МІНІСТЕРСТВО ОСВІТИ І НАУКИ, МОЛОДІ ТА СПОРТУ
УКРАЇНИ
ДОНБАСЬКИЙ ДЕРЖАВНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

М. М. Заблодський,
В. Є. Плюгін

СИСТЕМИ АВТОМАТИЗОВАНОГО ПРОЕКТУВАННЯ ЕЛЕКТРОМЕХАНІЧНИХ ПРИСТРОЇВ

Навчальний посібник

Рекомендовано вченою радою ДонДТУ

Алчевськ
2011

УДК 621.313: 004.896
З-12

Заблодський Микола Миколайович – докт. техн. наук, професор, зав. каф. електричних машин і апаратів, проректор з наукової роботи ДонДТУ;

Плюгін Владислав Євгенович – канд. техн. наук, доцент каф. електричних машин і апаратів ДонДТУ.

Рецензенти:

В. І. Мілих – докт. техн. наук, професор, зав. кафедри електричних машин Національного технічного університету «Харківський політехнічний інститут»;

В. С. Петрушин - докт. техн. наук, професор, зав. кафедри електричних машин Одеського національного політехнічного університету.

Рекомендовано вченою радою ДонДТУ
(Протокол № від .03.2010)

Заблодський М. М., Плюгін В. Є.

З-12 Системи автоматизованого проектування електротехнічних пристроїв: Навч. посіб. / М. М. Заблодський, В. Є. Плюгін. - Алчевськ: ДонДТУ, 2011. - 317 с.

ISBN

У навчальному посібнику розглянуті питання розрахунку і автоматизованого проектування асинхронного двигуна з короткозамкненим ротором.

Наведені відомості про мову програмування С++ в середовищі розробки програмного забезпечення Visual Studio, розглянуті питання автоматичної генерації креслярської документації, методи і засоби оптимального автоматизованого проектування електричних машин.

Для студентів спеціальності "Електричні машини і апарати" і інших електротехнічних спеціальностей.

УДК 621.313: 004.896

ISBN

ПЕРЕДМОВА

У основу навчального посібника з курсу "Системи автоматизованого проектування електромеханічних пристроїв" (САПР) покладений однойменний курс лекцій, який читається на факультеті автоматизації виробничих процесів Донбаського державного технічного університету для студентів V курсу спеціальності "Електричні машини і апарати".

Навчальний посібник включає усі розділи односеместрового курсу, а також усі теоретичні відомості, які знаходять застосування при виконанні курсового проекту, контрольних і лабораторних робіт з курсу САПР.

У тексті навчального посібника по усіх розділах є лістинги програм на мові об'єктно-орієнтованого програмування C++ з детальними коментарями, необхідними не лише для розширення уявлень про матеріал, що викладається, але також для самостійної роботи студентів і організації їх знань.

У кінці кожного розділу (окрім останнього) наведені контрольні запитання та практичні завдання, які сприяють кращому сприйняттю та засвоєнню матеріалу навчального посібника.

Низка запитань має оригінальне авторське трактування, відсутнє у виданнях інших авторів. Це структура багатофайлової програми, що забезпечує обмін даними між діалоговим вікном Windows-програми і розрахунковою функцією, розміщеною в окремому файлі; реалізація функції бінарного читання даних з текстового файлу з одночасним їх перетворенням в необхідний тип даних і збереженням в масиві; розробка діалогового застосування з ілюстрованим викладом

створення основних елементів управління; побудова графіка функції у вікні програми-майстра; підготовка робочого місця в середовищі проектування AutoCad; автоматична генерація скриптовых команд AutoLisp в C++ при побудові креслень електродвигуна; алгоритм і програмна реалізація сплайн-інтерполяції, метода покоординатного спуску, Нелдера-Міду і метода зовнішніх штрафних функцій, адаптованих до проектування асинхронного двигуна; авторська розробка повної методики і програмної реалізації оптимального проектування асинхронного двигуна з короткозамкненим ротором з автоматичним формуванням графічної документації; елементи теорії класової побудови електромеханічних пристроїв на прикладі поліфункціонального електротепломеханічного перетворювача енергії; розробка класової структури і програмна реалізація об'єктно-орієнтованого проектування асинхронного двигуна з короткозамкненим ротором.

Вивчені літературні джерела не дозволяють отримати цілісного уявлення про створення повноцінної САПР-програми. При цьому доводиться звертатися до літератури по проектуванню електричних машин, вищій математиці, програмуванню, комп'ютерній графіці та ін. Більше того, розрізнені джерела не мають електротехнічної спрямованості і викладають матеріал у рамках конкретної дисципліни. Автори даного навчального посібника вкладають в поняття САПР електромеханічних пристроїв такі етапи, як проектування електричної машини, її оптимізацію по заданих критеріях, розробку програмного забезпечення, автоматичну генерацію супутньої креслярської документації.

Автори вважають за доцільне і необхідне розвивати у студентів навички програмування для реалізації поставлених проектних завдань в працездатних Windows застосуваннях, а також відноситися до графі-

чних пакетів, таких як AutoCad, не лише як до електронного кульмана, але і як до середовища автоматизованого креслення. Так, окремий розділ присвячений створенню проекту в середовищі Visual Studio, який дозволяє в діалоговій формі пройти етапи проектування від введення початкових даних у вікнах програми-майстра, до генерації креслень асинхронного двигуна з короткозамкненим ротором.

Нарешті, в навчальному посібнику в доступній формі викладаються питання, що знаходять практичне застосування в автоматизованому проектуванні електричних машин і оптимізації: сплайн-інтерполяція табличних залежностей, метод покоординатного спуску, метод багатогранника Нелдера-Міда, що деформується, метод зовнішніх штрафних функцій, метод Парето.

Матеріали, викладені в навчальному посібнику, використовуються у курсовому проектуванні по дисципліні "САПР електромеханічних пристроїв" студентами V курсу спеціальності "Електричні машини і апарати" на кафедрі "Електричні машини і апарати" Донбаського державного технічного університету. Студенти освоюють мову програмування C++, програмні продукти Visual Studio і Visual Lisp в сучасному комп'ютерному класі кафедри. Для інтерактивного навчання студентів застосовується мультимедійний проектор, підключений до робочого місця викладача. Крім того, в локальній мережі кафедри створена і активно використовується студентами електронна бібліотека технічної і методичної літератури, об'єм якої на теперішній час перевищує 1000 екземплярів і більше 10 Гб корисної інформації.

Автори виражають щиру вдячність за допомогу, зроблену при написанні навчального посібника, а також за цінні поради і критичні зауваження:

- Казачишину Віталієві Анатолійовичу, канд.техн.наук, доценту каф. "Машинобудування металургійного комплексу і прикладної механіки" ДонДТУ (розділ 3);

- співробітникам кафедри «Електричні машини і апарти» ДонДТУ канд.техн.наук, доценту Цодіку Ігору Абрамовичу, канд.техн.наук, доценту Овчару Олександру Петровичу;

- шановним рецензентам, за уважне рецензування рукопису та рекомендацій щодо поліпшення окремих його розділів, які не були враховані при доопрацюванні.

Відгуки і побажання просимо направляти за адресою: каф. "Електричні машини і апарати", ДонДТУ, просп. Ленина, 16, м. Алчевськ, Луганська обл., 94204.

ВСТУП

Характерною рисою сучасного етапу розвитку теорії і практики проектування електричних машин є перехід до автоматизованого проектування. Методи і засоби автоматизованого проектування істотно міняють характер інженерної праці, роблячи його творчим і ефективним за рахунок значного розширення можливостей інженера при проведенні проектних досліджень і пошуку оптимальних рішень. В той же час методика автоматизованого проектування припускає високу професійну підготовку сучасного інженера, його уміння грамотно використовувати математичні методи і ЕОМ для прийняття рішень на різних етапах проектування.

В умовах учбового процесу можливості практичного опанування машинних методів проектування обмежені в основному рамками курсового і дипломного проектування. У зв'язку з цим актуальним залишається завдання розвитку і вдосконалення методичного і програмного забезпечення цих видів занять.

Учбове проектування служить придбанню практичних навичок автоматизованого проектування електричних машин. Проектування - ітераційний процес, на кожному кроці якого виконуються з різною мірою детальності опрацювання типові проектні операції:

а) синтез деякої безлічі варіантів проектованого об'єкту (електричної машини);

б) аналіз працездатності синтезованих варіантів на моделі, абстрактній або фізичній;

в) ухвалення рішення по відбору підмножини ефективних варіантів для детальнішого розгляду на наступному кроці.

Учбове (курсове) проектування повинне моделювати ці етапи однієї з ітерацій хоч би спрощено. Пошук оптимального варіанту повинен здійснюватися при варіюванні не менше чим двох незалежних змінних. В той же час збільшення цього числа більше двох недоцільно, щоб не ускладнювати надмірно завдання.

Безліч літературних джерел, присвячених САПР, повною мірою не освітлюють зміст дисципліни, упускаючи з уваги ті необхідні і важливі складові, які роблять матеріал, що вивчається, життєздатним, таким, який можна узяти як керівництво до створення повноцінних САПР проектів.

Розглядаючи САПР в контексті електричних машин, можна виділити наступні компоненти системи, які застосовуються в сучасному електромашинобудуванні:

1) автоматизоване проектування електричної машини;

2) пошук оптимального варіанту спроектованої машини;

3) програмна реалізація проектування і пошуку оптимуму;

4) автоматична генерація креслярської документації.

