Міністерство освіти і науки України

Чернівецький національний університет імені Юрія Федьковича

КУРСОВА РОБОТА

на тему: **«Апроксимація даних сплайновими кривими»**

Студента 4 курсу 422 групи

напряму підготовки

6.113 - «Прикладна математика»

Беньковського А. Р

Керівник: кандидат фізико-математичних наук, доцент Маценко В. Г.

Національна шкала \_\_\_\_\_\_\_\_\_\_\_\_\_

Кількість балів:\_\_\_\_\_

Оцінка: ECTS \_\_\_\_\_

Чернівці – 2020

**АНОТАЦІЯ**

У курсовій роботі розглядається апроксимація даних сплайновими кривими.Реалізовано програмний проект мовою програмування C++.Проводиться розповідь про саму апроксимацію ,що таке апроксимація.Також розповідається про два головних підходи до апроксимації даних. Проводиться порівняльний аналіз апроксимація даних та робиться висновок.

**ЗМІСТ**

Вступ…………………………………………………………………………………………………….4

Розділ 1. Основні поняття та оцінка складності алгоритму………………………………………….5

1.1.Основні поняття………………………………………………………………………………..5

1.2. Поняття складності алгоритму……………………………………………………………….5

Розділ 2. Апроксимація даних сплайновими кривими………………………………………………10

2.1. Складені криві Без’є…………………………………………………………………………10

Розділ 3. Опис та ілюстрація написаного програмного продукту…………………………………..13

3.1.Складені криві Без’є………………………………………………………………………….13

Висновок……………………………………………………………………………………………… ..15

Список використовуваної літератури………………………………………………………………….16

Додаток (код програми)…………………………………………………………………………………17

**ВСТУП**

Метою курсової роботи є вивчення та аналіз апроксимації даних сплайновими кривими, а також розгляд задачі, де використовується апроксимація.

Завданням курсової роботи є вивчення проблем, пов’язаних з апроксимацією даних сплайнових кривих, розгляд застосування апроксимації сплайнових кривих; написання програмного продукту .

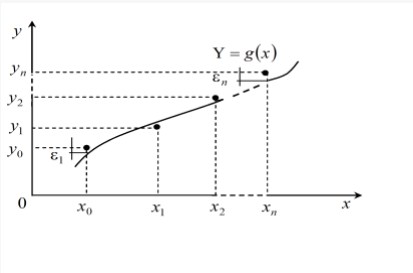
4

Розділ 1. Основні поняття та оцінка апроксимації даних сплайнових кривих

1.1. Основні поняття

*Взизначення.* Апроксимація(Approximation)взагалі- це наближений опис однією функцією (ампроксимувальною) заданого вигляду іншої функції (апроксимованої), яка задається у будь-якому вигляді(при апроксимації даних вона задається у вигляді масивів даних).

Існує два головних підходи до апроксимації даних.При одному з них вимагають,щоб апроксимувальна крива(можливо кусково-гладка) проходила через всі точки,які задані таблицею.Це можна зробити з допомогою методів інтерполяції,які були розглянуті в попередньому розділі. При іншому підході дані апроксимують простою функцією, яка використовується при всіх табличних значеннях, але не обов’язково,щоб вона проходила через всі точки. Такий підхід зветься припасуванням кривої, яку прагнуть провести так, щоб її відхилення від табличних даних був мінімальним. Як правило, користуються методом найменших квадратів (МНК),тобто зводять до мінімуму суму квадратів різниць між значенням функції, яка визначена обраною кривою,та таблицею.



1.2. Поняття складності алгоритму

Створення та реалізація алгоритму відповідно до свого призначення визначає його складність. Проте не існує інтегрованого показника складності алгоритму, хоча існує спеціальний навіть розділ – метрична теорія алгоритмів, що займається саме проблемами складності.

**5**

Інтуїтивно можна виділити такі основні складові складності алгоритму:

1. Логічна складність кількість людино-місяців, витрачених на створення алгоритму.
2. Статична складність − довжина опису алгоритмів (кількість операторів).
3. Тимчасова складність − час виконання алгоритму, кіькість операцій.

4.Ємнісна складність − кількість умовних одиниць пам'яті, необхідних для роботи алгоритму. Головною метою теорії складності є забезпечення механізму класифікації алгоритмів за складністю. Складність алгоритму дозволяє визначитися з вибором ефективного алгоритму серед існуючих, що побудовані для розв’язання конкретної проблеми. А саме вибір серед уже існуючих алгоритмів дозволяє не розглядати логічну та статичну складність, а оцінювати ті ресурси, що знадобляться під час реалізації обраних алгоритмів.

*Означення*.Складність алгоритму – це кількісна характеристика, що відображує споживані алгоритмом ресурси під час свого виконання.Основними ресурсами, що оцінюються, є час виконання і простір пам’яті.Інтуїтивно це поняття досить зрозуміле. В алгоритму є вхід − опис завдання, яке потрібно вирішити. На його розв’язання алгоритм витрачає певний час (тобто кількість операцій). Складність − це функція від довжини входу, значення якої дорівнює максимальному (за будь − якими входами даної довжини) кількості операцій, необхідних алгоритму для отримання відповіді.Одним зі спрощених видів аналізу складності алгоритмів, що використовують при комп’ютерній реалізації, є асимптотичний аналіз алгоритмів. Він використовується з метою порівняння витрат часу та інших ресурсів різноманітними алгоритмами, призначеними для вирішення одного і того самого завдання. Досліджуючи зростання часу роботи алгоритму при вхідних даних досить великих розмірів, ми тим самим вивчаємо асимптотичну ефективність алгоритмів. Це означає, що нас цікавить тільки те, як час роботи алгоритму зростає зі збільшенням розміру вхідних даних,

**6**

коли цей розмір збільшується до нескінченності. Зазвичай алгоритм, більш ефективний в асимптотичному сенсі, буде більш продуктивним для всіх вхідних даних, за винятком дуже маленьких.

