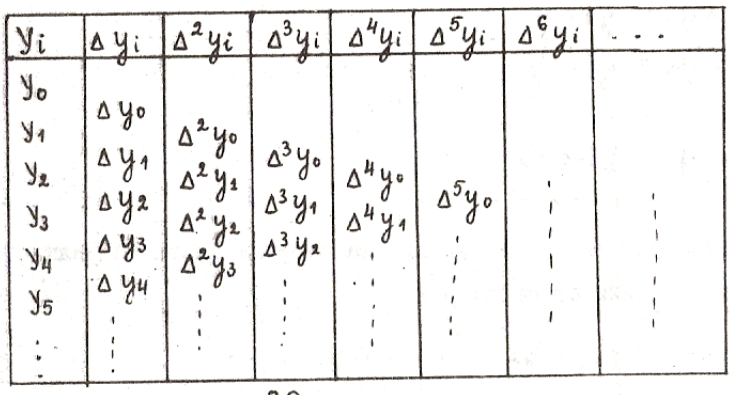
$\Delta^к y_1 = \Delta^{к-1} y_2 - \Delta^{к-1} y_1$

- - - - - - - - - - - - - -

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

- - - - - - - - - - - - -

Крайните разлики е удобно да се подредят в таблица, както това е направено на фиг. 30



Фиг.30

От таблицата се вижда, че ако редицата е крайна и има *к + 1* члена, могат да се намерят крайни разлики най- много от *к*- ти ред.

Пример. Да се направи таблица на крайните разлики на редицата 8, 16, 7, 0, -2.

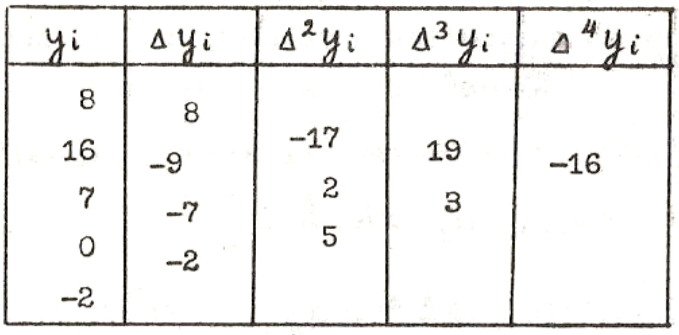
$У_0 = 8$, $y_1 = 16$, $y_2 = 7$, $y_3 = 0$, $y_4 = -2$.

За крайни разлики от първи ред се получава: $\Delta y_0 = 16 - 8 = 8$; $\Delta y_1 = 7- 16$; $\Delta y_2 = 0 - 7 = -7$; $\Delta y_3 = -2 - 0 = -2$. Аналогично се намират разликите от втори ред: $\Delta^2 y_0 = - 9- 8 = -17$; $\Delta^2 y_1 = - 7 - (-9) = 2$; $\Delta^2 y_2 = -2 - (-7) = 5$. Разликите от трети ред са: $\Delta^3 y_0 = 2 -(-17) = 19$; $\Delta^3 y_1 = 5 - 2 = 3$;

И от четвърти ред :

$\Delta^4 y_0 = 3 - 19 = -16$.

Крайните разлики са подредени в таблица на фиг. 31



Фиг. 31

Крайните разлики притежават редица интересни свойства, които тук няма да бъдат разгледани.

Интерполационните формули на Нютон са валидни за функции, зададени таблично, като интерполационните възли са равно отдалечени със стъпка h:

$X_0, x_1 = x_0 + h, x_2 = x_0 + 2h, \ldots, x_n = x_0 + nh.$

Интерполационния полином на Нютон се построява във вида:

$Hn\ (x) = a_0 + a_1 (x - x_0) + a_2 (x - x_0)(x - x_1) + \ldots + a_n (x - x_0)(x - x_1) \ldots (x - x_n)$

Изисква се (7) $Hn\ (x_i) = y_i,\ i = 0, 1, 2, \ldots, n.$ Трябва да се намерят коефициентите $a_0, a_1, \ldots, a_n$, така, че да се изпълнят условията (7). За целта $Hn\ (x)$ аргументът x се замества с $x_0, x_1, \ldots, x_n$ и отчитайки (7), се определят $a_0, a_1, \ldots, a_n.$

При $x = x_0$ се получава:

$Hn\ (x_0) = a_0 = y_0$

Следователно $a_0 = y_0$

При $x = x_1$: $Hn\ (x_1) = a_0 + a_1(x - x_0) = y_0 + a_1 h = y_1,$

Откъдето $a_1 = \frac{y_1 - y_0}{h} = \frac{\Delta y_0}{h}$

При $x = x_2$: $Hn\ (x_2) = a_0 + a_1(x - x_0) + a_2 (x_2 - x_0)(x_2 - x_1) = y_0 + \frac{y_1 - y_0}{h}.2h$

$+ a_2.2h.h = y_0 + 2(y_1 - y_0) + a_2.2h^2 = a_2$

Следователно:

$a_2 = \frac{y_2 - y_0 - 2(y_1 - y_0)}{2h^2} = \frac{y_2 - 2y_1 - y_0}{2h^2} = \frac{(y_2 - y_1) - (y_1 - y_0)}{2h^2} = \frac{\Delta y_1 - \Delta y_0}{2h^2} = \frac{\Delta^2 y_0}{2!\, h^2}$

аналогично се получават и останалите коефициенти:

$a_3 = \frac{\Delta^3 y_0}{3!\, h^3}$

$a_4 = \frac{\Delta^4 y_0}{4!\, h^4}$

.

.

$a_n = \frac{\Delta^n y_0}{n!\, h^n}$

след заместване на получените коефициенти в $Hn\ (x)$ се получава:

$Hn\ (x) = y_0 + \frac{\Delta y_0}{h}(x - x_0) + + \frac{\Delta^2 y_0}{2!\, h^2}(x - x_0)(x - x_1) + \ldots + \frac{\Delta^k y_0}{k!\, h^k}(x - x_0)\ldots(x - x_{k-1})$

$+ \ldots + \frac{\Delta^n y_0}{n!\, h^n}(x - x_0)\ldots(x - x_{n-1}).$

Формула (8) се нарича се нарича първа интерполационна формула на Нютон или още формула на Нютон за интерполиране напред. В нея участват крайните разлики на първия възел: $\Delta y_0, \Delta^2 y_0, \ldots, \Delta^n y_0$, Първата формула на Нютон се използва обикновено при интерполиране в точки, близки до началния възел x_0.