Численна вивчена література, присвячена САПР і проблемам САПР, не пропонує комплексного освітлення перерахованих складових. Для створення працездатного і закінченого САПР проекту доводиться удаватися до додаткових джерел, які, як правило, мають вузьку спрямованість, представляючи матеріали, тематично не пов'язані один з одним. Методики рішення завдань проекту в різних джерелах також відрізняються, що привносить ще більше труднощів в досягненні поставленої мети.

Даний навчальний посібник покликаний об'єднати усі етапи створення повноцінного САПР проекту і розглянути їх поетапно, по-

чинаючи з вивчення мови програмування, і закінчуючи автоматичним створенням проектної документації.

Програмна реалізація САПР виконуватиметься на мові С++, обраної авторами з наступних міркувань:

- простота в освоєнні на рівні достатньому для формування послідовності розрахунків;

- синтаксис математичних виразів, близький по написанню до таких пакетів як MathCad;

- зручна організація оптимізаційних процедур завдяки використанню усієї потужності об'єктно-орієнтованого програмування, бібліотеці шаблонів STL;

- висока швидкодія, компактність коду, розширюваність програми при внесенні змін;

- просте в розумінні формування людино-машинного інтерфейсу в програмному середовищі розробника Visual Studio;

- сумісність з AutoCad, що використовується надалі для генерації креслярської документації;

- використання мови С++ провідними проектними організаціями усіх країн світу, включаючи Україну.

Формування креслярської документації буде орієнтовано на програму графічного проектування AutoCad, і реалізується у вбудованій в AutoCad середовища Visual Lisp з мовою AutoLisp.

У навчальному посібнику розглядається робота в AutoCad від форматування робочої області до написання програм на Ліспі.

Заключним етапом САПР буде розгляд формування коду побудови креслень в AutoLisp безпосередньо із С++ програми.

Таким чином, досягається поставлена при написанні навчального посібника завдання: пройти непростий шлях від технічного завдання до автоматизованого отримання програмно-реалізованого проекту оптимальної електричної машини і комплекту інженерно-креслярської документації.

РОЗДІЛ 1

ВИКОРИСТАННЯ ОБ'ЄКТНО-ОРІЄНТОВАНОЇ МОВИ ПРОГРАМУВАННЯ С++

У розділах, присвячених програмуванню і складанню програм на мові С++, не ставиться завдання навчитися програмуванню, для цього було б недостатньо навіть двох керівництва, об'ємом як даний навчальний посібник. У наступних шести розділах розглядаються ті аспекти програмування, на які слід звернути увагу при написанні проектів оптимального проектування електричних машин.

1.1 Типи даних

Кожна змінна, яка використовується в програмах на мові С++, має бути "оголошена" [1]. Оголошення змінної має на увазі приналежність її до певного типу даних і резервування пам'яті під цю змінну (табл. 1.1).

Таблиця 1.1 - Основні типи даних мови С++

Назва типу	**Нижня межа діапазону**	**Верхня межа діапазону**	**Розмір в байтах**
bool логічний	False	True	1
char символьний	-128	127	1
int цілий	-2 147 483 648	2 147 483 647	4
float дійсний	$3.4 \cdot 10^{-38}$	$3.4 \cdot 10^{38}$	4

Залежно від призначення, змінні бувають логічного, символьного, цілого і дійсного типів. На практиці використовується значно більше типів даних, включаючи призначені користувачем, але в розрахункових програмах в об'ємах курсу вони будуть незатребуваними.

Приклад оголошення змінних в програмі:

```
int x; //оголошення цілої змінної
float y, z; // оголошення декількох дійсних змінних
int x1 = 10; // оголошення цілої змінної з ініціалізацією
float y1 = - 2.568; // оголошення дійсної змінної з ініціалізацією
float y2 = 3.5e08; // оголошення і ініціалізація дійсного числа в
//експоненціальній формі вигляду 3,7·10^8
int A[10]; // оголошення одновимірного масиву цілого типу
float B[5][6]; // оголошення двовимірного масиву дійсного типу
char s = 'A'; // оголошення і ініціалізація символьної змінної
bool flag = true; // оголошення і ініціалізація логічної змінної
```

У наведеному прикладі подвійною похилою рисою позначається коментар. Будь-які символи, розміщені за коментарем, не обробляються компілятором і виділяються в редакторові середовища спеціальним шрифтом і кольором.

У програмі часто доведеться виконувати операції з рядками. Для оголошення рядків використовуватимемо клас *CString*. Робота із строковими змінними типу *CString* відрізняється простотою, зручністю форматування, підтримує операції виводу в потік. Приклад оголошення строкової змінної:

```
CString text1; // оголошення строкової змінної
CString text 2 = "Приклад рядка"; // оголошення і ініціалізація
//строкової змінної
CSntring text3[3]; // оголошення масиву строкового типу з трьох
//елементів
```

1.2 Перетворення типів

При написанні програми можна зіткнутися з ситуацією, коли доведеться виконувати математичні операції із змінними різних типів. Наприклад, виконати добуток числа полюсів цілого типу і частоти обертання дійсного типу. Якого типу буде результат? Щоб відповісти на це питання, необхідно розібратися з правилами приведення типів.

Перетворення типів здійснюється в наступному форматі (змінна типу 2 перетворюється в тип 1) :

ім'я змінної типу 1 = (тип змінної 1) ім'я змінної типу 2.

Наприклад, перетворення цілої змінної X в дійсну Y:

```
Y = (float) X;
```

Якщо ціла змінна перетворюється в речову "безболісно", то при зворотному перетворенні відсікається частина числа після коми. У прикладі показані різні варіанти перетворення :

```
int X1, X2, X3, X4;
float Y1 = 10.4;
float Y2 = 10.5;
float Y3 = 10.8;
float Y4 0.6;

X1 = (int)Y1; // результат 10
X2 = (int)Y2; // результат 10
X3 = (int)Y3; // результат 10
X4 = (int)Y4; // результат 0
```

Для перетворення типів за правилами округлення необхідно самостійно розробляти підпрограму такого перетворення.

Розглянутий вище спосіб перетворення типів є неявним, оскільки не контролюється програмістом, а виконується в процесі виконання операцій програми. Явне перетворення типів можливе при використ-

танні оператора *static_cast*, наприклад, перетворення типу *int* до типу *float* :

```
int A;
float B;
B = static_cast<float>(A);
```

1.3 Арифметичні операції

У С++ використовуються чотири основні операції: складання "+", віднімання "-", множення "*", ділення "/" та додаткова - залишок від ділення "%".

Арифметична операція "залишок від ділення" або "узяття по модулю" застосовується тільки до цілих чисел, і її результатом буде залишок, що отримується при діленні лівого операнду на правий:

```
6 % 8 (результат 6)
7 % 8 (результат 7)
8 % 8 (результат 0)
9 % 8 (результат 1)
10 % 8 (результат 2)
```

Як видно з прикладу, в результаті виконання операції "%" отриманий чисельник правильного дробу.

Для використання розширених математичних операцій необхідно підключати спеціальну бібліотеку С++ <*cmath*>. Як її підключати, буде розглянуто нижче; при цьому стають доступними наступні операції:

sqrt(x) - витягання квадратного кореня з числа x;

pow(x, y) - піднесення числа x до ступеня y;

log(x) - знаходження десяткового логарифма числа x;

abs(x) - модуль цілого числа x;

fabs(x) - модуль дійсного числа x;

y = modf(x,&n) – повертає дробну **y** та цілу **n** частини числа **x**.

rand() - генерація випадкового числа.

У С++ для скорочення розміру коду використовується особливий запис команд :

- *арифметичні операції з привласненням*

X += Y еквівалентно запису X = X + Y;

X -= Y еквівалентно запису X = X - Y;

X *= Y еквівалентно запису X = X * Y;

X /= Y еквівалентно запису X = X / Y;

- *інкремент*

X++ еквівалентний запису X = X + 1;

- *декремент*

X-- еквівалентний запису X = X - 1.

Інкремент і декремент можуть мати префіксну і постфіксну форми, які найчастіше застосовуються в циклічних обчисленнях:

- *префіксна форма*

++x; // число x спочатку збільшується на 1, а потім передається в цикл;

- *постфіксна форма*

x--; // число x передається в цикл, а при наступному зверненні зменшується на 1.

1.4 Логічні операції і галуження

До логічних операцій відносяться операції порівняння :

- менше <

- більше >

- менше або рівно <=

- більше або рівно >=

- не рівно !=

- рівно == (два знаки рівно без пропуску)

- логічне «або» ||

- логічне «і» &&.

У С++ існує декілька типів галужень, найбільш важливим з яких є *if...else*, що здійснює вибір між двома альтернативами. Для вибору однієї з безлічі альтернатив використовується оператор галуження *switch*, дія якого визначається набором значень відповідної змінної цілого або символьного типу.

Оператор порівняння if. Логічні операції порівняння записуються в блоці *if ... else* (*якщо…інакше*), наприклад:

```
if (a < 2)
        a++;
if (a == b)
        c = 2;
else
        c = 3;
```

Порівняння з нулем має декілька інший, спрощений запис:

```
if(!a) // аналогічно запису if (a == 0)
        b = 1;
if (a) // аналогічно запису if (a != 0)
        b = 3;
```

Проте, вирази a == 0 і a != 0 не вважатимуться помилкою.

Якщо в логічних операціях порівняння вимагається більш, ніж один рядок, то група операцій об'єднується у фігурні дужки:

```
if (!a) //якщо a = 0
{
b =2:
c += 3
}
else if (a > 0) //інакше, якщо a > 0
{
b = 10;
c = 5;
}
else //інакше (якщо перші дві умови не виконуються)
{
b = 0;
c = 0;
}
```

Умовна операція виду

```
if (x < y)
        min = x;
else
        min = beta;
```

ідентична наступному запису:

```
min = (x < y)? x: y;
```

Після знаку питання перша змінна привласнюється у разі істинності результату порівняння, а друга - при помилковому.