Позначення, що вводяться для опису асимптотичної поведінки часу роботи алгоритму, використовують функції, область визначення яких *T(n)* в найгіршому випадку як функції, визначеної тільки для цілих чисел, що становлять розмір вхідних даних.

*Означення.* Функція складності алгоритму *f(n)* має оцінку *Θ(тета)* й записується як *f(n) = Θ(g(n))*, якщо існує невід’ємна функція *g(n)* та додатні *n0, с1, с2*такі, що

*c1g(n) ≤ f(n) ≤ c2g(n),*  (1)

при *n > n0 .*У такому разі говорять, що функція *g(n)* є асимптотично точною оцінкою функції *f(n),* оскільки за визначенням функція *f(n)* не відрізняється від функції *g(n)* з точністю до сталого множника. Важливо розуміти, що *Θ(g(n))* є не однією функцією, а множиною функцій для опису зростання *f(n)* з точністю до сталого множника. Іншими словами, функція *f(n)* належить множині *Θ(g(n)),* якщо існуютьдодатні *с1* та *с2* , що дозволяють обмежити цю функцію у рамки між функціями *c1g(n)* та *c2g(n)* для достаньо великих значень n.

Наприклад, для методу сортування послідовності чисел алгоритмом heapsort оцінка трудомісткості становить *f(n) = Θ(nlog2n)*, тобто в цьому разі *g*(*n*) = *n*log*2n.*

З означення *f(n) = Θ(g(n))* випливає, що *g(n)* = *Θ(f(n)).*Інтуїтивно зрозуміло, що при асимптотично точній оцінці асимптотично невід’ємних функцій, доданками нижчих порядків можна знехтувати, оскільки при великих n вони стають неістотними. Навіть невеликої частки доданка найвищого порядку достатньо для того, щоб перевершити доданки нижчих порядків. Таким чином, для виконання нерівностей (1), достатньо як *с1*

**7**

вибрати значення,

яке дещо менше коефіцієнта при самому старшому доданку, а як *с2* − значення, яке дещо більше цього коефіцієнта. [3]

Тому коефіцієнт при старшому доданку можна не враховувати, тому що він лише змінює зазначені константи.

*Означення*. Функція складності алгоритму *f(n)* має оцінку *О*(«О − велике») й записується як *f(n) = О(g(n)),* якщо існує невід’ємна функція *g(n)* та додатні *n0*, стакі, що

*0 ≤ f(n) ≤ cg(n),* (2)

*при n > n0 .*

*Означення***.** Функція складності алгоритму *f(n)* має оцінку *Ω*(«омега − велике») й записується як *f(n) = Ω(g(n)),* якщо існує невід’ємна функція *g(n)* та додатні *n0,* стакі, що

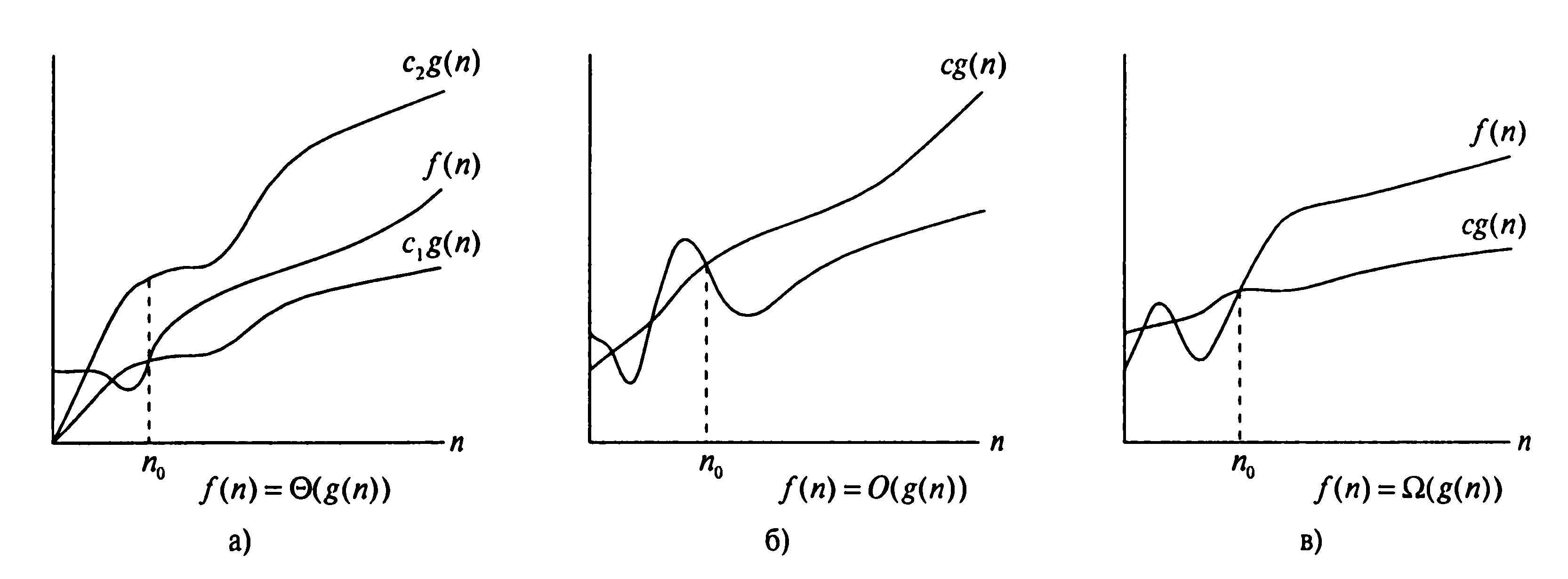
*0 ≤, cg(n) ≤ f(n)*  (3)

*при n > n0 .*

*О ̶* позначення застосовуються, коли необхідно вказати верхню межу функції з точністю до сталого множника. В позначеннях теорії множин *Θ(g(n)).*