Следната програма на езика C++ служи за интерполиране на функции по първата формула на Нютон.

```
 #include <iostream>
using namespace std;
int main(int argc, char **argv)
{
   int N;
   double stoinost;
   double H, HN, T;
   cout << "Priblijavane na funkcii po purva formula na Nyuton." << endl << endl;
   do {
      cout << "Vuvedete N: ";
      cin >> N;
   } while(N < 1);
   int X[N], Y[N], KR[N];
   for(int i = 0; i < N; i++) {
      cout << "Vuvedete X(" << i+1 << "): ";
      cin >> X[i];
      cout << "Vuvedete Y(" << i+1 << "): ";
      cin >> Y[i];
   }
   cout << endl << "Izchislyavane na neobhodimite kraini razliki." << endl;
   for(int i = 0; i < N; i++) {
      KR[i] = Y[0];
      for(int j = 0; j < N - i - 1; j++) {
         Y[j] = Y[j+1] - Y[j];
      }
   }
   cout << endl << "Izchislyavane na priblijenata stoinost na funkciata." << endl;
   cout << "Vuvedete stoinost na X: ";
   cin >> stoinost;
```

```
    H = X[1] - X[0];
    HN = KR[0];
    T = 1;
    for(int i = 1; i < N; i++) {
       T = T * (stoinost - X[i - 1]) / (H * i);
       HN = HN + KR[i] * T;
    }
    cout << endl << "Priblijena stoinost = " << HN << endl;
}
```

За интерполиране на функции с равно отдалечени възли в точки, близки до най-големия възел X_n се използва втора интерполационна формула на Нютон:

$$Nn(x) = y_n + \frac{\Delta y_{n-1}}{h}(x - x_n) + \frac{\Delta^2 y_{n-2}}{2!h^2}(x - x_n)(x - x_{n-1}) + \frac{\Delta^3 y_{n-3}}{3!h^3}(x - x_n)(x - x_{n-1})(x - x_{n-2}) + \ldots + \frac{\Delta^n y_0}{n!h^n}(x - x_n)(x - x_{n-1})\ldots(x - x_1).$$

Във формулата участват крайните разлики:

$\Delta y_{n-1}, \Delta^2 y_{n-2}, \ldots, \Delta^n y_0$, на функцията $y = f(x)$. В таблицата на крайните разлики те са разположени по диагонала отдолу нагоре.

2.4. Понятие за обратна интерполация и екстраполация

С помощта на интерполационните формули на Нютон и Лагранж може да се намери приближената стойност на функция зададена таблично за стойност на аргумента отсъстваща от таблицата. Често в практиката възниква обратната задача – по дадена стойност $\bar{y}$ на функцията $f(x)$ да се намери стойността $\bar{x}$ на аргумента x, за която:

$$\bar{y} = f(\bar{x})$$

Един метод за решаване на тази задача е използването на интерполацията, но понеже се решава обратна задача, той носи името обратна интерполация. Обратната интерполация се осъществява по два различни начина в зависимост от това дали функцията $f(x)$ е строго монотонна (растяща или

намаляваща), на всяка стойност y отговаря точно една стойност на x. Може да се разгледа функцията $x = \varphi(y)$ зададена чрез таблицата:

y_i	y_0	y_1	…	y_n
x_i	x_0	x_1	…	x_n

За нея построяваме интерполационен полином, по някоя от известните вече формули, например:

$$Ln(y) = x_0 \frac{(y-y_1)\dots(y-y_n)}{(y_0-y)\dots(y_0-y)} + \dots + x_n \frac{(y-y_0)\dots(y-y_{n-1})}{(y_n-y_0)\dots(y_n-y_{n-1})}$$

Заместваме y с $\bar{y}$ получаваме приближената стойност на x. Когато функцията не е строго монотонна, описаният по-горе начин е неприложим. Ето защо се построява интерполационен полином за функцията $y = f(x)$ по някоя от формулите, например на Лагранж:

$$Ln(y) = x_0 \frac{(x-x_1)\dots(x-x_n}{(x_0-x)\dots(x_0-x)} + \dots + y_n \frac{(x-x_0)\dots(x-x_{n-1})}{(x_n-x_0)\dots(x_n-x_{n-1})}$$

В нея x се замества с $\bar{x}$, а $Ln(\bar{x})$ с $\bar{y}$. Получава се уравнение от степен n относно неизвестното $\bar{x}$. Самият процес за намиране стойността на интерполациония полином $Pn(\bar{x})$ се нарича различно в зависимост от местоположението на $\bar{x}$ спрямо интерполационните възли. Ако $\bar{x}$ е между най-левия и най- десния възел се казва, че функцията се интерполира, а ако $\bar{x}$ е извън тях, функцията се екстраполира. Например, ако възлите $x_0, x_1, \dots, x_n$ и числата a и $в$ са подредени по големина:

$a < x_0 < x_1 < \dots < x_n < в$, то при $\bar{x} \in (x_0, x_n)$ приближаването се нарича интерполация, а при $\bar{x} \in [a, x_0)$ или $\bar{x} \in (x_n, в)$ – екстраполация.

Упражнения

1. Постройте непосредствено интерполациония полином на функциите зададени с таблиците

а/

x_i	1	2	4
y_i	12	8	24

в/

x_i	0	2	3	5
y_i	1	3	2	5

2. Да се построи интерполационния полином на Лагранж за функциите зададени със следните таблици:

а/

x_i	-3	-2	-1	1
y_i	6	4	2	14

б/

x_i	0	1	2	3
y_i	1	2	4	8

3. По стойностите на функцията $y = f(x)$ зададена с таблицата:

x_i	11	13	14	18	19	21
y_i	1342	2210	2758	5850	6878	9282

Да се намери f(13,5).

4. Да се построи интерполациония полином на Нютон за следните функции:

а/

x_i	-2	-1	0	1
y_i	12	3	-1	5

б/

x_i	-3	-2	-1	0	1	2
y_i	-27	-16	-5	7	15	21

5. Да се състави програма на езика C++ за приближаване на функции по втората формула на Нютон – формула (9).
6. Електропроводимостта K на стъклото в зависимост от температурата t е измерена при следните температури

t	$-14,5^0$	$30,0^0$	$64,5^0$	$74,5^0$
k	0	0,004	0,018	0,029

Да се намери електропроводимостта на стъкло при $t_1 = 20^0$C и при $t_2 = 40^0$C.

7. Количеството Q от дадено вещество (в %), което остава в системата t минути след започването на химическата реакция се дава със следната таблица:

t	7	12	17	22	27
Q	83,7	72,9	63,2	54,7	47,5

Да се намери количеството на веществото в системата 15 минути след започването на химическата реакция и след 20 минути.

8. В следващата таблица е дадено налягането P на наситена пара в кг./см^2, съответстващо на относителен обем V в м^3/кг.:
9.

V	0,4323	0,2646	0,1699	0,1146
P	4,247	7,164	11,48	17,60

Какво е налягането на парата при относителен обем V =0,5м^3/кг. и V= 0,1м^3/кг

3. Табулиране в C++

3.1. Табулиране на функции

Една от най-често срещаните изчислителни задачи е задачата за пресмятане стойността на функция. Обикновено аналитичния вид на функцията е неизвестен или пък е сложен и неудобен за пресмятане. Затова се построява функция, приближена на дадената, с по- прост аналитичен вид. Времето за изчисляване на произволна стойност на функция може да се съкрати още повече, ако се пресметнат и оформят във вида на таблица, достатъчно много стойности на приближената таблица.

Съставянето на таблицата, от която се намират стойностите на дадена функция, се нарича табулиране на функция. Добре известни са таблиците на логаритмичната функция и таблиците на тригонометричните функции.

При табулиране на дадена функция y = f (x) се съставя таблица от вида:

X	y = f(x)
x_0	y_0
x_1	y_1
.	.
.	.
X_n	y_n

Най- често точките $x_0 < x_1 < \ldots < x_n$ се равностоящи. Тогава $x_1 - x_0 = x_2 - x_1 = \ldots = x_n - x_{n-1} = h$. положителната разлика h се нарича стъпка на таблицата и се избира да бъде 0.1, 0.01, 0.001 и т.н. Колкото по- малка е стъпката h, толкова по-голяма е точността на таблицата.