Потрібно відмітити, що в операторові галуження *if...else* використання *else* не є обов'язковим.

Оператор switch. Якщо галуження в програмі залежать від значення однієї змінної цілого або символьного типу, то можна замість багатоступінчастої послідовності *if...else* застосувати оператор *switch*:

```
switch (index) // залежно від значення цілої змінної index
{
case 0: // якщо index = 0
        text = "вибраний перший варіант";
        break;
case 1: // якщо index = 1
        text = "вибраний другий варіант";
        break;
default: //якщо не вибраний жоден із варіантів в записі case
        text = "варіант не вибраний";
        break;
}
```

1.5 Організація циклів

Дія циклів полягає в послідовному повторенні певної частини програми деякої кількості разів. Повторення триває до тих пір, поки виконується відповідна умова. Коли значення виразу, яке задає умову, стає помилковим, виконання циклу припиняється, а управління передається операторові, що йде безпосередньо за циклом. У С++ існує три типи циклів : *for, while і do*.

Оператор циклу for. Цикл *for* застосовується при зміні величини від початкового значення до кінцевого із заданим кроком. Змінні, записані в круглих дужках циклу мають бути цілого типу. Наприклад:

```
for (int i = 0; i <= 10; i++)
{
        A[i] = 10*I;
        N += i;
}
```

Оператор циклу while. Оператор *while* застосовується у тому випадку, якщо кількість циклів невідома, а кількість циклічних операцій залежить від виконання певної умови.

Наприклад:

```
i = 0;
while ( a < E) // доки а менше, ніж Е
{
        a = 10*B[i]; // виконуємо операцію
        i++; // збільшуємо індекс масиву
}
```

Оператор циклу do. Оператор *do*, також як і *while*, застосовується у тому випадку, якщо кількість циклів невідома, а кількість циклічних операцій залежить від виконання певної умови. Головною відмінністю циклу з використанням оператора *do* є те, що тіло циклу, незалежно від істинності умови, що перевіряється, виконається хоч би один раз:

```
i = 0;
do
{
        a = 10*B[i]; // виконуємо операцію
        i++; // збільшуємо індекс масиву
} while ( a < E); // доки а менше, ніж Е
```

1.6 Структури

Структура є об'єднанням простих змінних. Ці змінні можуть мати різні типи: *int*, *float* і так далі. Саме різнорідністю змінних структури відрізняються від масивів. Змінні, що входять до складу структури, називаються полями.

```
struct motor //оголошення структури
{
int serial_ID; //серійний номер
String type; //тип двигуна
float power; //потужність
}; //обмежувач визначення структури
```

```
//Основна програма
{
motor AM; // визначення структурної змінної
//Ініціалізація полів змінної AM
AM.serial_ID = 10;
AM.type = "Asynchronous motor";
AM.power = 15;
//Визначення і ініціалізація полів змінної SM
motor SM = {100, "Synchronous motor", 200F};
…
}//кінець тіла основної програми
```

Для структурних змінних допустима операція привласнення. Структури можуть бути вкладеними.

```
struct size
{
int ID;
float length;
float width;
float height;
};
struct motor //оголошення структури
{
int serial_ID; //серійний номер
CString type; //тип двигуна
float power; //потужність
size limits; // вкладена структура
};
```

```
motor AM; // визначення структурної змінної
//Ініціалізація полів змінної AM
AM.serial_ID = 10;
AM.type = "Asynchronous motor";
AM.power = 15;
AM.limits.length = 100;
AM.limits.width = 200;
AM.limits.height = 300;

//Визначення і ініціалізація полів змінної SM
motor SM = {100, "Synchronous motor", 200F, {150, 250, 350}};
…
}//кінець тіла основної програми
```

1.7 Перерахування

Перерахування використовуються в тих випадках, коли змінні створюваного типу можуть приймати заздалегідь відому кінцеву безліч значень.

Оголошення типу починається із слова *enum* і містить перерахування усіх можливих значень змінних створюваного типу. Ці значення називаються константами перераховуваного типу.

Звернення до значень змінної перераховуваного типу здійснюється по іменах.

Першої з перераховуваних констант відповідає ціле значення, рівне 0, другої - значення, рівне 1, і так далі:

```
enum state {NO, YES}; // NO = 0, YES = 1
…
state flag;
if (flag == YES)
{
        flag = NO;
}
```

Для того, щоб змінити значення, з якого починається нумерація, можна за допомогою операції привласнення задати це значення першої з перераховуваних констант:

```
enum Part {a = 1, b, c, d};
```

В цьому випадку наступним за списком константам відповідатимуть числа 2, 3 і 4 відповідно. Ми можемо змінити величину цілого числа, відповідного будь-якому елементу списку, застосувавши до нього операцію привласнення.

Приклади:

```
enum season {Winter, Spring, Summer, Autumn};
enum switch {off, on};
```

Недоліком перерахувань є те, що вони не розпізнаються засобами введення/виведення C++.

1.8 Структура програми

Розглянуті вище оператори у відповідність з поставленими завданнями формують програму, яку прийнято називати "код". Код виконуваних операцій "пишеться" в текстових файлах, що відкриваються, наприклад, програмою Windows "Блокнот". Нині будь-яка програма супроводжується призначеним для користувача інтерфейсом, що складається з діалогових вікон, полів введення, випадних списків і так далі. Крім того, набраний код має бути скомпільований в виконавчий *exe*-файл. Для вирішення складних питань створення таких Windows-орієнтованих пакетів програм використовується просте в розумінні середовище проектування Visual Studio. У ньому розміщуватимуться файли для написання коду програми, що розробляється. Створення

файлів детально розглядається як в поточному розділі, так і в наступних розділах навчального посібника.

Заголовні файли і функції. Програму слід розміщувати в двох файлах: заголовному, в якому знаходиться оголошення і ініціалізація змінних і головному, в якому пишеться код розрахунків.

Ім'я заголовного файлу повинне мати розширення "h". Ім'я файлу з текстом програми має розширення "cpp". Заголовний файл на диску повинен знаходитися в одній теці з "cpp" файлом. Включення заголовного файлу здійснюється за допомогою директиви препроцесору *#include*. При цьому усі змінні, оголошені в заголовному файлі, стають доступними для "cpp" файлу так, як ніби то були оголошені в ньому безпосередньо.

Переваги оголошення змінних в заголовному файлі:

- легкість для читання основного коду програми, не переобтяженого оголошеннями змінних і функцій;

- доступність змінних для міжфайлового обміну.

В курсі САПР усі розрахунки будуватимуться на виклику підпрограм, розміщених в одному файлі. У С++ ці підпрограми називають функціями. Кожна функція має наступний формат:

```
<тип повертаного значення> ім'я (<аргумент 1>, <аргумент 2>, …)
{ // початок тіла функції
// код функції
} // кінець тіла функції
```

Приклад програми на С++

```
//Заголовний файл з оголошенням змінних
// Файл program.h
int x;
int y;
```

```
//Розрахунковий файл program.cpp
#include "program.h"
int f1() //ім'я функції
{
x = 10*y; // розрахунок в тілі функції
return x; // повернення значення типу int в основну програму
}
```

Якщо функція не повертає ніяких значень, а просто виконує послідовність закладених алгоритмом дій, то вона повинна мати тип *void*. При цьому оператор *return* не використовується:

```
void f2()
{
x = 10*y;
}
```

Структура програми з декількох функцій. Якщо в одному файлі оголошено декілька функцій, то треба забезпечити доступ однієї функції до іншої. Як правило, у файлі завжди розміщується головна функція і ряд допоміжних, які викликаються з головної. Усі допоміжні функції мають бути оголошені як "зовнішні" з ключовим словом *extern*:

```
#include "program.h" //включення заголовного файлу
extern int integ(float); //оголошення зовнішньої функції

void f1 ()
{
x = integ(y); //виклик функції integ()
}
//Реалізація зовнішньої функції
int integ(float yy)
{
return (int) yy;
}
```

Збереження інформації в тестовому файлі. У С++ операції виводу у файл вирішуються просто завдяки концепції потоків. Передати значення змінної у файл можна в три прийоми:

1) включити заголовний файл <*ofstream*> без розширення;

2) оголосити змінну типу *ofstream*;

3) передати значення у файл за допомогою оператора виводу в потік "<<".

Повний приклад програми

```
//Заголовний файл program.h
#include <ofstream> //для операцій запису у файл
#include  <cmath> //для розширених математичних функцій
float x;
float pi = 3.14;

//Файл програми program.cpp
#include "program.h"
using namespace std; //оголошення стандартного простору імен
void f1()
{
ofstream df("file.txt"); // оголошення файлової змінної "df"
x = pow(pi, 3);
df << "x = " << x << "\n"; //виведення значення "X" у файл //"file.txt"
}
```

У лістингу для виконання операції зведення до ступеня *pow()* була підключена бібліотека математичних операцій <*cmath*>. При виведенні значень у файл використовуються символи форматування рядка - послідовності, що управляють виводом у потік, наведені в табл. 1.2.

Оголошення стандартного простору імен командою *using namespace std* застосовується тільки у тому випадку, якщо в програму включаються заголовні файли без стандартного розширення "h".