Оскільки *О*−позначення описують верхню межу, то в ході їх використання для обмеження часу роботи алгоритму в найгіршому випадку ми отримуємо верхню межу цієї величини для будь − яких вхідних даних.Таким чином, асимптотична оцінка *О(n2)* для часу роботи алгоритму у найгіршому випадку застосовна для часу виконання завдання з будь − якими вхідними даними, чого не можна сказати про *Θ* − позначення. Наприклад, оцінка в *Θ(n2)* для часу сортування вставками в найгіршому випадку не придатна для довільних вхідних даних. Оскільки *Ω* — позначення використовуються для визначення нижньої межі часу роботи алгоритму в найкращому випадку, то вони визначають нижню межу часу роботи алгоритму при довільних вхідних даних. . Наприклад, час роботи алгоритму сортування вставками знаходиться в межах між *Ω(n)* та *O(n2),* тобто між лінійною та квадратичною функціями від n. На рис.1 графічно подані введені вище позначення.

**8**



**Рис. 1.** *Асимптотичний ріст функцій*

Ефективність алгоритму буде оцінюватися за допомогою підрахунку часу виконання алгоритмом конкретно поставленої задачі, тобто з допомогою експерименту.

**9**

Розділ 2. Апроксимація даних сплайновими кривим

2.1.Складені криві Без’є

Опис:

Криві Без’є- це поліноміальні криві, форму яких визначають контрольні точки. Чим більша кількість контрольних точок тим більший ступінь поліноміального опису кривої і тим точніша його побудова. На практиці застосовують криві третього ступеня , які визначаються 4-ма контрольними точками.

Властивості кривої Без’є:

--безперервність заповнення сегмента між початковою та кінцевою точками;

--крива завжди розташовується всередині фігури, утвореної лініями, що з'єднують контрольні точки;

--при наявності лише двох контрольних точок сегмент являє собою пряму лінію;

--пряма лінія утворюється лише тоді, коли контрольні точки розташовані на одній прямій;

--крива Безьє симетрична, тобто обмін місцями між початковою та кінцевою точками (зміна напрямку траєкторії) не впливає на форму кривої;

--масштабування та зміна пропорцій кривої Безьє не порушує її стабільності, оскільки вона з математичної точки зору [«аффінно](https://uk.wikipedia.org/wiki/%D0%90%D1%84%D1%96%D0%BD%D0%BD%D0%B5_%D0%BF%D0%B5%D1%80%D0%B5%D1%82%D0%B2%D0%BE%D1%80%D0%B5%D0%BD%D0%BD%D1%8F) інваріантна»;

--зміна координат хоча б однієї з точок веде до зміни форми всієї кривої Безьє;

--будь який частковий відрізок кривої Безьє також є кривою Безьє;

--степінь кривої завжди на одиницю менший від кількості контрольних точок. Наприклад, при трьох контрольних точках форма кривої — парабола;

--коло не може бути описане параметричним рівнянням кривої Безьє;

--неможливо створити паралельні криві Безьє, за винятком тривіальних випадків

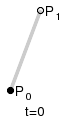
**10**

(прямі лінії та однакові криві), хоча існують алгоритми, що будують наближену

паралельну криву Безьє з прийнятною для практики точністю.

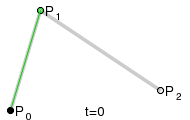
**Види кривих Без’є:**

**Лінійні криві Безьє**



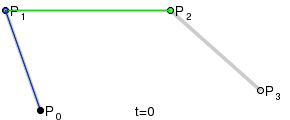
При n = 1 крива є відрізком від точки **P**0 до точки **P**1 ([лінійна інтерполяція](https://uk.wikipedia.org/wiki/%D0%9B%D1%96%D0%BD%D1%96%D0%B9%D0%BD%D0%B0_%D1%96%D0%BD%D1%82%D0%B5%D1%80%D0%BF%D0%BE%D0%BB%D1%8F%D1%86%D1%96%D1%8F)).

**Квадратичні криві Безьє**



Квадратична крива Без'є (n = 2) задається трьома опорними точками: **P**0, **P**1 та **P**2..

**Кубічні криві Безьє**



Чотири опорні точки **P**0, **P**1, **P**2 та **P**3, задані в 2-х чи 3-мірному просторі визначають форму кривої:

Лінія починається в точці **P**0 направляється до **P**1 і закінчується в точці **P**3 підходячи до неї з боку точки **P**2. Тобто крива не проходить через точки **P**1 та **P**2, вони використовуються для напрямку руху

.

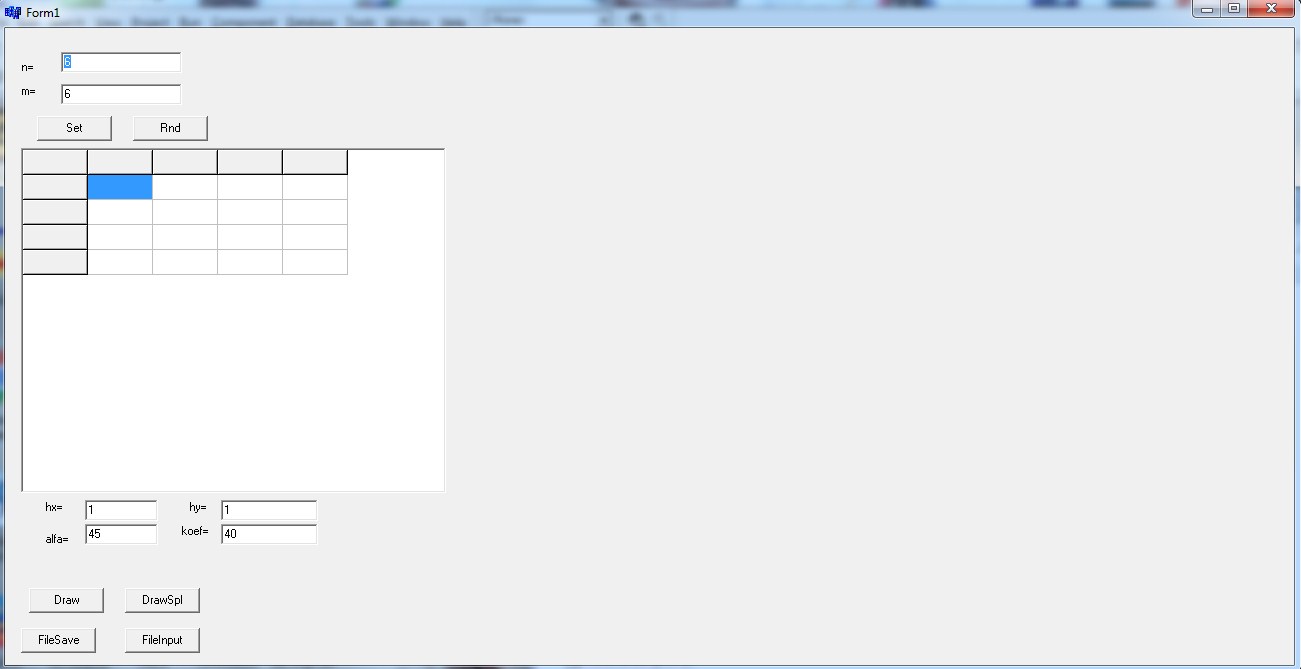
{\displaystyle \mathbf {B} (t)=(1-t)\mathbf {P} \_{0}+t\mathbf {P} \_{1},\quad t\in [0,1].}