4. Двумерни масиви и работа с тях

4.1. Операции с матрици

Матрица е всяка правоъгълна таблица от числа. Например

$$A=\begin{pmatrix} a_{11} & a_{12} \dots & a_{1n} \\ a_{21} & a_{22} \dots & a_{2n} \\ a_{m1} & a_{m2} & a_{mn} \end{pmatrix}$$

Използват се следните кратки означения: A= (a_{ij}) или A= $(a_{ij})_{i=1}^{m}$, j^n = 1. При второ означение се показват броят на матрицата – m , и броя стълбове- n. Когато m= n матрицата се нарича квадратна от ред n, в противен случай – правоъгълна от вид *m x n.*

Две матрици са равни,ако са от един и същ вид и съответните им елементи са равни .

С матрици могат да се извършват следните основни операции: събиране на матрици, изваждане на матрици, умножение на матрица с число и умножение на матрици.

За да се съберат (извадят) две матрици, трябва те да са от един и същи вид, т.е да имат равен брой редове и стълбове. Тогава сумата (разликата) на две матрици е нова матрица от същия вид, всеки елемент на която е равен на сумата(разликата) на съответните елементи на дадените матрици.

Ако дадените матрици а означени с A и B, а получената с C, то $c_{ij} = a_{ij} \pm b_{ij}$ за всяко i = $\overline{1,m}$ и i = $\overline{1,n}$ където m е броя на редовете , а n е броя на стълбовете.

Например, нека

$$A=\begin{pmatrix} 1 & 5 & 7 & 4 \\ 2 & 0 & -3 & 6 \\ 1 & 0 & 4 & -2 \end{pmatrix} \quad \text{и} \quad B=\begin{pmatrix} 7 & -2 & 6 & 9 \\ -7 & 8 & 4 & -1 \\ 0 & 0 & 2 & 1 \end{pmatrix}$$

$$\text{Тогава } C = A+B = \begin{pmatrix} 8 & 3 & 13 & 13 \\ -5 & 8 & 1 & 5 \\ 1 & 0 & 6 & -1 \end{pmatrix}$$

Следва програма за изваждане на две матрици, която съдържа два вложени цикъла. В C++, както и в другите алгоритмични езици, матриците се представят чрез двумерни масиви.

```
#include <iostream>
using namespace std;
int main()
{
	int cols, rows;
	do {
		cout << "Vuvedte broya na redovete: "; cin >> rows;
	} while(rows < 1);
	do {
		cout << "Vuvedte broya na kolonite: "; cin >> cols;
	} while(cols < 1);

	/*	Dvumerni masivi
		Matrica - naprimer 4x4
	*/
	int a[rows][cols], b[rows][cols], c[rows][cols];
	for(int i = 0; i < rows; i++) {
		for(int j = 0; j < cols; j++) {
			cout << "Vivedte A[" << i+1 << ", " << j+1 << "]: ";
			cin >> a[i][j];
		}
	}
	for(int i = 0; i < rows; i++) {
		for(int j = 0; j < cols; j++) {
			cout << "Vivedte B[" << i+1 << ", " << j+1 << "]: ";
			cin >> b[i][j];
		}
	}
```

```
        for(int i = 0; i < rows; i++) {
                for(int j = 0; j < cols; j++) {
                        c[i][j] = a[i][j] - b[i][j];
                        cout << "C[" << i+1 << ", " << j+1 << "] = " << c[i][j];
                }
        }
        cout << endl << "Rezultat" << endl;
        cout << "Matrica A:" << endl;
        for(int i = 0; i < rows; i++) {
                for(int j = 0; j < cols; j++) {
                        cout << a[i][j] << " ";
                }
                cout << endl;
        }
        cout << endl << "Matrica B:" << endl;
        for(int i = 0; i < rows; i++) {
                for(int j = 0; j < cols; j++) {
                        cout << b[i][j] << " ";
                }
                cout << endl;
        }
        cout << endl << "Matrica C:" << endl;
        for(int i = 0; i < rows; i++) {
                for(int j = 0; j < cols; j++) {
                        cout << c[i][j] << " ";
                }
                cout << endl;
        }
}
```

С незначителни изменения тази подпрограма може да се използва и за събиране на матрици и за умножение на матрица с число. Произведение на матрица (A) с число (x) е матрица от същия вид D , всеки елемент на която е произведение на съответния елемент от дадената матрица с число, т.е $d_{ij} = a_{ij} . x$.

Операцията събиране и изваждане на матрици и произведение на матрици с число се извършват по елементно, като резултатната матрица е от същия вид както и операндите. Но това правило не е вярно за умножение на две матрици.

За да се умножат две матрици $A_{m\ x\ n}$ и $B_{p\ x\ q}$ е необходимо броят на стълбовете на първата матрица да е равен а на броя на редовете на втората т.е n= q. Тогава произведението на двете матрици е нова матрица C с m реда и p стълба, (i j) итият елемент на която се получава чрез скаларно произведение на i-тия ред на матрицата A с j- тия ред на матрицата B, т.е. елементите на матрицата $C_{m\,x\,p} = A_{m\,x\,n} . B_{n\,x\,p}$ се получава по формулите:

$C_{ij} = a_{i1}.b_{1j} + a_{i2}.b_{2j} + \ldots + a_{j\,n}.b_{n\,j}$, където $i = \overline{1,m}$, $j = \overline{1,p}$.

Например , нека

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \qquad B = \begin{pmatrix} 2 & 0 \\ 3 & -1 \end{pmatrix}$$

Тогава

$$AB = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} . \begin{pmatrix} 2 & 0 \\ 3 & -1 \end{pmatrix} = \begin{pmatrix} 8 & -2 \\ 18 & -4 \end{pmatrix};$$

$$BA = \begin{pmatrix} 2 & 0 \\ 3 & -1 \end{pmatrix} . \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} = \begin{pmatrix} 2 & 4 \\ 0 & 2 \end{pmatrix},$$

т.е $AB \neq BA$. Този резултат показва, че умножението на матрици няма всички свойства на умножението с числа.

След подпрограмата за умножение на две матрици $A_{m\,x\,n}$ и $B_{p\,x\,q}$. В нея са използвани три вложени цикъла.

```
#include <iostream>
#include <cstdlib>
using namespace std;
 int main(int argc, char *argv[])
{
```

```
//брой редове в matrA
int xA=2;
//брой колони в matrA
int yA=4;
//указател към указател
//представяме масивът като едномерен масив от указатели към
//едномерни динамични масиви
int **matrA;
//резервираме памет, ако не става излизаме
if(!(matrA=new int *[xA])) exit(0);
//На всеки елемент на горният масив се присвоява едномерен динамичен масив
for(int i=0;i<xA;i++)
   if(!(matrA[i]=new int[yA]))exit(0);
//Вторият множител.Аналогично като за А
int xB=4;
int yB=3;
//За да бъде възможно умножението трябва броят на
//колоните на А да е равен на броя на редовете на В
if(yA!=xB)
   cout<<"Error!";
int **matrB;
if(!(matrB=new int *[xB]))exit(0);
for(int i=0;i<xB;i++)
   if(!(matrB[i]=new int[yB]))exit(0);
//Резултатът е масив с рамерност брой на редове=бр.ред. на А и
//бр.кол.=бр.кол. на В
int xR=xA;
int yR=yB;
int **matrRez;
if(!(matrRez=new int *[xR]))exit(0);
```

```
for(int i=0;i<xR;i++)
   if(!(matrRez[i]=new int[yR]))exit(0);
//Инициализираме масивът с резултата
for(int i=0;i<xR;i++)
   for(int j=0;j<yR;j++)
      matrRez[i][j]=0;
//Въвеждане на масива A
int imp;
for(int i=0;i<xA;i++)
   for(int j=0;j<yA;j++)
   {
      cin>>imp;
      matrA[i][j]=imp;
   }
//Въвеждане на масива B
for(int i=0;i<xB;i++)
   for(int j=0;j<yB;j++)
   {
      cin>>imp;
      matrB[i][j]=imp;
   }
//Фактическото умножение на матриците
for(int i=0;i<xR;i++)
   for(int j=0;j<yR;j++)
      for(int l=0;l<yA;l++)
       matrRez[i][j]+=matrA[i][l]*matrB[l][j];
//Извеждане на matrA
for(int i=0;i<xA;i++)
{
   for(int j=0;j<yA;j++)
      cout<<matrA[i][j]<<",";
```

```
    cout<<'n';
}
    cout<<'n';
//Извеждане на matrB
for(int i=0;i<xB;i++)
{
    for(int j=0;j<yB;j++)
       cout<<matrB[i][j]<<",";
    cout<<'n';
}
    cout<<'n';
//Извеждане на резултата
for(int i=0;i<xR;i++)
{
    for(int j=0;j<yR;j++)
       cout<<matrRez[i][j]<<",";
    cout<<'n';
}
//унищожаваме елементите на matrA
for(int i=0;i<xA;i++)
  delete [] matrA[i];
//унищожаваме matrA
delete matrA;
for(int i=0;i<xB;i++)
  delete [] matrB[i];
delete matrB;
for(int i=0;i<xR;i++)
  delete [] matrRez[i];
delete matrRez;
   return 0;
}
```