Таблиця 1.2 - Символи форматування рядка

Послідовність	Символ
\n	Переведення в початок наступного рядка
\t	Табуляція
\\	Обернена коса риска
\'	Одинарні лапки
\"	Подвійні лапки

Відкриття файлу в бінарному вигляді. Бінарна передача даних здійснюється набагато швидше, ніж в десятковому форматі, особливо якщо доводиться звертатися до файлів даних багато разів в циклах розрахунку. Бінарний тип обміну даних знадобитися при роботі з кривими намагнічування стали, а також для інтерполяції графіків розрахункових коефіцієнтів. Зрозуміти суть бінарного обміну даними допоможе простий приклад передачі даних з файлу в масив.

```
float dimB[100]; // оголошення масиву типу float
int N; //кількість даних, завантажених з файлу
void f1()
{
ifstream df("B.txt"); //оголошення файлової змінної df
df.seekg(0, ios::beg); //установка покажчика читання в початок файлу
char buffer[100]; //масив символьного типу (буфер)
int index = 0; //обнулення лічильника завантажених з файлу даних
while (df.good()) // доки не досягнутий кінець файлу
{
        df.getline(buffer, sizeof(buffer)); // перевести рядок в буфер
        dimB[index] = atof(buffer); //перетворити символи в дійсний тип
        index++; // збільшити лічильник
}
N = index - 1; //ініціалізація кількості завантажених даних
}
```

Заповнений даними масив *dimB[]* можна тепер використовувати в програмі, наприклад, помістити в цикл розрахунку. Значення *N* кількості елементів масиву може знадобитися при знаходженні мінімального/максимального значення масиву.

У лістингу програми функція *atof()* перетворює символи в дійсний тип. Для перетворення символьних даних в цілий тип використовується схожа функція *atoi()*. Якщо з тестового файлу завантажується символьна інформація, наприклад, найменування марки проводу, то має бути виконана процедура перетворення масиву символів *char* в строковий тип *CString*, яка реалізується автоматично завдяки приведенню типів:

```
char buffer[100];
CString text;
//Завантаження даних з файлу df в символьний масив buffer
df.getline(buffer, sizeof(buffer));
//Перетворення масиву символів buffer в строковий тип
text = buffer;
```

Структура багатофайлової програми. Усі С++ програми, з якими доведеться стикатися при вивченні даного курсу, матимуть до 100 файлів в одному проекті. Тим часом, код головної розрахункової програми розміщуватиметься тільки в двох файлах: заголовному "h" і розрахунковому "cpp" файлі. Розглянемо порядок дій для зв'язку розрахункового коду програмами з діалоговим вікном (рис. 1.1).

Для визначеності приймемо наступні припущення:

- змінні діалогу m_x1, m_x2, m_y1, m_y2 типу *float* оголошені у файлі "Dialog.h", а код представлення діалогу знаходиться у файлі "Dialog.cpp";

- змінні програми розрахунку x1, x2, y1, y2 типу *float* оголошені у файлі "program.h", код програми розрахунку знаходиться у файлі "program.cpp";

- дані з вікна діалогу прочитуються при натисненні на кнопку "SAVE".

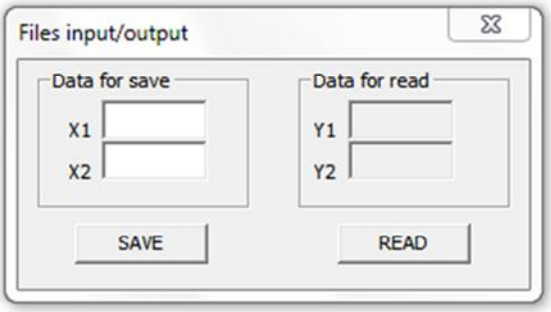


Рисунок 1.1 - Діалогове вікно простої багатофайлової програми

Порядок дій наступний.

1) Створюємо файл "program_extern.h", в якому оголошуємо зовнішні змінні, до яких необхідно відкрити доступ у файлі діалогу "Dialog.cpp":

```
//Файл program_extern.h
extern float x1;
extern float x2;
extern float y1;
extern float y2;
extern void function();
```

2) Включаємо у файл діалогу "Dialog.cpp" файл program_extern.h, розмістивши в секції *#include* рядок:

```
#include "program_extern.h";
```

Цими діями змінні x1, x2, y1 і y2, оголошені у файлі program.h, придбають статус глобальних і стають доступними для використання у файлі діалогу.

3) Розміщуємо в тілі функції, сформованої після подвійного натиснення лівої кнопки миші (далі ЛКМ) на кнопку "SAVE" на формі діалогу, код привласнення змінним x1, x2 тих значень, які були введені користувачем у вікні застосування, і виклик розрахункової функції *function()* програми :

```
{
UpdateData(); //Відновити вікно і переписати їх вміст в m_x1, m_x2
x1  = m_x1; //привласнити змінній x1 значення, введеного у вікні  m_x1
x2 = m_x2; //привласнити змінній x2 значення, введеного у вікні  m_x2
function(); // викликати розрахункову функцію з файлу program.cpp
m_y1 = y1; // привласнити розраховане значення y1 змінній m_y1
m_y2 = y2; // привласнити розраховане значення y2 змінній m_y2
UpdateData(FALSE); //Відобразити у вікні значення змінних m_y1 і m_y2
}
```

Для повноти опису покажемо зміст розрахункових файлів програми.

```
//Файл program.h
#include "cmath" //заголовний файл математичних операцій
float x1;
float x2;
float y1;
float y2;
//кінець файлу program.h
```

```
//Файл program.cpp
#include "stdafx.h" //необхідний файл для роботи в середовищі Windows
#include " program.h" //включаємо заголовний файл
```

```
//Обов'язкова секція забезпечення міжфайлового обміну
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE
#endif

using namespace std; //оголошуємо стандартний простір імен

//функція розрахункової програми
void function()
{
//при виклику функції значення x1 і x2 вже ініціалізовані
//даними вікна діалогу m_x1, m_x2
y1 = x1 + 10;
y2 = x2 + 100;
}
//кінець файлу program.cpp
```

Схематично операції міжфайлового обміну даними показані на рис. 1.2.

1.9 Передача аргументів по посиланню

Посилання є псевдонімом або альтернативним ім'ям змінної. Посилання застосовуються для передачі аргументів у функції.

При передачі аргументів за значенням функція, що викликається, створює нові змінні, які мають ті ж типи, як і аргументи, і копіює значення аргументів в ці змінні. При цьому функція не має доступу до змінних-аргументів, а працює із зробленими їй копіями значень.

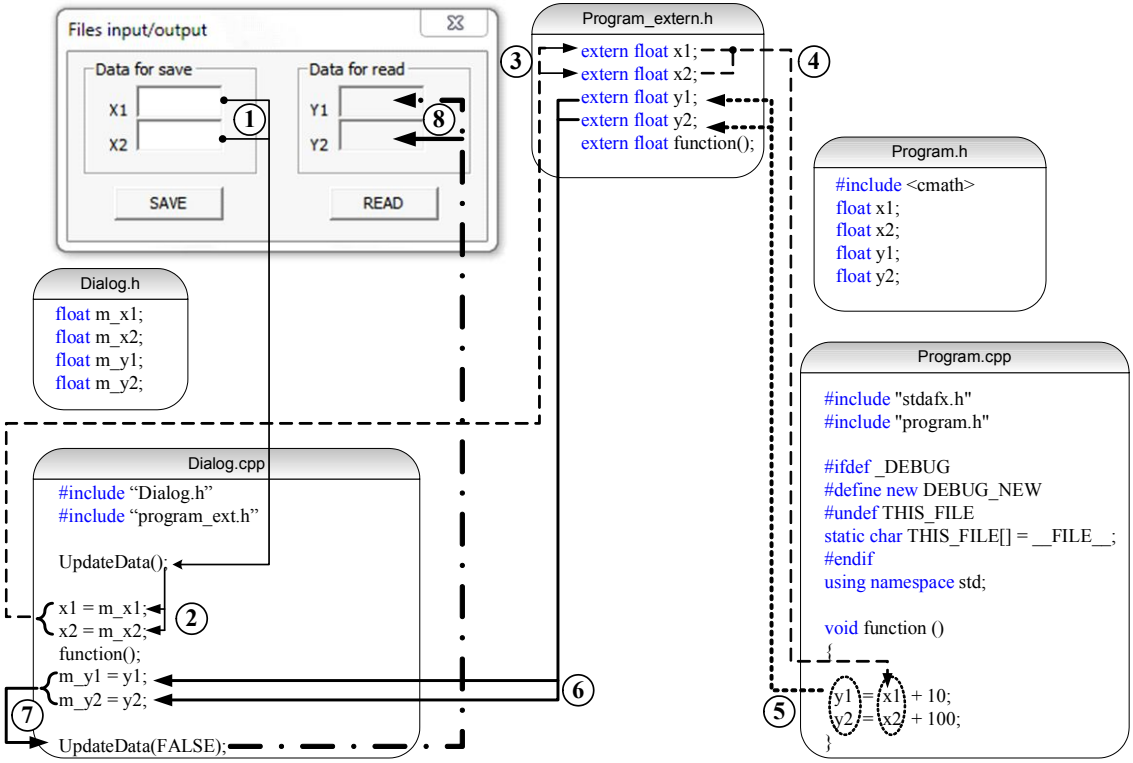


Рисунок 1.2 - Схема міжфайлового обміну даними

При передачі по посиланню програма виклику вказує функції змінні, які треба обробити, а функція обробляє ці змінні, навіть не знаючи їх справжніх імен.

Переваги механізму посилань:

1) функція має прямий доступ до значень аргументів;

2) можливість повернення функцією програмі не одного, а безлічі значень.

Приклад передачі аргументів по посиланню:

```
void function ()
{
int& ref (int& x1); // прототип функції (попереднє оголошення)
int x, y;
y = ref(x);
}

int& ref (int& x1)
{
return x1*10;
}
```

Якщо до позначення типу функції додати знак &, виділений в лістингу напівжирним шрифтом, то по посиланню також повертатимуться і значення функції.