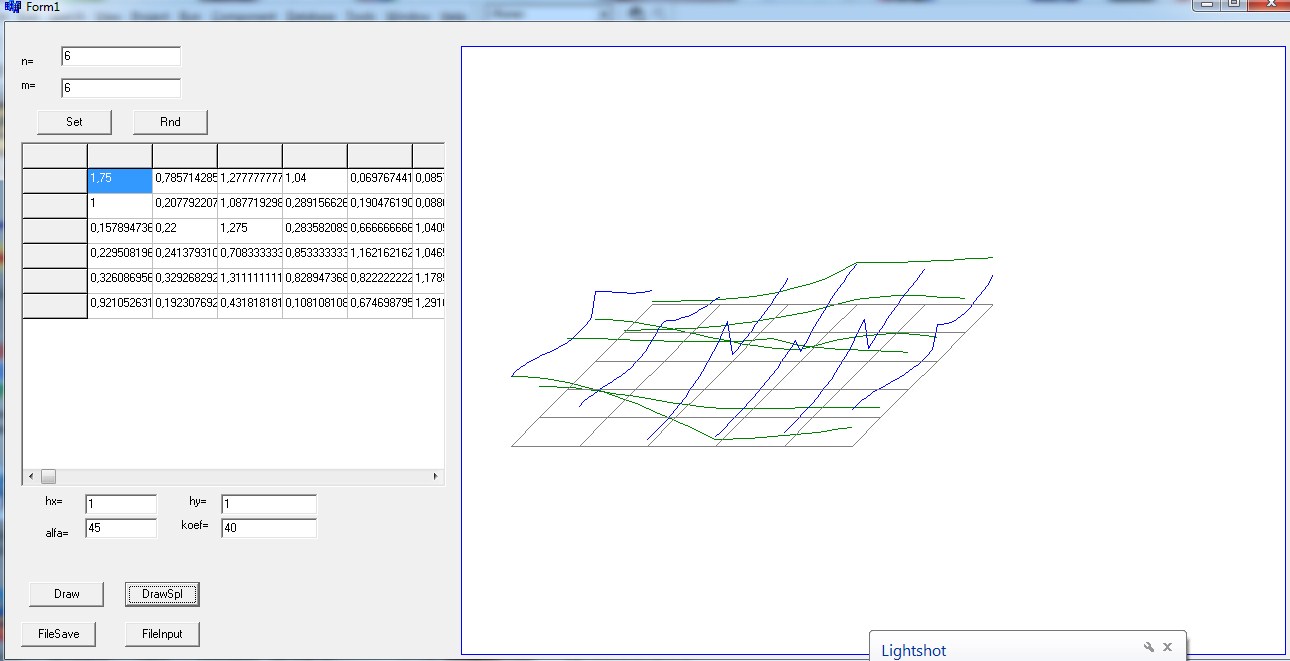
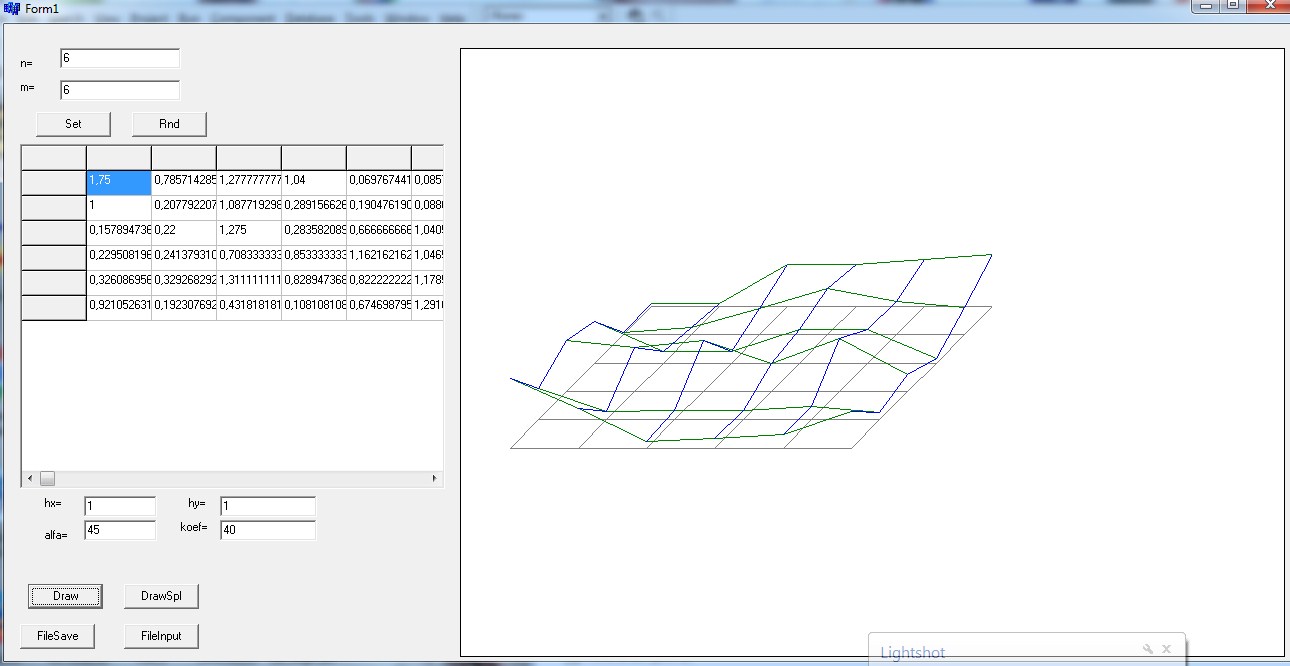
**11**