4.2. Линеаризация на двумерни масиви

Определение: Масив е наредена последователност от елементи от един и същи базов тип. Определен елемент на масива се указва посредством името на целия масив, последвано от поредния номер (индекс), който ще се запознаем при овладяване на обектноориентираното програмиране.

Когато базовият тип е прост (скаларен), той се нарича едномерен масив (вектор). Двумерен масив (матрица) ще наричаме масив, чийто базов тип е едномерен масив. По същата логика N-мерен масив в C++ ще наричаме масив, чийто базов тип е (N-1)-мерен масив.

За улеснение структуриран базов тип може да се дефинира посредством **typedef** преди декларирането на масива.

В следващия пример, базовият тип е структуриран – едномерен масив:

typedef double vector [20];

vectori matrix [256];

индексите на matrix са от 0 до 255, а всеки елемент на matrix е едномерен масив. Образува се масив от 256 вектора с елементи от тип double (матрица от тип double с 256 реда и 20 стълба). Същата матрица би могла да се дефинира по седният начин:

double matrix1 [256] [20];

двете дефиниции са еквивалентни и за масива се отделя едно и също количество памет.

В езика C++ двумерните масиви се представят в оперативната памет на компютъра като едномерни. Елементите им се разполагат последователно по стълбове.

По- общо, ако масивът A е оразмерен *(A:[M][N]),* то поредният номер на елемента A([I] [K]) се изчислява по формулата *K*(M +1) +I+1.*

В редица линеаризация на двумерни масиви се спестява оперативна памет или се увеличава бързодействието на програмата. Например, при умножение на матрица $A_{m \times n}$ със симетрична матрица $B_{p \times q}$, т.е матрицата $b_{ij} = b_{ji}$, i, j = $\overline{1,k}$, може втората матрица да се представи чрез едномерен масив, съдържащ елементите над и върху главния диагонал. По този

начин се спестява оперативна памет, тъй като вместо двумерен масив с [к + 1] [к + 1] елемента се използва едномерен масив с $\frac{(к+1),(к+1)}{2}$ елемента. Следващата подпрограма реализира това умножение. В нея се използва работен масив с к+1 елемента, в който се охранява текущия стълб на B. С цел по-малко да се изменя работния масив, елементите на търсената матрица се получават по стълбове.

```
#include <iostream>
using namespace std;
int main()
{
	int m, n, k;
	/* Втвеждане на размерностите на масивите */
	cout << "Vuvedete broya na redovete na masiva A: ";
	cin >> m;
	cout << "Vuvedete broya na kolonite na masiva A: ";
	cin >> n;
	cout << "Vuvedete razmernostta na masiva B: ";
	cin >> k;
	int A[m][n];
	int B[k][k];
	int C[k+1];
	int R[k+1];
	/* Въвеждане на елементите на масивите */
	for(int i = 0; i < m; i++) {
		for(int j = 0; j < n; j++) {
			cout << "Vuvedete A[" << i+1 << ", " << j+1 << "]: ";
			cin >> A[i][j];
		}
	}
	cout << endl;
	for(int i = 0; i < k; i++) {
```

```
        for(int j = 0; j < k; j++) {
                cout << "Vuvedete B[" << i+1 << ", " << j+1 << "]: ";
                cin >> B[i][j];
        }
}
/* Обработка на масива B */
cout << endl;
for(int j = 0; j < k; j++) {
        int RO = j * (j - 1) / 2;
        /* Присвояване на колоната */
        cout << "Populvane na raboten masiv" << endl;
        for(int i = 0; i < j; i++) {
                C[i] = B[][k];
        }
        if(j < k - 1) {
                /* RO = RO + j */
                RO += j;
                for(int i = j+1; i < k; i++) {
                        RO += i - 1;
                        R[i] = B[RO];
                }
        }
        cout << "Умножение" << endl;
        for(int t = 0; t < m; t++) {
                int s = 0;
                for(int l = 0; l  < k; l ++) {
                        s = s + A[t][l] * R[l];
                }
                C[t][j] = s;
        }
        /* Извеждане на C */
```

```
            for(int i = 0; i < k+1; i++) {
                cout << C[i];
            }
        }
        return 0;
    }
```

Упражнения

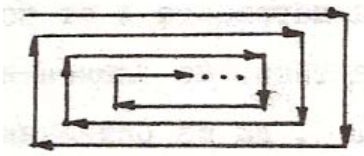
1.Ако A= $\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$, E= $\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$, намерете матрицата X, за която AX = E.

2. Пресметнете матрицата B= A^2 - 5A + 6E, където A и E са дефинирани в задача

3. решете задача 2 за произволна матрица A, като използвате подпрограма за операции с матрици.

4. Линеанизирайте матрицата $A_{m x n}$ по спиралата , както е показано на фиг.32

.



фиг.32

5. Изведете формула за поредния номер на произволен елемент от тримерен масив T, оразмерен чрез *DIM T(M, N, L).*

6. Съставете програма , която умножава матрица $A_{m x n}$ със симетрична матрица B, зададена като едномерен масив, без да се използва работен масив.

5. Методи за решаване на системи

В този параграф се разглеждат два основни вида методи за решаване на линейни системи с n уравнения и n неизвестни – точни и итерационни.

При точните методи се предполага, че ако данните са въведени точно и всички междинни пресмятания (те са краен брой) се извършват точно, то ще се получи точно решение на задачата, ако те има такова.

При интерационните методи решението на задачата се получава приближено като граница на безкрайна редица от последователни приближения, които се извършват еднообразен начин.

Навсякъде се предполага, че коефициентите пред неизвестните образуват матрица А от вида n x n, свободните членове се означават чрез $в_i$, а неизвестните – чрез x_i , i = $\overline{1,n}$.

5.1. Метод на Гаус за последователно изключване на неизвестните

Исторически това е първия метод за решаване на системи линейни уравнения. Извършва се на два етапа, наречени прав и обратен ход. При правия ход матрицата А от коефициентите пред неизвестните се преобразува така, че елементите по главния диагонал, т.е. елементите $a_{i\,j}$, да са различни от нула, а елементите под диагонала, т.е a_{ij}, i > j, да са нули. Прието е това преобразование да се нарича преобразуване на матрица А в триъгълен вид. При обратния ход, ако системата има решение и не е преопределена, се изчислява стойността на последното неизвестно, тази стойност се замества в предпоследното уравнение, от него се изчислява предпоследното неизвестно и т.н докато се получи стойността на x_1.