Розглянемо приклад програми знаходження коріння квадратного рівняння використовуючи набуті відомості про перерахування, структури і посилання.

```
#include <cmath> //підключення математичної бібліотеки
using namespace std;

enum state {NO, YES}; //стан знаходження коріння (перерахування)
```

```
//структура коріння квадратного рівняння
struct root
{
float x1; //поле першого кореня
float x2; //поле другого кореня
state flag;//прапор наявності/відсутності коріння
};
//структура коефіцієнтів квадратного рівняння вигляду y = ax^2 + bx +c
struct koeff
{
float a;
float b;
float c;
};


//Головна програма, запускається по натисненню на кнопку ОК діалогу
void CSimpleDlg::OnBnClickedOk()
{
root& find_roots(koeff&); //прототип функції розрахунку коріння
koeff k; //оголошення структури коефіцієнтів
root y; //оголошення структури коріння рівняння
//ініціалізація полів структури коефіцієнтів
k.a = 1;
k.b = 4;
k.c = 2;
//виклик функції пошуку коріння
y = find_roots(k);
//інформаційне повідомлення про результати розрахунку
if (y.flag) //якщо прапор встановлений в положення 1 (YES)
        AfxMessageBox("Коріння знайдене"!);
else //інакше (прапор в положенні 0 - NO)
        AfxMessageBox("Коріння не знайдене, дискримінант негативний");
}
//Функція знаходження коріння кв. рівняння вигляду y = ax^2 + bx + c
root& find_roots(koeff& k1)
{
root y1; //тимчасова змінна структури коріння
float D; //дискримінант
D = k1.b*k1.b - 4*k1.a*k1.c; //Розрахунок дискримінанта
```

```
if (D>=0)//якщо дискримінант позитивний
{//виконується розрахунок коріння
        y1.x1(2*k1.a);
        y1.x2(2*k1.a);
        y1.flag = YES;//прапор встановлений в положення 1
}
else //якщо дискримінант негативний
{//коріння не розраховується
        y1.x1 = - 1;
        y1.x2 = - 1;
        y1.flag = NO;//прапор встановлений в положення 0
}
return y1;//повернення структури коріння рівняння
}//кінець функції знаходження коріння квадратного рівняння
```

1.10 Перевантаження

Функції, що мають однакові імена, але різні типи і/або кількість аргументів, сприймаються програмою як різні функції. Перевантажена функція виконує різні дії залежно від типу і/або кількості аргументів.

Наприклад, три оголошення функції з однаковими іменами

```
int function(int, int);
float function(int);
int function(float);
```

розрізняються, і при зверненні до свого імені викликатимуться у відповідності за типом і кількістю аргументів.

1.11 Типи змінних і зона видимості

Зона видимості визначає, з яких частин програми можливий доступ до змінної. Існує дві зони видимості: локальна і зона видимості файлу. Змінні, визначені в локальній зоні видимості доступні усередині того блоку, в якому визначені. Змінні, що мають зону видимості файлу, доступні з будь-якого місця файлу.

Змінні можуть відноситися до певного класу пам'яті: автоматичному або статичного. Клас пам'яті - це час, протягом якого змінна існує в пам'яті комп'ютера. Автоматичні змінні мають час життя, рівний часу життя функції. Час життя змінних, що мають статичний клас пам'яті, дорівнює часу життя програми.

Локальні змінні. Локальні змінні визначаються усередині функції і відносяться до автоматичного класу пам'яті. Локальна змінна не існує в пам'яті до тих пір, поки не буде викликана функція, в якій вона визначена. Час життя локальної змінної дорівнює часу, що проходить з моменту оголошення змінної в тілі функції до завершення виконання функції.

Глобальні змінні. Глобальні змінні визначаються поза якимись-небудь функціями і видимі з усіх функцій цього файлу і, як буде розглянуто далі, з інших файлів. Глобальні змінні видимі тільки для тих функцій, які розташовані нижче за їх оголошення. Якщо глобальна змінна не була ініціалізованою при оголошенні, то на момент запуску програми їй буде привласнено нульове значення. Глобальні змінні мають статичний клас пам'яті.

Статичні локальні змінні. Статична локальна змінна має таку ж зону видимості, як і автоматична. Проте, час життя статичної локальної змінної починається при першому виклику функції, в якій вона визначена і закінчується при закритті програми.

Статичні локальні змінні визначаються за ключовим типом *static* перед вказівкою типу змінної і використовуються в тих випадках, коли необхідно зберегти значення змінної в пам'яті після того, як виконання функції буде завершено (між викликами функцій).

Константні аргументи. Для того, щоб запобігти зміні значення змінної, треба вказати модифікатор *const* перед відповідним аргументом. Якщо константні аргументи використовуються в прототипах функції, то обов'язкове оголошення імені константного аргументу:

```
int function(int&, const int& b);
```

1.12 Масиви

Масиви призначені для угрупування елементів одного типу. Вони можуть містити від декількох одиниць даних до декількох мільйонів. Дані, згруповані в масиві, можуть бути як основних типів, таких як *int*, *float*, *CString*, так і визначених користувачем типів - структур або об'єктів. Відмінність масивів від структур полягає в тому, що в структурі зазвичай групуються змінні різних типів, а в масиві - тільки однотипні дані. Крім того, дістати доступ до елементу структури можна по імені, а в масиві - по індексу. Використання індексу для кожного з елементів дозволяє спростити доступ у разі великої кількості елементів.

Оголошення масивів. У розрахункових програмах найчастіше використовуються одновимірні масиви, вживані для зберігання послідовності даних, таких, що складаються з одного рядка. Приклад ініціалізації одновимірних масивів :

- при оголошенні

```
float A[] = {0.1, 0.2, 0.3}; // ініціалізація масиву з трьох елементів
CString S[] = {"сталь 2013", "сталь 2212"}; // ініціалізація строкового
//масиву
```

- в циклі розрахунку

```
for (int i = 0; i <= 10; i++)
{
	dim_P1[i] = P2/kpd;
	P2 += 1000;
}
```

Для зберігання даних, представлених в табличній формі, застосовуються двовимірні масиви.

Приклад ініціалізації двовимірних масивів :

- при оголошенні

```
float A[2][3] =
{
	{10 20 30},
	{40 50 60}
};
```

- в циклі розрахунку

```
float B[3][5];
for (int i = 0; i <= 2; i++)
{
	for (int j = 0; j <= 4; j++)
	{
		B[i][j] = i*j;
	}
}
```

Тривимірні масиви призначені для зберігання інформації, що задається об'ємним розподілом даних в просторі.

Потрібно відмітити, що в масивах в неявному виді реалізований механізм звернення до його елементів методом покажчиків.

Так, запис *dim[0]* вказує на перший елемент масиву, *dim[1]* - на другій і так далі. Покажчики будуть розглянуті пізніше, але факт їх застосування в масивах слід пам'ятати, і мати на увазі при використанні масивів як аргументів функції.

Передача масивів у функції. Масиви можуть бути використані як аргументи функцій. При цьому слід дотримуватися правил оголошення, виклику і визначення масивів:

```
//Оголошення функції з масивом як аргумент (прототип)
void function (int sqr [2][3]);

//Спрощене оголошення двовимірного масиву
void function (int sqr [][3]);

//Спрощене оголошення одновимірного масиву
void function (int sqr []);


//Виклик функції і передача масиву як аргумент
function (sqr); //вказується тільки ім'я масиву

//Визначення функції з аргументом-масивом
void function (int sqr[2][3])
{
}
```

У прикладах програм, розглянутих вище, змінні у функцію передавалися за значенням. При такому обміні створюється копія початкової змінної, яка потім бере участь в розрахунках. Якщо в програмі, наприклад, є масив з 200 значень, то при роботі програми буде створено 200 копій змінних, що входять у масив. Кількість копій значно зростає, якщо масив викликається в циклі розрахунку. При цьому пам'ять, виділена під запуск програми, переповнюється.

Для підвищення ефективності обміну даними необхідно передавати значення у функцію не в явному виді, а через покажчик. Покажчик - це змінна, в якій зберігається адреса розрахункової змінної. При цьому функція працюватиме не з копією, а безпосередньо з самою

змінною. У обміні даними покажчик не передається, функція відразу "дізнається", де знаходиться змінна.

У програмах, які будуть написані при вивченні курсу, покажчики використовуватимуться тільки при роботі з масивами.

Розглянемо приклад передачі масиву по покажчику. Для розробника програми необхідно виконати лише одну умову: при оголошенні функції, що приймає як аргумент масив, використовувати знак покажчика "*":

```
//Оголошення прототипу функції
extern int function (int*);
//Виклик функції
function (dim);
//Оголошення функції
int function (int* dimension);
```

У іншому операції з масивами нічим не відрізняються від звичайного їх використання, передача даних за допомогою покажчика реалізуються автоматично.

У наведених вище прикладах не вказано оголошення масиву, проте воно повинно бути зроблено у заголовному файлі до передачі масива у якості аргумента. Використання покажчиків дозволяє уникнути копіювання даних при передачі масиву. Посилальний механізм для масивів непридатний.