**РОЗДІЛ 3. ОПИС ТА ІЛЮСТРАЦІЯ РОБОТИ НАПИСАНОГО ПРОГРАМНОГО ПРОДУКТУ**

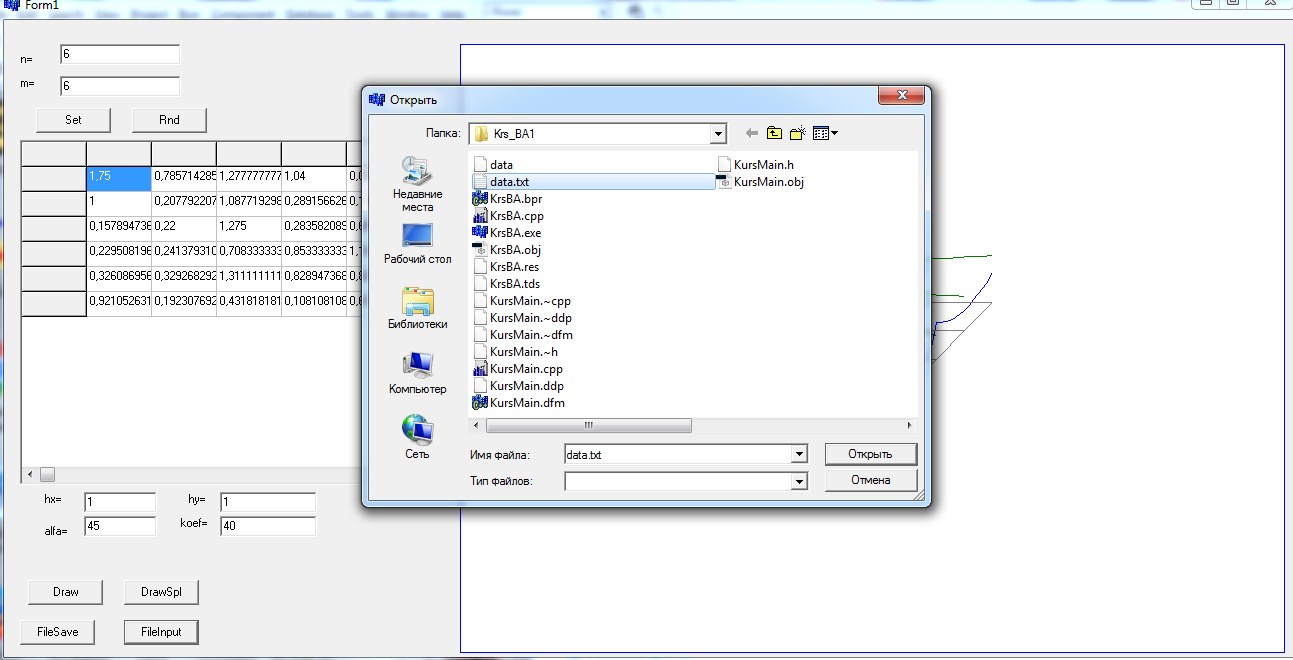
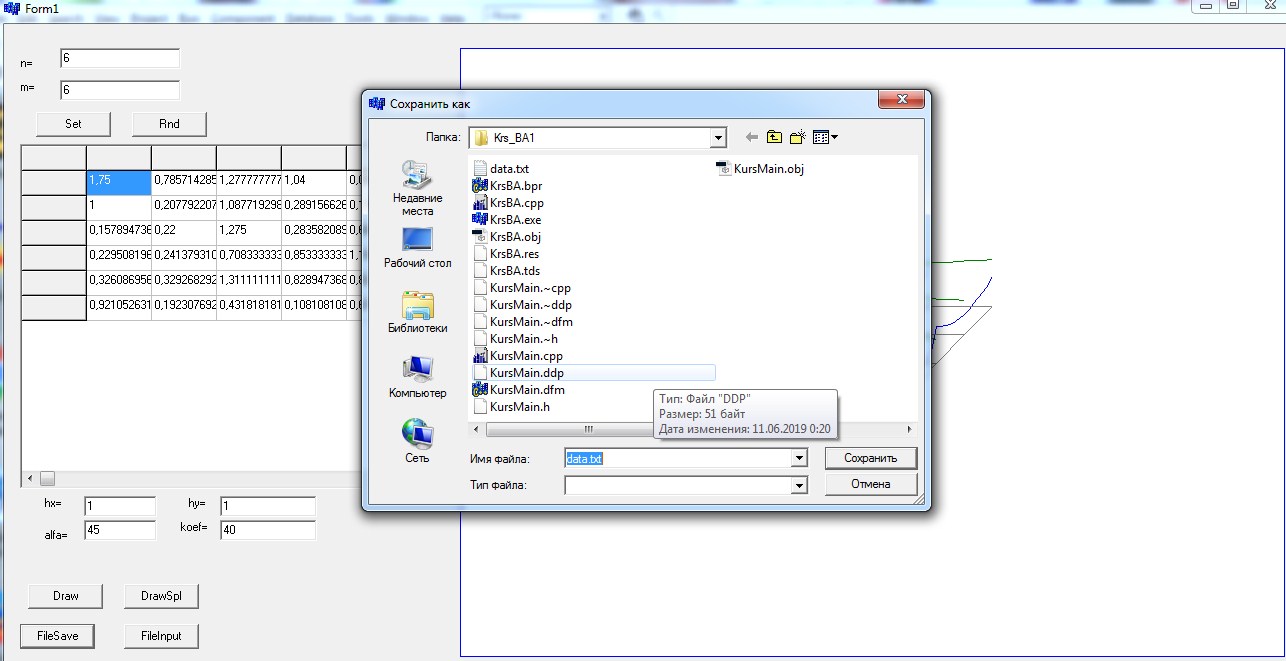
**Ілюстрація програми**