По- подробно метода на Гаус се прилага по следния начин.

I. Прав ход

За всеки ред от първия до предпосления се извършват следните преобразувания;

А. ако $a_{ii} = 0$, се преминава към Б, в противен случай се преминава към Г.

Б. Ако всички елементи a_{ki} , $k > i$ са нули, то системата е предопределена – край.

В. Ако съществува елемент $a_{ki} \neq 0$, $k > i$, то разменят се местата на i - тия и к-тия ред.

Г. Изчислява се $t_i = \frac{1}{a_{ii}}$

Д. Преобразуват се елементите от i- тия ред: $a_{ij} = 1$,

$a_{ij} = a_{ij}.\ t_i$, $b_i = b_i.\ t_i$, $i < j$.

Е. Преобразуват се елементите на останалите редове:

$d_k = a_{ki}$, $a_{ki} = 0$, $a_{kj} = a_{kj} - d_k\, a_{ij}$, $b_k = b_k - d_k\, b_i$, $k > i$

II. Обратен ход (в случай, че системата има решение):

А. Изчислява се $x_n = b_n.\ t_n$.

Б. За всяко i = $\overline{1, n-1}$, намерените неизвестни x_j, j > i се заместват в i-тото уравнение и от там се изчислява x_j.

Пример. Да се реши системата

$$\begin{cases} X_1 + 2X_2 + X_3 = 3 \\ X_1 + 4X_2 + X_3 = 4 \\ 4X_1 + X_2 + X_3 = 4 \end{cases}$$

I. Прав ход

Записва се матрицата А от коефициентите пред неизвестните, разширена със стълба от свободните членове:

$$A = \left(\begin{array}{ccc|c} 1 & 2 & 1 & 3 \\ 1 & 4 & 1 & 4 \\ 4 & 1 & 1 & 4 \end{array}\right).$$

Тъй като $a_{11} = 1$, то елементите от първия ред се запазват, а елементите от останалите редове се преобразуват съгласно т.е. получава се:

$$\left(\begin{array}{ccc|c} 1 & 2 & 1 & 3 \\ 0 & 2 & 0 & 1 \\ 0 & -7 & -3 & -8 \end{array}\right)$$

Сега $a_{22} = 2 \neq 0$ и първо се умножават елементите на втория ред с 1/2, а след това елементите от третия ред се преобразуват съгласно т.е. Получава се:

$$\left(\begin{array}{ccc|c} 1 & 2 & 1 & 3 \\ 0 & 1 & 0 & 0,5 \\ 0 & 0 & -3 & -4,5 \end{array}\right)$$

Така дадената система е еквивалентна на системата:

$$\begin{cases} X_1 + 2X_2 + X_3 = 3 \\ X_2 = 0,5 \\ -X = -4,5 \end{cases}$$

Накрая се прилага обратния ход и се получава последователно $x_3 = 1,5$, $x_2 = 0.5$, $x_1 = 0,5$.

Следва програма, чрез която се решават произволни системи от n-уравнения с n- неизвестни $n \leq 30$.

```
#include <iostream>
using namespace std;
int main()
{
	cout << "Programa za reshavane na sistema" << endl;
	cout << "ot lineyni urawneniya s N neizvestni" << endl;
	cout << "\t Izpolzva se metoda na Gaus" << endl;
	int A[30][30], B[30], X[30];
	int N;
	/* Vuvejdane na n Dokato ne stane m/u 1 i 30 */
	do {
		cout << "Vuvedete N:";
		cin >> N;
		if(N < 1 || N > 30) {
			cout << "Greshno vuvedeno chislo" << endl;
		}
	} while(N < 1 || N > 30);
```

```
/* Zapylvane na A I B */
for(int i = 0; i < N; i++) {
	for(int j = 0; j < N; j++) {
		cout << "Vivedte A[" << j+1 << ", " << i+1 << "]: ";
		cin >> A[j][i];
		cout << "Vuvedete svobodnia chlen";
		cin >> B[j];
	}
}
/* PRAV HOD */
for(int i = 0; i < N - 1; i++) {
	/* Diagonal */
	/*
		* 0 0 0
		0 * 0 0
		0 0 * 0
		0 0 0 * # ne se gleda poslednia red
	*/
	int t = 1;
	/* Izbor na vodesht element */
	if(A[i][i] == 0) {
		for(int k = i + 1; k < N; k++) {
			if(A[k][i] != 0) {
				// Nyakakvo dvuetochie. red 540
				t = 1;
			}
		}
		// red 560
	}
	// red 150. Ne e znam ot kude idva
	t  = 1;
```

```
        for(int j = i = 1; j < N; j++) {
                /* Preobrazuvane na I-ti element */
                A[i][j] = A[i][j] * t;
        }

        B[i] = B[i] * t;
        /* Preobrazuvane na ostanalite redove */
        for(int k = i + 1; k < N; k++) {
                // red 210
                int d = 0;
                for(int j = i + 1; j < N; j++) {
                        A[k][j] = A[k][j] - d * B[i];
                        B[k] = B[k] - d * B[i];
                }
        }
}
if(A[N][N] == 0) {
        cout << "Sistemata e preopredelena" << endl;
        return 1;
}
        B[N] = B[N] * 1;
/* KRAY NA PRAVIYA HOD */

/* OBRATEN HOD */
X[N] = B[N];
for(int i = N - 2; i = 0; i--) {
        int XS = B[i];
        for(int j = i + 1; j < N; j++) {
                XS = XS - A[i][j] * X[j];
        }
        /*  Pak dvuetochie. red 360 */
```

```
            X[i] = X[i] * 1;
        }
        /* KRAY NA OBRATNIA HOD */

        cout << "Reshenie na sistemata:" << endl;
        for(int i = 0; i < N; i++) {
            cout << "X[" << i+1 << "] = " << X[i];
        }
        return 0;
}
```

5.2. Метод на Гаус- Жордан

Обратният ход от метода на Гаус може да се избегне, ако при правия ход матрицата от коефициентите пред неизвестните се преобразува не в триъгълен, а в диагонален вид, т.е. ненулеви елементи да има само по главния диагонал. Затова е достатъчно само при правия ход от метода на Гаус преобразуванията от т.е да се приложат не само към редовете след i-тия, а също и за редовете след i- тия ред. Тази модификация на метода на Гаус се нарича метод на Гаус- Жордан.

Пример. Да се реши предната система по метода на Гаус- Жордан.

Както при метода на Гаус, след първата стъпка се получава матрицата

$$\left(\begin{array}{ccc|c} 1 & 2 & 1 & 3 \\ 0 & 2 & 0 & 1 \\ 0 & -7 & -3 & -8 \end{array}\right)$$

След това елементите от втория ред се умножават с 1/2, а елементите от първия и третия ред се преобразуват съгласно т.е

Получава се:

$$\left(\begin{array}{ccc|c} 1 & 0 & 1 & 2 \\ 0 & 1 & 0 & 0,5 \\ 0 & 0 & -3 & -4,5 \end{array}\right)$$

Накрая третия ред се умножава с -1/3, а втория ред се преобразуват съгласно т.е получава се:

$$\left(\begin{array}{ccc|c} 1 & 0 & 1 & 0,5 \\ 0 & 1 & 0 & 0,5 \\ 0 & 0 & 1 & 1,5 \end{array}\right)$$

Като неизвестните в системата са равни на свободните членове, т.е $x_1 = 0,5$, $x_2 = 0.5$, $x_3 = 1,5$.