Приклад повної програми для пошуку максимуму в масиві:

```
//Файл program.h
float dimA[100];
float Amax;
int N, imax;
```

```
//Файл program.cpp
#include "program.h"
extern float fmax(float*, int) //оголошуємо прототип функції
void f1() //головна функція програми
{
for (int i = 0; i <= 90 ; i++)
        dimA[i] = i + 1/i;  //заповнення масиву даними

N = i; //запам'ятовуємо індекс останнього елементу в масиві
Amax = fmax(dimA, N); // виклик функції пошуку максимуму
}  //кінець функції f1


//Функція пошуку максимального значення масиву
float fmax(float* dim, int n)
{
imax = 0; //індекс максимального елементу дорівнює першому в масиві
float max = dim[imax]; //приймаємо максимальним перший елемент
for (int i = 0; i <= n; i++)
{
        if (dim[i] > max;) //якщо поточне значення більше
        //максимального
        {
        max = dim[i]; //максимальне значення дорівнює поточному
        imax = i; //запам'ятовуємо індекс максимального елементу
        }// кінець if
} //кінець циклу for
return max; //повертаємо знайдений максимум у функцію f1()
} //кінець функції fmax
```

Масив може бути використаний як поле класів. Ініціалізація даних в класі неможлива, тим часом, масиву при визначенні має бути вказаний конкретний розмір. Використання глобальних констант суперечить ідеології класу і не вирішує проблему. Можливі два способи: використання статичних констант або перерахувань для визначення розмірності масиву:

```
//оголошення константи усередині класу (не завжди компілюється)
static const int MAX = 10;
//використання нестандартного запису перерахувань
enum {MAX = 10};
```

Через покажчик можна також передавати імена файлів для функцій введення/виведення. Тоді функція виводу не буде жорстко прив'язана до одного текстового файлу. Розглянемо приклад.

```
//Файл program.h
#include <ofstream> //для операції файлового виводу
int x;

//Файл program.cpp
##include "program.h"
extern void Output(char*); // оголошуємо ф-ю виводу у зовнішній файл
using namespace std;
//Функція головної програми
void f1()
{
x = 10;
Output ("file 1.txt"); // виведення значення x у файл "file 1.txt"
x = 100;
Output ("file 2.txt"); // виведення значення x у файл "file 1.txt"
}
//Функція виводу у файл
void Output(char* f)
{
ofstream df(f); // оголошення файлової змінної df, відкрити файл
df  << x; виведення значення x у файл
close(df); // зруйнувати змінну df, закрити файл
} кінець функції Output
}//кінець файлу
```

У лістингу імена файлів "file 1.txt" і "file 2.txt" є аргументами функції файлового введення-виведення *Output()*. При зверненні до цієї функції відбувається підміна фіктивною змінною f іменами файлів, переданими як аргумент.

1.13 Об'єкти і класи

Клас, на відміну від масивів і структур, дозволяє об'єднувати не лише дані, але і методи, тобто функції. Клас є формою, що визначає, які дані і функції будуть включені в об'єкт класу.

Класи і створення об'єктів. При оголошенні класу не створюються його об'єкти, не виділяється пам'ять під змінні. Клас служить для організації представлення даних і методів (функцій).

Приклад простого класу.

```
//Визначення класу
class Motor
{
private: //Захищені дані
int ID;
float U;
public: //Відкриті дані (методи класу)
void set_data (int a, float b) //ініціалізація закритих полів класу
{ID = f; U = b;}
float show_data() //повідомлення класу
{return U;}
};
//Використання класу
Motor m1; //створення об'єкту (екземпляра) класу
m1.set_data(1, 380); //виклик функції для ініціалізації полів об'єкту
float U = m1.show_data(); //виклик функції для доступу до поля об'єкту
```

Конструктор. Поля об'єкту можна ініціалізувати автоматично при його створенні, без явного виклику методів класу. Метод, що реалізовує автоматичну ініціалізацію полів об'єкту, називається конструктором. Ім'я методу конструктора має бути таким же, як і ім'я класу. Модифікуємо попередній приклад з використанням конструктора.

```
class Motor
{
private: //Захищені дані
int ID;
float U;
public: //Відкриті дані (методи класу)
Motor (){} //Конструктор за умовчанням
Motor (int a, float b) : ID(a), U(b){} //Конструктор з ініціалізацією
void set_data (int a, float b) //ініціалізація закритих полів класу
{ID = f; U = b;}
float show_data() //повідомлення класу
{return U;}
~Motor() {}//Деструктор
};

//Використання класу
Motor m1; //створення порожнього об'єкту конструктором за
умовчанням
Motor m2(2, 220); //створення об'єкту з автоматичною ініціалізацією
m1.set_data(1, 380); //виклик функції для ініціалізації полів об'єкту
float U1 = m1.show_data();
float U2 = m2.show_data();
```

Конструктори, так само як і функції з різною кількістю аргументів, розглянуті вище, розрізняються компілятором. Можливо також використання порожніх конструкторів: об'єкт створюється, але не ініціалізується. Такий конструктор носить назву конструктора за умовчанням.

Методи класу можуть бути оголошені поза класом. Усередині класу визначається тільки прототип функції :

```
//Визначення класу
class Example
{
int a; //захищене поле (private за умовчанням)
public:
Example(int a1) : a(a1)
{}
int get_data(int); //прототип методу класу
~Example() // Деструкція
{}
}
//Реалізація методів класу
int Example:: get_data(int a1)
{
return a + a1;
}
```

Як правило, визначення класу розміщується в заголовному файлі з розширенням *.h, а реалізація методів класу - у файлі з розширенням *.cpp.

Деструктор. Деструктор - це особливий метод класу, який автоматично викликається при руйнуванні об'єкту. Деструктор має ім'я, яке співпадає з ім'ям класу з попереднім символом ~. Призначення деструктору - вивільнення пам'яті, виділеної конструктором при створенні об'єкту.

Приклад використання класів. Розглянемо приклад програми знаходження коріння квадратного рівняння з використанням класів:

```
//Опис класу, файл root.h
#include <cmath>
class Root
{
float a, b, c;
float root1, root2, D;
void get_root(Root&); //внутрішній метод класу (прототип)
```

```
public:
class Negative{}; //порожній клас для обробки виключень
Root(){}
Root(float a1, float b1, float c1) : a(a1), b(b1), c(c1)
{
D = b*b - 4*a*c;
if (D < 0)
        throw Negative(); //обробка виключень
        get_root(*this);
}
friend float X1(Root&); //дружній метод
friend float X2(Root&);//дружній метод
};


//Реалізація класу, файл root.cpp
#include "stdafx.h"
#include "root.h"
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

using namespace std;
void Root::get_root(Root& r)
{
root1(2*r.a);
root2(2*r.a);
}

float X1(Root& r)
{
return r.root1;
}


float X2(Root& r)
{
return r.root2;
}
```

```
//Застосування програми знаходження коріння квадратного рівняння
//Включення класу заголовку програми файлу *.cpp
#include "root.h"

//Тіло функції, в якій реалізовується розрахунок коріння
UpdateData();
try //блок виконується, якщо немає помилок
{
Root r(m_a, m_b, m_c); //створення об'єкту і його ініціалізація
m_x1 = X1(r);
m_x2 = X2(r);
UpdateData(FALSE);
}
catch(Root::Negative) //перехоплення виключення
{
AfxMessageBox("Коріння не знайдене! Дискримінант негативний");
}
```

Конструктор копіювання за умовчанням. Дія конструктора копіювання за умовчанням полягає в тому, що в результаті операції привласнення, поля одного об'єкту класу будуть скопійовані автоматично у відповідні поля іншого об'єкту цього ж класу.

1.14 Потоки і файли

Потоки - це загальна назва потоку даних. Найпоширенішим прикладом використання потоків є операції виводу у файл і читання інформації з файлу.

Потокові операції супроводжуються застосуванням операторів введення/виведення в потік << і >>.

Для використання операторів виводу в потік необхідно включити в програму заголовний файл <*iostream*>. Для підтримки файлових операцій додатково включається файл <*fstream*>.

Приклад програми запису в текстовий файл:

```
#include <iostream>
#include <fstream>
using namespace std;
ofstream df ("Report.txt");
float P2 = 15000;
df << "Номінальна потужність = " << P2/1000 << " кВт\n";
```

Приклад програми читання з текстового файлу:

```
#include <iostream>
#include <fstream>
using namespace std;

ifstream df ("data.txt");
int a;
CString txt;
while (!df.eof())
{
df >> a;
txt.Format("Завантажена змінна: %d", a);
m_list.InsertString(index, txt);
index++;
}
```

Контрольні питання

1. Назвати основні типи даних С++. Що таке перетворення типів? Навести приклади.

2. Привести арифметичні і логічні операції С++. Синтаксис оператора *if*.

3. Застосування оператора галуження *switch*.

4. Оператори організації циклу *for*, *while*, *do*. Особливості застосування кожного з методів.

5. Поняття масивів. Оголошення одновимірних і двовимірних масивів. Організація доступу до елементів масиву.

6. Збереження інформації в текстовому файлі. Відкриття файлу в бінарному виді. Які заголовні файли потрібні для файлових операцій?

7. Структура програми на С++. Поняття заголовного файлу. Структура багатофайлової програми на С++. Обмін даними між змінними, що знаходяться в різних файлах.

Практичні завдання

1. Написати програму округлення дійсних чисел з виділенням цілої і дробової частини. Округлення виконувати до цілих за правилами округлення.

2. Написати програму розрахунку коріння квадратного рівняння виду $ax^2+ + bx + c = 0$. Коефіцієнти a, b, c мають бути аргументами функції знаходження коріння. Виконати умову перевірки дискримінанта на негативне значення з використанням оператора *if*.

3. Написати програму розрахунку електромагнітного моменту за формулою Клосса. Перевірку поточного значення ковзання на негативне значення виконати за допомогою оператора *switch*.

4. Написати програму пошуку мінімального і максимального значення одновимірного масиву. Заповнення масиву даними виконати з використанням оператора *for*. Пошук мінімуму і максимуму організувати з використанням оператору *while*.