**12**



**13**



**14**

**ВИСНОВОК**

Завдяки простоті завдання і виконанню операцій, криві Безьє знайшли широке застосування в комп'ютерній графіці для моделювання гладких ліній. Крива цілком лежить в опуклій оболонці своїх опорних точок. Ця властивість кривих Безьє з одного боку значно полегшує завдання знаходження точок перетину кривих (якщо не перетинаються опуклі оболонки опорних точок, то не перетинаються і самі криві), а з іншого боку дозволяє здійснювати інтуїтивно зрозуміле управління параметрами кривої в графічному інтерфейсі за допомогою її опорних точок. Крім того, афінні перетворення кривої (перенесення, масштабування, обертання та ін.) також можуть бути виконані через застосування відповідних перетворень до опорних точок.

Найбільше значення мають криві Безьє другого та третього ступенів (квадратичні і кубічні). Криві вищих ступенів при обробці вимагають більшого обсягу обчислень і для практичних цілей використовуються рідше. Для побудови складних за формою ліній, окремі криві Безьє можуть бути послідовно з'єднані один з одним в сплайн Безьє. Для того, щоб забезпечити гладкість лінії в місці з'єднання двох кривих, три суміжні опорні точки обох кривих повинні лежати на одній прямій. У програмах векторної графіки на зразок Adobe Illustrator або Inkscape подібні фрагменти відомі під назвою «контурів» (path).

**15**

**СПИСОК ВИКОРИСТОВУВАНОЇ ЛІТЕРАТУРИ**

1. Альсведе Р., Вегенер І. "Завдання пошуку", − Видавництво "Світ" − 1982. − 268 с.
2. Ахо, Альфред "Структура даних і алгоритми". − Видавничий дім «Вільямс». − 2000. − 384 с.
3. Гасфилд Д. − "Строки, деревья и последовательности в алгоритмах. Информатика и вычислительная биология" − Видавництво "Невской диалект" − 2003. − 658с.
4. Брайан, К. "Практика програмування" − Невськой диалект − 2001. − 381 с.
5. Вірт, М. "Алгоритми і [структури даних](http://ua-referat.com/%D0%A1%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B8_%D0%B4%D0%B0%D0%BD%D0%B8%D1%85" \o "Структури даних)" −  Світ − 1989. − 360 с.
6. Кнут, Д. "[Мистецтво](http://ua-referat.com/%D0%9C%D0%B8%D1%81%D1%82%D0%B5%D1%86%D1%82%D0%B2%D0%BE) програмування на ЕОМ": Том 3. − Світ − 1978. − 356 с.
7. Кормен, Т. "Алгоритми: побудова та аналіз" / Т. Кормен, Ч. Лейзерсон, Р. Ривест − МЦНМО − 2002. − 540 с.
8. Успенський В. "[Теорія алгоритмів](http://ua-referat.com/%D0%A2%D0%B5%D0%BE%D1%80%D1%96%D1%8F_%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC%D1%96%D0%B2): основні відкриття та використання" − [Наука](http://ua-referat.com/%D0%9D%D0%B0%D1%83%D0%BA%D0%B0), 1987. − 288 с.
9. Алгоритми пошуку та сортування [1] — Режим доступу до ресурсу: http://www.ipkro.isu.ru/informat/methods/findsort/
10. Алгоритм Бойера — Мура [2] — Режим доступу до ресурсу: http://cybern.ru/bojera-mura-csharp.html
11. Різні алгоритми пошуку тексту [3] — Режим доступу до ресурсу: http://studopedia.ru/17\_24980\_pryamiy-poshuk-strichki.html
12. Алгоритм Кнутта — Морріса — Пратта [4] — Режим доступу до ресурсу: <https://habrahabr.ru/post/307220/>
13. Роджерс Д. Алгоритмические основы машинной графики. – М.: Мир, 1989. - 512с.
14. Роджерс Д., Адамс Дж. Математические основы машинной графики. – М.: Мир, 2001. - 552с.