По- долу е приложена модифицирана програма, реализираща метода на Гаус- Жордан.

```
#include <iostream>
using namespace std;
int main()
{
	cout << "Programa za reshavane na sistemi" << endl;
	cout << "lineini uravneniya s N neizvestni" << endl;
	cout << "Izpolzva se metoda na Gaus-Jordan" << endl;
	int A[30][30], B[30], X[30];
	int N;

	* Vuvejdane na n Dokato ne stane m/u 1 i 30 */
	do {
		cout << "Vuvedete N:";
		cin >> N;

		if(N < 1 || N > 30) {
			cout << "Greshno vuvedeno chislo" << endl;
		}
	} while(N < 1 || N > 30);
```

```
/* Zapulvane na A I B */
for(int i = 0; i < N; i++) {
	for(int j = 0; j < N; j++) {
		cout << "Vivedte A[" << j+1 << ", " << i+1 << "]: ";
		cin >> A[j][i];
		cout << "Vuvedete svobodnia chlen";
		cin >> B[j];
	}
}
int d = 1;
/* PRAV HOD */
for(int i = 0; i < N; i++) {
	/* Izbor na vodesht element */
	float t = 1;
	if(A[i][i] == 0) {
	}
	for(int j = i + 1; j < n; j ++) {
		A[i][j] = A[i][j] * t;
	}
	B[i] = B[i] * t;
	/* Preobrazuvane na ostanalite redove */
	for(k = 0; k < N; k++) {
		if(k < i) {
			// red 210
			d = 1;
			for(j = i + 1; j < N; j++) {
				A[k][j] = A[k][j] - d * A[i][j];
			}
			B[k] = B[k] - d * B[i];
		}
	}
```

```
        }


        if(A[N][N] == 0) {
                cout << "Sistemata e preopredelena" << endl;
                return 1;
        }
        B[N] = B[N] * 1;
        for(int k = 0; k < N-1; k++) {
                d = 1;
                B[k] = B[k] - d * B[N];
        }
        /* KRAY NA PRAVIYA HOD */
          cout << "Reshenie na sistemata:" << endl;
        for(int i = 0; i < N; i++) {
                cout << "X[" << i+1 << "] = " << B[i];
        }
        return 0;
}
```

5.3. Итерационни методи

Както беше споменато, при итерационните методи се построяват последователни приближения на неизвестните, които все повече се доближават до решението. В тази част ще бъдат разгледани два метода: метод на простата итерация и метод на Зайдел. И при двата най- напред линейната система с n уравнения и n неизвестни се подготвя за итерациите, като за всяко i= $\overline{1,n}$ i- тото уравнение се реши относно x_i. Така системата добива следния вид:

$$X_1 = 0.X_1 - \frac{a_{12}}{a_{11}} X_2 - \dots - \frac{a_{1n}}{a_{11}} X_n + \frac{b}{a_{11}}$$

$$X_2 = -\frac{a_{21}}{a_{22}} X_1 - 0.X_2 - \dots - \frac{a_{2n}}{a_{22}} X_n + \frac{b_2}{a_{22}}$$

- - - - - - - - - - - - - - - - - -

$$X_n = -\frac{a_{n1}}{a_{nn}} X_1 - \frac{a_{n2}}{a_{nn}} X_2 - \dots - 0\, X_n + \frac{b_n}{a_{nn}}$$

Това изразяване на неизвестните се приема за първо приближение на неизвестните. За начално приближение е избират произволни n числа, например нули и се записват в десните части на горните равенства.

При метода на простата итерация к- тото приближение се получава от (к - 1) - вото по формулите (с горен индекс е означен номерът на приближението).

$$X_1^{К} = 0.\, X_1^{(К-1)} - \frac{a_{12}}{a_{11}} X_2^{(К-1)} - \dots - \frac{a_{1n}}{a_{11}} X_n^{(K-1)} + \frac{b}{a_{11}}$$

$$X_2^{К} = -\frac{a_{21}}{a_{22}} X_1^{(К-1)} - 0.\, X_2^{(К-1)} - \dots - \frac{a_{2n}}{a_{22}} X_n^{(К-1)} + \frac{b_2}{a_{22}}$$

- - - - - - - - - - - - - - - - - -

$$X_1^{К} = -\frac{a_{n1}}{a_{nn}} X_1^{(К-1)} - \frac{a_{n2}}{a_{nn}} - X_2^{(К-1)} \dots - 0\, X_n^{(К-1)} + \frac{b_n}{a_{nn}}$$

Ако всички коефициенти пред неизвестните и свободните членове са зададени точно и всички междинни пресмятания се извършват с достатъчна точност, то метода на простата итерация, може да се получи решение на системата с произволна отнапред зададена точност. Казва се, че решението е намерено с точност ε, ако за някое к и за всяко i= $\overline{1,n}$, $\left|x_i^{(k)} - x_i^{(k-1)}\right| < \varepsilon$. За да се намери такова к след краен брой стъпки, достатъчно е диагоналният коефициент a_{ij} във всяко уравнение на системата да бъде по- голям по абсолютна стойност от сумата от абсолютните стойности на останалите коефициенти в уравнението, т.е за i= $\overline{1,n}$

$|a_{ii}| > |a_{i1}| + |a_{i2}| + \dots + |a_{ii-1}| + |a_{ii+1}| + \dots + |a_{in}|.$

Когато е изпълнено последното условие се казва, че матрицата A е с преобладаващ главен диагонал.

В програмата, реализираща метода на простата итерация, освен матрицата A и свободните членове, се въвеждат и точността, с която да се намери решението и максималния брой итерации, които да се извършат. Прави се проверка дали всеки диагонален елемент е преобладаващ по модул. Програмата завършва изпълнението си или когато е намерено решението, или когато се извършват максимално допустимия брой итерации.

```
#include <iostream>
#include <cmath>
using namespace std;
int main()
{
	cout << "Programa za reshavane na sistemi" << endl;
	cout << "lineini uravneniya s N neizvestni" << endl;
	cout << "po metoda na prostata iteracia" << endl;
	int A[30][30], B[30], X[30];
	int N, IMAX, S = 0, T;
	float EPS;
	/* Въвеждане на N докато не стане м/у 1 и 30 */
	do {
		cout << "Vuvedete N:";
		cin >> N;
		if(N < 1 || N > 30) {
			cout << "Greshno vuvedeno chislo" << endl;
		}
	} while(N < 1 || N > 30);
	do {
		cout << "Vuvedete tochnostta: "; cin >> EPS;
	} while(EPS < 0);
	do {
```

```
        cout << "Vuvuedete maksimalen broy iteracii: "; cin >> IMAX;
    } while(IMAX < 0);

    cout << endl;
    cout << "Koeficientite na neizvetnite tryabva da" << endl;
    cout << "Otgovaryat na slednoto iziskvane: " << endl;
    cout << "2*ABS(A[i][i]) > ABS(A[i][1]) + ... + ABS(A[i][N])" << endl;
    cout << "t.e. diagonalniya koeficient da preobladava po modul." << endl << endl;
    for(int i = 0; i < N; i++) {
        do {
            for(int j = 0; j < N; j++) {
                cout << "Vivedte A[" << i+1 << ", " << j+1 << "]: ";
                cin >> A[i][j];
                S = S + abs(A[i][j]);
            }
            T = abs(A[i][i]);
        } while ( T <= S-T);
        cout << "Vuvedete svobodniya chlen :"; cin >> B[i];
    }
    /* Карай на въвеждането */
    /* НАчало на присвояване */
    for(int i = 0; i < N; i++) {
        T = 1 / A[i][i];
        A[i][i] = 0;
        B[i] = B[i] * T;
        X[i] = B[i];
        for(int j = 0; j < N; j++) {
            A[i][j] = A[i][j] * T;
        }
    }
```

```
int L = 1;
bool FL = true;
do {
	L = IMAX + 1; // Брой итерации
	FL = true;   // Флаг за достигната точност
	int XN[N];

	/* Получаване на следваща итерациия */
	for(int i = 0; i < N; i++) {
		S = B[i];
		for(int j = 0; j < N; j++) {
			S = S - A[i][j] * X[j];
		}
		XN[i] = S;
		if(abs(S - X[i] > EPS) {
			FL = false;
			X[i] = S;
		}
	}
	for(int i = 0; i < N; i++) {
		X[i] = XN[i];
	}
	/* Проверка за край */
	if(L >= IMAX) {
		cout << endl << "Reshenieto ne se dostiga s " << L << " iteracii"
<< endl;
		return 1;
	}
} while(FL == false);
cout << endl << "Reshenito se dostiga sled " << L << " iteracii" << endl;
```

```
        cout << "Reshenieto e:" << endl;

        for(int i = 0; i < N; i++) {
                cout << "X[" << i+1 << "] = " << X[i] << endl;
        }
}
```