5. Написати програму, в якій розраховується значення електромагнітного моменту залежно від ковзання. Результати розрахунку моменту зберегти в масив і вивести в текстовий файл.

6. Написати програму розрахунку магнітної напруженості H по значеннях магнітної індукції, що завантажуються з текстового файлу в бінарній формі. Вважати, що у файлі даних знаходиться 100 значень магнітної індукції (таблиця розміром 10 x 10).

7. Написати програму розрахунку модуля цілого числа. Модуль числа повинен розраховуватися в допоміжній функції і повертати значення в головну програму.

РОЗДІЛ 2

СЕРЕДОВИЩЕ ПРОГРАМНОГО ПРОЕКТУВАННЯ VISUAL STUDIO

Для створення повноцінних Windows - застосувань існує середовище проектування Microsoft Visual Studio. Програмне середовище Visual Studio, орієнтоване на мову C++, є потужним і складним інструментом для створення 32-розрядних застосувань Windows. Ці застосування набагато перевершують як за об'ємом, так і по складнощі своїх попередників для 16-розрядної Windows і старіші програми, які взагалі обходилися без графічного інтерфейсу. Але, незважаючи на те, що об'єм і складність програм збільшуються, для їх створення вимагається не більше, а менше зусиль, принаймні для тих, хто правильно вибирає необхідні інструментальні засоби.

Саме таким інструментом є Visual Studio.Net. Оснащений набором різноманітних майстрів (Wizard), що формують програмний код, цей продукт дозволяє в лічені секунди створити цілком працездатне застосування Windows. Включена до складу Visual Studio бібліотека класів Microsoft Foundation Classes (MFC) вже стала фактично стандартом для розробників програм на мові C++. Візуальні засоби розробки інтерфейсу користувача перетворюють процес компонування різноманітних меню і діалогових вікон на просте і захоплююче заняття. Час, витрачений на вивчення цього продукту, сторицею окупиться при створенні першого ж проекту.

2.1 Введення в програмне середовище Microsoft Visual Studio

Практично жодне застосування Windows не обходиться без діалогового вікна, без текстових полів або кнопок. Діалогові вікна і елементи управління є важливими засобами інтерфейсу користувача. Використання стандартних елементів управління має безперечну перевагу, оскільки економить і час користувача, якому не треба наново вивчати технологію маніпулювання знайомими вікнами або засобами навігації, і час розробника.

Що б не сталося в Windows-машині (натиснення кнопки миші, клавіші на клавіатурі), усі ці події породжують повідомлення, які передаються в одне і більш вікна. Повідомлення - це основний елемент програмування в середовищі Windows. Visual Studio.Net полегшує створення програм, які перехоплюють повідомлення і обробляють їх.

У MFC прийняли як базову концепцію наступне припущення: будь-яка програма формує об'єкт, який припускає збереження у вигляді файлу. Саме цю сукупність інформації і позначили як документ. Представлення (View) - це один із способів перегляду документу. MFC пропонує набір класів, які можна наслідувати при створенні класів документів і класів представлень. В результаті, без втручання розробника, вирішуються, наприклад, такі поширені завдання програмування, як екранна прокрутка, вкладки вікон, кнопки, перемикачі та ін.

Виведення програми на екран значною мірою виконується автоматично класами представлення. Проте, багато що доведеться зробити самостійно. Зокрема, одним з таких завдань буде виведення графіка функції в діалогове вікно.

Деякі програми, такі як екранний калькулятор і вікно для обміну короткими повідомленнями по мережі, потрібні тільки тимчасово. Бі-

льшість же програм зберігають інформацію у файлах, а потім багаторазово відкривають і закривають їх в процесі модифікації раніше збережених документів. MFC дозволяє значно спростити програмування операцій файлового введення/виведення і розширити можливості операторів виводу в потік.

Завдання будь-якої системи автоматизованого проектування полягає у виконанні послідовності певних операцій, знаходження кращого варіанту розрахунку серед численних, генерування проектно-конструкторської документації. В курсі "САПР електромеханічних пристроїв" розглядається оптимальне проектування електричних машин, яке неможливо реалізувати без програмного забезпечення, що виконує алгоритми розробника. Visual Studio.Net значною мірою спрощує формування людино-машинного інтерфейсу, забезпечуючи введення даних в діалогові вікна, передачу їх в розрахункову програму оптимізації, виведення результатів розрахунку на екран і у файл, а також автоматичну генерацію креслярської документації.

Для отримання розширених відомостей по роботі в середовищі Visual Studio можна ознайомитися із спеціальною літературою [2].

2.2 Базові елементи інтерфейсу

У даному навчальному посібнику розглядаються тільки ті елементи середовища проектування, в яких найчастіше виникає необхідність при рішенні інженерних завдань.

На рис. 2.1 показана форма діалогового застосування з розміщеними на ньому елементами управління, а також панель інструментів, за допомогою якої створюються відповідні елементи:

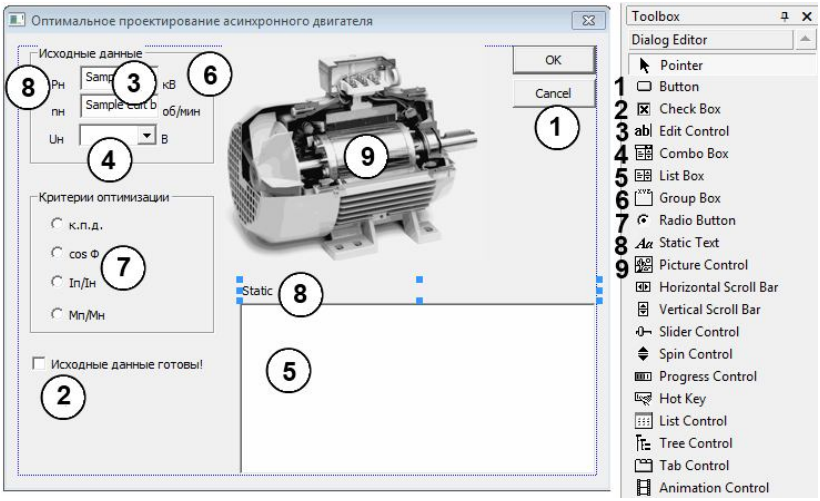


Рисунок 2.1 - Форма діалогового проекту

1 - кнопка (Button). У заготовці кожного діалогового вікна вже присутні кнопки OK і Cancel, але можна додати ще і свої;

2 - прапор стану (Check box); використовується для установки опцій, кожна з яких може бути вибрана незалежно від інших;

3 - текстове поле (Edit box); текстове поле може бути однорядковим або багаторядковим; сюди користувач може ввести текст або число - послідовність цифр – у якості даних для програми;

4 - поле із списком (Combo box); це комбінація текстового поля і списку; такий елемент управління дозволяє користувачеві не лише вибирати елементи з раніше підготовленого набору, але і самостійно поповнювати його, безпосередньо "друкувати" необхідний текст в текстове поле;

5 - список (List box); елемент цього типу використовується для вибору одного елементу із заздалегідь підготовленого набору; набір

може бути як жорстко встановленим на етапі розробки програми, так і мінятися програмно в процесі виконання застосування; головне - користувач по своїй волі не може міняти елементи в наборі; він може їх лише вибирати;

6 - рамка; використовується для об'єднання елементів, пов'язаних загальним сенсом; служить тільки для оформлення форми діалогового застосування;

7 - перемикач (радіокнопка); ці елементи управління використовуються для вибору одного варіанту з групи пов'язаних опцій; якщо вибрана одна з них, то інші вважаються невибраними;

8 - напис; по суті, це "неповноцінний" елемент управління, оскільки він використовується тільки як поле для виведення напису, що відноситься до "справжнього" елементу управління, розташованому по сусідству; тим часом, можна програмно міняти вміст напису;

9 - малюнок; елемент, що дозволяє виводити на форму діалогу заздалегідь підготовлені малюнки; також, за певних умов, може служити рамкою, що обмежує поле графіка функцій або довільного графічного об'єкту.

Розміщення елементів управління здійснюється простим перетягання необхідного елементу з панелі інструментів на форму діалогу. Детальний опис властивостей кожного з елементів буде розглянутий нижче.

2.3 Створення діалогового вікна

По-перше, необхідно відкрити середовище проектування застосувань Visual Studio. Створити новий проект, вибравши в меню File пункт New → Project (рис. 2.2)

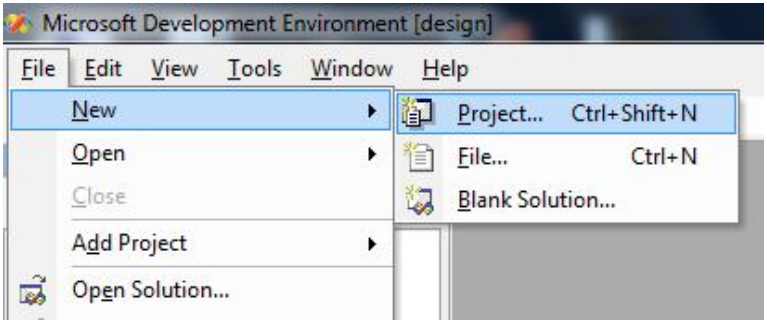


Рисунок 2.2 - Меню програми : створення нового проекту

Далі вибрати тип проекту MFC Application і ввести ім'я проекту (рис. 2.3).