**16**

**Додаток (код програми)**

**Криві Без’є**

**Програмний код реалізовано на мові С++ в середовищі Borland C++ Builder**

**//---------------------------------------------------------------------------**

**#include <vcl.h>**

**#pragma hdrstop**

**#include "KursMain.h"**

**//---------------------------------------------------------------------------**

**#pragma package(smart\_init)**

**#pragma resource "\*.dfm"**

**#include<stdlib.h>**

**#include<math.h>**

**#include<stdio.h>**

**TForm1 \*Form1;**

**int SizeN = 9,SizeM = 9;**

**bool setset = false, setrnd =false;**

**double H[81][81], hx, hy;**

**int pt[81][81][2];**

**//---------------------------------------------------------------------------**

**\_\_fastcall TForm1::TForm1(TComponent\* Owner)**

**: TForm(Owner)**

**{**

**setset = false; setrnd =false;**

**}**

**//---------------------------------------------------------------------------**

**void \_\_fastcall TForm1::Button1Click(TObject \*Sender)**

**{**

**SizeN = StrToInt(Edit1->Text);**

**SizeM = StrToInt(Edit2->Text);**

**StringGrid1->RowCount = SizeM + 1;**

**StringGrid1->ColCount = SizeN + 1;**

**setset = true;**

**setrnd = false;**

**}**

**//---------------------------------------------------------------------------**

**void \_\_fastcall TForm1::Button4Click(TObject \*Sender)**

**{**

**int n,m,i,j,r,y;**

**n = SizeN;**

**m = SizeM;**

**randomize();**

**double f;**

**for(i = 0; i<n; i++)**

**for(j = 0; j<m; j++) {**

**r = rand()%100; y = rand()%100;**

**f = (double) (r+3)/(1 + y);**

**if( f>3 ) f = (double)(3 + y )/(r+1);**

**StringGrid1->Cells[i+1][j+1] = FloatToStr(f);**

**}**

**setrnd = true;**

**}**

**//---------------------------------------------------------------------------**

**void \_\_fastcall TForm1::Button5Click(TObject \*Sender)**

**{**

**if(setset==false) Button1Click(Sender);**

**if(setrnd==false) Button4Click(Sender);**

**double al,koef;**

**al = StrToFloat(Edit5->Text); al = al\*M\_PI/180;**

**hx = StrToFloat(Edit3->Text);**

**hy = StrToFloat(Edit4->Text);**

**koef = StrToFloat(Edit6->Text);**

**int n,m,i,j,r,y;**

**n = SizeN;**

**m = SizeM;**

**for(i = 0; i<n; i++)**

**for(j = 0; j<m; j++) {**

**H[i][j] = StrToFloat(StringGrid1->Cells[i+1][j+1]);**

**}**

**int cx, cy;**

**cx = 50;**

**cy = 400;**

**// osnova**

**for(i = 0; i<n; i++)**

**for(j = 0; j<m; j++) {**

**pt[i][j][0] = cx + koef\* ( hx\*j + hx\*j \* cos(al) + hy \* i \*sin(al) );**

**pt[i][j][1] = cy - koef\*( /\*H[i][j] + \*/ hy\*i \* sin(al) /\*+ hy \* j \*cos(al)\*/) ;**

**}**

**Image1->Canvas->Rectangle(0,0,Image1->Width, Image1->Height);**

**//Image1->Canvas->Ellipse(pt[0][0][0]-3,pt[0][0][1]-3,pt[0][0][0]+3,pt[0][0][1]+3);**

**Image1->Canvas->Pen->Color = clGray;**

**for(i = 0; i<n; i++) {**

**Image1->Canvas->MoveTo(pt[i][0][0],pt[i][0][1]);**

**for(j = 1; j<m; j++) {**

**Image1->Canvas->LineTo(pt[i][j][0],pt[i][j][1]);**

**}**

**}**

**// Image1->Canvas->Pen->Color = clGreen;**

**for(j = 0; j<m; j++) {**

**Image1->Canvas->MoveTo(pt[0][j][0],pt[0][j][1]);**

**for(i = 1; i<n; i++) {**

**Image1->Canvas->LineTo(pt[i][j][0],pt[i][j][1]);**

**}**

**}**

**// karkas**

**for(i = 0; i<n; i++)**

**for(j = 0; j<m; j++) {**

**pt[i][j][0] = cx + koef\* ( hx\*j + hx\*j \* cos(al) + hy \* i \*sin(al) );**

**pt[i][j][1] = cy - koef\*( H[i][j] + hy\*i \* sin(al) /\*+ hy \* j \*cos(al)\*/) ;**

**}**

**//Image1->Canvas->Rectangle(0,0,Image1->Width, Image1->Height);**

**//Image1->Canvas->Ellipse(pt[0][0][0]-3,pt[0][0][1]-3,pt[0][0][0]+3,pt[0][0][1]+3);**

**Image1->Canvas->Pen->Color = clGreen;**

**for(i = 0; i<n; i++) {**

**Image1->Canvas->MoveTo(pt[i][0][0],pt[i][0][1]);**

**for(j = 1; j<m; j++) {**

**Image1->Canvas->LineTo(pt[i][j][0],pt[i][j][1]);**

**}**

**}**

**Image1->Canvas->Pen->Color = clBlue;**

**for(j = 0; j<m; j++) {**

**Image1->Canvas->MoveTo(pt[0][j][0],pt[0][j][1]);**

**for(i = 1; i<n; i++) {**

**Image1->Canvas->LineTo(pt[i][j][0],pt[i][j][1]);**

**}**

**}**

**}**

**//---------------------------------------------------------------------------**

**inline double cube(double x) { return x\*x\*x; }**

**inline double sqr(double x) { return x\*x; }**

**void BSL(double h, double p0, double p1, double p2, double p3, int num, double Rez[])**

**{**

**int i;**

**double ch,t;**

**ch = 1./num; t=0;**

**for(i=0;i<num;i++) {**

**Rez[i] = cube(1-t)\*p0 + 3\*t\*sqr(1-t) \*p1**

**+ 3\*sqr(t)\*(1-t)\*p2 + cube(t) \*p3;**

**t+=ch;**

**}**

**}**

**//---------------------------------------------------------------------------**