Методът на Зайдел представлява модификация на метода на простата итерация, при която в намирането на к- тото приближение на x_i участват вече намерените к-ти приближения на x_j за j < i. По- точно к-тото приближение се изчислява по следните формули:

$$X_1^{К} = 0.\ X_1^{(К-1)} - \frac{a_{12}}{a_{11}}\ X_2^{(К-1)} - \ldots - \frac{a_{1n}}{a_{11}}\ X_n^{(K-1)} + \frac{b_1}{a_{11}}$$

$$X_2^{К} = -\frac{a_{21}}{a_{22}}\ X_1^{(К-1)} - 0.\ X_2^{(К-1)} - \ldots - \frac{a_{2n}}{a_{22}}\ X_n^{(К-1)} + \frac{b_2}{a_{22}}$$

$$- - - - - - - - - - - - - - - - - -$$

$$X_1^{К} = -\frac{a_{n1}}{a_{nn}}\ X_1^{(К-1)} - \frac{a_{n2}}{a_{nn}} - X_2^{(К-1)} \ldots - 0\ X_n^{(К-1)} + \frac{b_n}{a_{nn}}$$

Достатъчно условие да се достигне решението след краен брой стъпки и при този метод е матрицата А да има преобладаващ по модул главен диагонал. В повечето случаи решението на метода на Зайдел се достига по- бързо (с по-малък брой итерации).

```
        #include <iostream>
        #include <cmath>
        using namespace std;
        int main()
        {
                        cout << "Programa za reshavane na sistemi" << endl;
                        cout << "lineini uravneniya s N neizvestni" << endl;
                        cout << "po metoda na prostata iteracia" << endl;
                        int A[30][30], B[30], X[30];
                        int N, IMAX, S = 0, T;
                        float EPS;
```

```
                    /* Въвеждане на N докато не стане м/у 1 и 30 */
                    do {
                        cout << "Vuvedete N:";
                        cin >> N;
                        if(N < 1 || N > 30) {
                            cout << "Greshno vuvedeno chislo" << endl;
                        }
                    } while(N < 1 || N > 30);
                    do {
                        cout << "Vuvedete tochnostta: "; cin >> EPS;
                    } while(EPS < 0);
                    do {
cout << "Vuvuedete maksimalen broy iteracii: "; cin >> IMAX;
                    } while(IMAX < 0);
                    cout << endl;
cout << "Koeficientite na neizvetnite tryabva da" << endl;
cout << "Otgovaryat na slednoto iziskvane: " << endl;
cout << "2*ABS(A[i][i]) > ABS(A[i][1]) + ... + ABS(A[i][N])" << endl;
cout << "t.e. diagonalniya koeficient da preobladava po modul." << endl <<
endl;

for(int i = 0; i < N; i++) {
                    do {
                        for(int j = 0; j < N; j++) {
                      cout << "Vivedte A[" << i+1 << ", " << j+1 << "]: ";
                                cin >> A[i][j];
                                S = S + abs(A[i][j]);
                            }
                            T = abs(A[i][i]);
                        } while ( T <= S-T);
                        cout << "Vuvedete svobodniya chlen :"; cin >> B[i];
```

```
}
/* Карай на въвеждането */

/* НАчало на присвояване */
for(int i = 0; i < N; i++) {
	T = 1 / A[i][i];
	A[i][i] = 0;
	B[i] = B[i] * T;
	X[i] = B[i];
	for(int j = 0; j < N; j++) {
		A[i][j] = A[i][j] * T;
	}
}
int L = 1;
bool FL = true;
do {
	L = IMAX + 1; // Брой итерации
	FL = true;   // Флаг за достигната точност
	int XN[N];
	/* Получаване на следваща итерациия */
	for(int i = 0; i < N; i++) {
		S = B[i];
		for(int j = 0; j < N; j++) {
			S = S - A[i][j] * X[j];
		}
		XN[i] = S;
		if(abs(XN[i]) - X[i] > EPS) {
			FL = false;
		}
	}
```

```
                    for(int i = 0; i < N; i++) {
                        X[i] = XN[i];
                    }
                    /* Проверка за край */
                    if(L >= IMAX) {
cout << endl << "Reshenieto ne se dostiga s " << L << " iteracii" << endl;
  return 1;
}
} while(FL == false);
cout << endl << "Reshenito se dostiga sled " << L << " iteracii" << endl;
cout << "Reshenieto e:" << endl;
for(int i = 0; i < N; i++) {
  cout << "X[" << i+1 << "] = " << X[i] << endl;
                        }
}
```

Упражнения

1. Кога може да се използват точните методи за решаване на линейни системи и кога итерационните?
2. Кои методи да по-бързи? Винаги ли?
3. Решете системата по всеки от познатите методи:

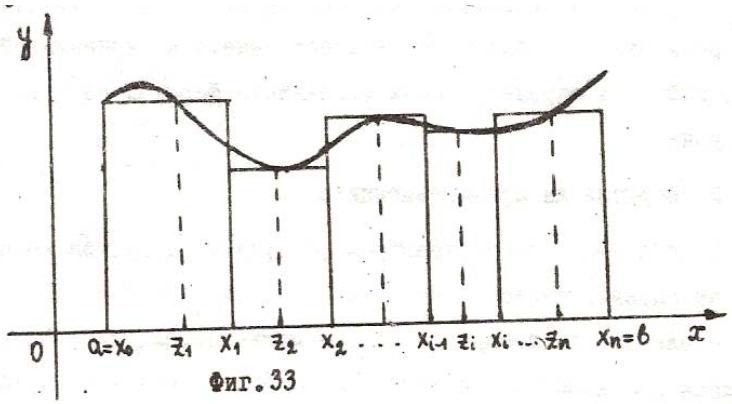
$$\begin{cases} 10X_1 + X_2 + X_3 = 12 \\ 2X_1 + 10X_2 + X_3 = 13 \\ 2X_1 + 2X_2 + 10X_3 = 14 \end{cases}$$

5. Решете задача 3 като използвате горните програми

6. Изчисляване лице на криволинеен трапец

6.1. Лице на криволинеен трапец

Криволинеен трапец се нарича фигура, заградена от абсцисната ос, правите x = a, x = в (a< в) и графиката на функцията y = f(x) (f (x) 0 за $x \in [a, в]$) – фиг.33.



Фиг. 33

Нека интервалът *[a, в]* е разделен на подинтервали с помощта на точките x_i :

$a_0 = x_0 < x_1 < x_2 < \ldots < x = b$

и нека са построени правоъгълниците с основа $\Delta x_i = x_i - x_{i-1}$, $i = \overline{1,n}$ и височини съответно $f(z_i)$, където z_i са произволни точки от подинтервалите [x_{i-1}, x_i]. Всичките правоъгълници образуват стъпаловидна фигура с лице:

$$\sum_{i=1}^{n} f(z_i)\,\Delta x_i = f(z_1)\Delta x_1 + f(z_2)\Delta x_2 + \cdots + f(z_n)x_n$$

Ако подинтервалът $[x_{i-1}, x_i]$ е достатъчно малък, то може да се приеме, че в него функцията е известно приближение с константа и е равна на $f(z_i)$. Грешката при това приближение зависи от големината на интервала и от избора на точката z_i . Геометрически е ясно, че с увеличаване на точките на деление и намаляване дължината на най-големия подинтервал, ще се получават стъпаловидни функции, все по-близки до криволинейния трапец.

Следователно, естествено и да се даде следното определение:

Лице на криволинеен трапец се нарича границата на сумата $\sum_{i=1}^{n} f(z_i)\,\Delta z_i$ при клоненето на $\max_i \Delta x_i$ към нула.

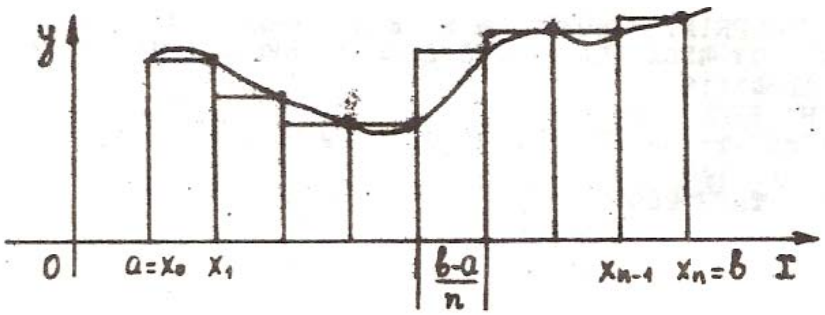
В много случаи се използват достатъчно прости и удобни формули, които позволяват по известен брой стойности на функцията, да се пресметне с необходимата точност лицето на криволинейния трапец, който тя определя. Тези приближени функции се наричат квадратурни.

6.2. Формула на правоъгълниците

Тази най- проста квадратурна формула се получава непосредствено за лице на криволинеен трапец. Нека точките на деление x_i се избират така, че подинтервалите да имат еднаква дължина $\Delta x_i = \frac{b-a}{n}$ и нека $z_i = x_i$. Тогава за приближена стойност на лицето S на криволинейния трапец се приема

$S = \frac{b-a}{n} \sum_{i=1}^{n} f(x_i) = \frac{b-a}{n} [f(x_1) + f(x_2) + \ldots + f(x_n)]$.

Последната формула се нарича квадратурна формула на правоъгълниците. Геометричното й тълкуване (фиг. 34) е просто. S е лицето на криволинейния трапец, а дясната страна е лицето на стъпаловидната фигура. При увеличаване броя на точките – n, лицата на тези две фигури все повече се изравняват



6.3. Формула на трапеците

Формулите на правоъгълниците не се използва много често, понеже за постигане на удовлетворителна точност, интервалът трябва да се раздели на много части. Много по-добри резултати се получават при замяна на правоъгълниците, образувани от всеки един подинтервал, с трапеци.

Дефиниционният интервал се разделя на n равни части с дължина $\frac{b-a}{n}$. Нека точките на разделяне са означени с x_i, а съответните точки на графиката с y_i . Съединяват се точките y_i последователно отсечки и се получава фигура от n трапеца, чието лице при растене на все повече се приближава към лицето на криволинейния трапец.

Тъй като лицето на трапеца $x_i\ y_i\ y_{i+1}\ x_{i+1}$ е равно на :

$$\frac{b-a}{n} \cdot \frac{f(x_i) + f(x_{i+1})}{2}$$

То за лицето на приближената фигура се получава

$$\frac{b-a}{n}.\sum_{i=0}^{n-1}\frac{f(x_i)+f(x_{i+1})}{2} = \frac{b-a}{n}[f(x_0)+\sum_{i=1}^{n}f(x_i)+f(x_n)].$$

Тази формула се нарича квадратурна формула на трапеците.

6.4. Формула на Симпсон

Следващата квадратурна формула дава значително по- добра приближение на лицето на криволинейния трапец от първите две. При нея интервалът [а,в] се разделя на 2n (четен брой) равни части с дължина $\frac{b-a}{2n}$. Нека $x_i = a + \frac{b-a}{2n}$ i са точките на разделяне, а y_i са съответните точки от графиката на функцията. През всяка тройка точки y_{2i}, y_{2i+1}, y_{2+2} се построява парабола и за приближение на кроволинейния трапец y_{2i}, y_{2i+1}, y_{2i+2} се взема лицето на съответния приближен трапец. Сумирайки лицата на всички n на брой параболични трапеца, се получава търсената приближена стойност на лицето на целия криволинеен трапец:

$$S \approx \frac{b-a}{6n}[f(x_0) + 2\sum_{i=0}^{n-1}f(x_{2i}) + 4\sum_{i=0}^{n-1}f(x_{2i-1}) + f(x_{2n})].$$

Тази формула се нарича квадратурна формула на параболите или формула на Симпсон. При еднакъв брой разделящи точки, тя дава най- добро приближение.

Упражнения

1.Пресметнете лицето на кроволинейния трапец, ограничен от функцията sin x и правите $x = 0$ и $x = \frac{\pi}{2}$. Използвайте трите формули при n= 6. Сравнете резултатите с точния резултат 1.

2. като използвате горните три програми, опитайте се да оцените колко повече точки на разделяне са необходими , за да може чрез формулите на правоъгълниците или на трапеците да се достигне точност на формулата на Симпсон.

ЗАКЛЮЧЕНИЕ

В обучението по Информатика се набляга върху усвояването на нови за учащите технологии – познаването и разбирането на работата с определения програмен продукт; характерните системни средства и подходи за обработване на информацията.

Компютърните езици са широко използвани за създаване на продукти, употребата на които имат голямо приложение в информатиката

Поради засиления интерес у учениците към програмирането и голямото му приложение в много от другите предмети изучавани в горния курс на обучение в средното училище,

Изработката му беше голямо предизвикателство за мен. То изисква задълбочено разучаване на програмните продукти и същевременно практикуване на отделните етапи при обработка на програми. За приноса на други проекти могат да се решават задачи посредством използването други езици.

Използвана литература

1. К. Гъров, А. Рахнев, Учебник-записки по програмиране на Бейсик за факултативна подготовка по математика в 9 и 10 клас на ЕСПУ, София 1986г.
2. Х. Крушков, Програмиране на C++, Пловдив 2007 г.
3. М. Петков, Ст. Борушкова, Числени методи за 9 клас, С 1978г.
4. М. Петков, Р. Калтинска, Г. Иванов Ст. Борушкова, Числени методи за 11 клас, София, 1978 г.