**void \_\_fastcall TForm1::Button6Click(TObject \*Sender)**

**{**

**if(setset==false) Button1Click(Sender);**

**if(setrnd==false) Button4Click(Sender);**

**int ptx[81][27][27][2];**

**int pty[27][81][27][2];**

**double al,koef;**

**al = StrToFloat(Edit5->Text); al = al\*M\_PI/180;**

**hx = StrToFloat(Edit3->Text);**

**hy = StrToFloat(Edit4->Text);**

**koef = StrToFloat(Edit6->Text);**

**int n,m,i,j,r,y;**

**n = SizeN;**

**m = SizeM;**

**for(i = 0; i<n; i++)**

**for(j = 0; j<n; j++) {**

**H[i][j] = StrToFloat(StringGrid1->Cells[i+1][j+1]);**

**}**

**int cx, cy;**

**cx = 50;**

**cy = 400;**

**// karkas**

**int j0,l;**

**double hxj;**

**int num;**

**num = 18;**

**double Rez[27];**

**for(i = 0; i<n; i++)**

**for(j = 0; j<m; j+=3) {**

**if(j==0) j0=0; else j0=j-1;**

**BSL(hx, H[i][j0], H[i][j], H[i][j+1], H[i][j+2],num,Rez);**

**for (l=0; l<num;l++) {**

**hxj = hx\*j + 3.0\*l/num;**

**ptx[i][j/3][l][0] = cx + koef\* ( hxj + hxj \* cos(al) + hy \* i \*sin(al) );**

**ptx[i][j/3][l][1] = cy - koef\*( Rez[l] + hy\*i \* sin(al) ) ;**

**}**

**}**

**Image1->Canvas->Rectangle(0,0,Image1->Width, Image1->Height);**

**Image1->Canvas->Pen->Color = clGreen;**

**for(i = 0; i<n; i++) {**

**Image1->Canvas->MoveTo(ptx[i][0][0][0],ptx[i][0][0][1]);**

**for(j = 0; j<m; j+=3) {**

**if (j\*3 <= m)**

**for (l=0; l<num;l++)**

**Image1->Canvas->LineTo(ptx[i][j/3][l][0],ptx[i][j/3][l][1]);**

**else for (l=0; l<=(num - num/3);l++)**

**Image1->Canvas->LineTo(ptx[i][j/3][l][0],ptx[i][j/3][l][1]);**

**}**

**}**

**int i0; double hyi;**

**for(j = 0; j<m; j++)**

**for(i = 0; i<n; i+=3)**

**{**

**if(i==0) i0=0; else j0=j-1;**

**BSL(hy, H[i0][j], H[i][j], H[i+1][j], H[i+2][j],num,Rez);**

**for (l=0; l<num;l++) {**

**hyi = hy\*i + 3.0\*l/num;**

**pty[i/3][j][l][0] = cx + koef\* ( hx\*j + hx\*j \* cos(al) + hyi \*sin(al) );**

**pty[i/3][j][l][1] = cy - koef\*( Rez[l] + hyi \* sin(al) ) ;**

**}**

**}**

**Image1->Canvas->Pen->Color = clBlue;**

**for(j = 0; j<m; j++) {**

**Image1->Canvas->MoveTo(pty[0][j][0][0],pty[0][j][0][1]);**

**for(i = 0; i<n; i+=3) {**

**if (i\*3 <= n)**

**for (l=0; l<num;l++)**

**Image1->Canvas->LineTo(pty[i/3][j][l][0],pty[i/3][j][l][1]);**

**else for (l=0; l<=(num - num/3);l++)**

**Image1->Canvas->LineTo(pty[i/3][j][l][0],pty[i/3][j][l][1]);**

**}**

**}**

**// karkas**

**for(i = 0; i<n; i++)**

**for(j = 0; j<m; j++) {**

**pt[i][j][0] = cx + koef\* ( hx\*j + hx\*j \* cos(al) + hy \* i \*sin(al) );**

**pt[i][j][1] = cy - koef\*( 0 + hy\*i \* sin(al) /\*+ hy \* j \*cos(al)\*/) ;**

**}**

**Image1->Canvas->Pen->Color = clGray;**

**for(i = 0; i<n; i++) {**

**Image1->Canvas->MoveTo(pt[i][0][0],pt[i][0][1]);**

**for(j = 1; j<m; j++) {**

**Image1->Canvas->LineTo(pt[i][j][0],pt[i][j][1]);**

**}**

**}**

**for(j = 0; j<m; j++) {**

**Image1->Canvas->MoveTo(pt[0][j][0],pt[0][j][1]);**

**for(i = 1; i<n; i++) {**

**Image1->Canvas->LineTo(pt[i][j][0],pt[i][j][1]);**

**}**

**}**

**}**

**//---------------------------------------------------------------------------**

**void \_\_fastcall TForm1::Button3Click(TObject \*Sender)**

**{**

**SaveDialog1->Execute();**

**AnsiString filename;**

**filename=SaveDialog1->FileName;**

**double d;**

**if(filename!="")**

**{**

**FILE \*fp;**

**fp = fopen(filename.c\_str(),"w");**

**if (fp==NULL) return;**

**fprintf(fp,"%d %d\n", SizeN ,SizeM);**

**for(int i =0; i< SizeN; i++)**

**{**

**for(int j =0; j< SizeM; j++)**

**{**

**d = StrToFloat(StringGrid1->Cells[i+1][j+1]);**

**fprintf(fp, "%f ", d);**

**}**

**fprintf(fp, "\n");**

**}**

**fclose(fp);**

**}**

**}**

**//---------------------------------------------------------------------------**

**void \_\_fastcall TForm1::Button2Click(TObject \*Sender)**

**{**

**OpenDialog1->Execute();**

**AnsiString filename;**

**filename=OpenDialog1->FileName;**

**double d;**

**if(filename!="")**

**{**

**FILE \*fp;**

**fp = fopen(filename.c\_str(),"r");**

**if (fp==NULL) return;**

**fscanf(fp,"%d %d\n", &SizeN ,&SizeM);**

**Edit1->Text = IntToStr(SizeN);**

**Edit2->Text = IntToStr(SizeM);**

**StringGrid1->RowCount = SizeM + 1;**

**StringGrid1->ColCount = SizeN + 1;**

**for(int i =0; i< SizeN; i++)**

**{**

**for(int j =0; j< SizeM; j++)**

**{**

**fscanf(fp, "%lf", &d);**

**StringGrid1->Cells[i+1][j+1] = FloatToStr(d);**

**}**

**}**

**fclose(fp);**

**}**

**setset = true;**

**setrnd = true;**

**}**

**//---------------------------------------------------------------------------**