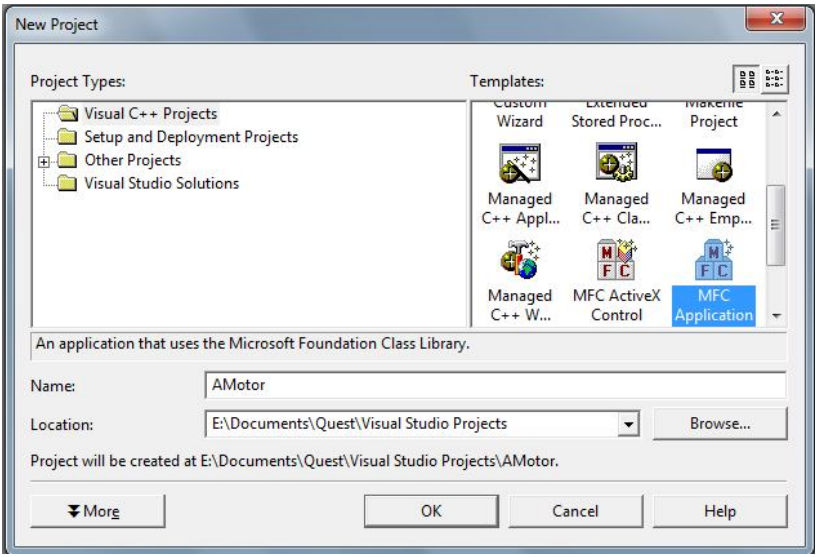


Рисунок 2.3 - Вибір типу і імені проекту

У майстрові (рис. 2.4), що з'явився, на вкладці завдання властивостей застосування (Application Type) вибрати тип додатка Dialog based і натиснути кнопку Finish.

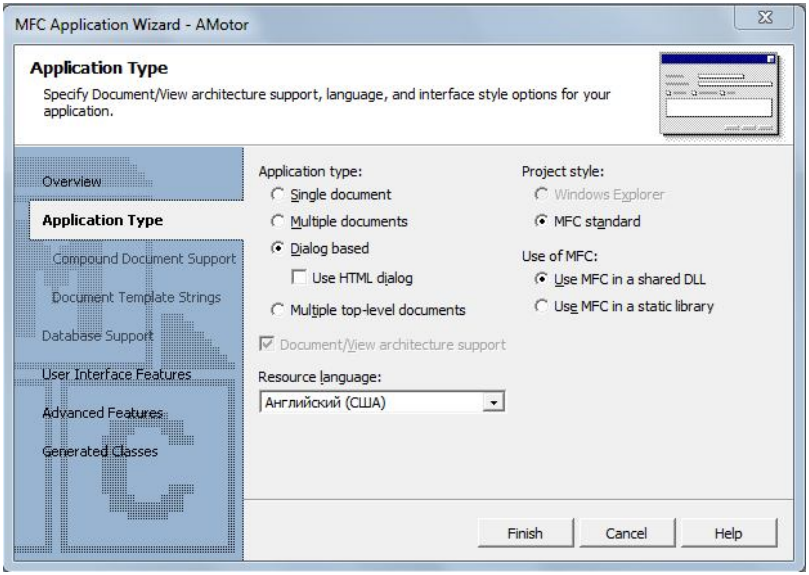


Рисунок 2.4 - Уточнення параметрів додатка

У вікні програми, що відкрилося, клацнути двічі лівою кнопкою миші (ЛКМ) на ідентифікаційному номері нашого діалогу IDD_AMOTOR_DIALOG (якщо у проекту інше ім'я, то його назва буде між IDD і DIALOG). При цьому відкриється редактор ресурсів, в якому з'явиться заготівля проекту (рис. 2.5).

На формі стандартно розміщуються дві кнопки (OK і Cancel) і елемент управління типу *напис* з вмістом "TODO : Place dialog controls here" в центрі. Її можна видалити.

Рекомендується виконати компіляцію застосування і здійснити його запуск, поки ми точно упевнені, що ніяких помилок в проект не було внесено. Запуск здійснюється кнопкою, показаною на рис. 2.5. Перший проект створено – за декілька хвилин!

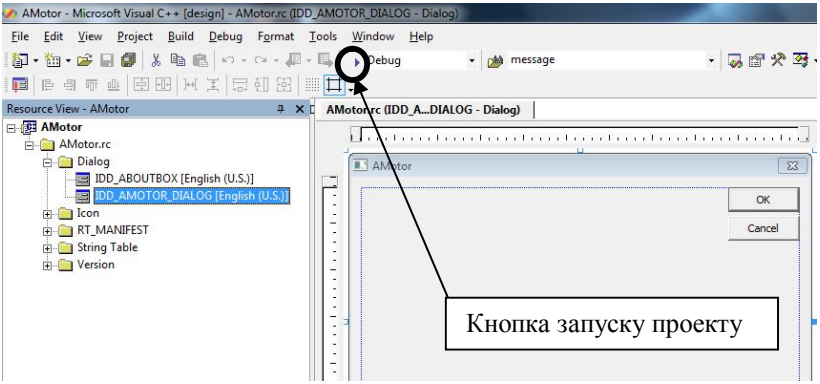


Рисунок 2.5 - Первинна форма діалогу

Дані проекту зазвичай зберігаються в теці Мої документи → Visual Studio Projects → <Ім'я проекту>. У теці *Debug* розміщується виконавчий файл проекту, що має розширення *exe*. У теці *Res* зберігаються ресурси проекту : малюнки, іконки, головний файл ресурсів. В корені теці проекту розміщуються заголовні (*.h) і програмні (*.cpp) файли.

Якщо виділити ЛКМ форму діалогу, то у вікні властивостей в полі Caption можна змінити стандартну назву, залежну від імені проекту (у прикладі AMotor), на необхідне (рис. 2.6). Є і простіший спосіб перейменувати діалог: при виділеній формі набрати потрібну назву на клавіатурі.

Створене застосування є повноцінною Windows програмою. Проте, функціональність її нульова: програма тільки закривається при натисненні на кнопки OK і Cancel, а також може вивести стандартне вікно довідки "Про програму" з вказівкою її поточної версії.

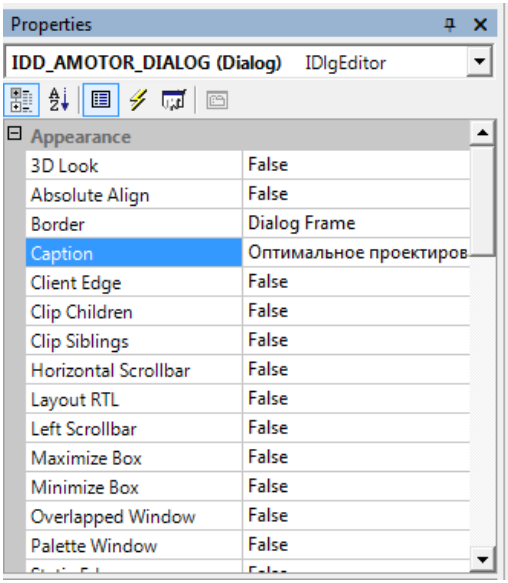


Рисунок 2.6 - Вікно властивостей форми діалогу

2.4 Стандартні елементи управління

Розглянемо детально усі елементи управління, показані на рис. 2.1 і перетворимо стандартну заготівлю на повнофункціональне застосування. Порядок розгляду елементів буде таким, яким він пронумерований на панелі інструментів рис. 2.1.

2.4.1 Кнопка. Для створення кнопки досить перетягнути елемент Button з панелі інструментів на поле діалогу. Якщо виділити ЛКМ отриману кнопку, то можна змінити її назву, просто набравши необ-

хідний текст на клавіатурі. Крім того, назву кнопки можна змінити у вікні властивостей в полі *Caption* (рис. 2.7). Вікно властивостей завжди відображує інформацію про властивості того елементу, який в даний момент виділений.

Крім того, у вікні властивостей можна змінити стандартний ідентифікаційний номер в полі ID на необхідний користувачеві (див. рис. 2.7).

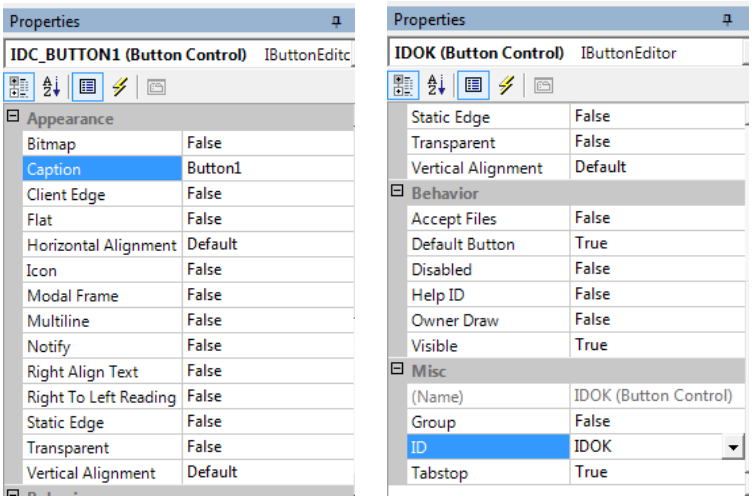


Рисунок 2.7 - Вікно властивостей кнопки : зміна назви і ID

Головне, щоб ідентифікаційний номер (далі ID) складався з рядкових букв англійського алфавіту або цифр після поєднання "IDC_". Виключенням є зарезервовані системою імена IDOK і IDCANCEL для стандартних кнопок діалогового застосування (діалогу). Ім'я ID будь-якого елементу, не лише даної кнопки важливо тим, що воно входить у назву функції, яка створюється середовищем Visual Studio, призначену для відробітку якої-небудь дії, пов'язаної з цим елементом.

Якщо ми хочемо, щоб при натисненні на кнопку виконувалася певна команда, наприклад виводилось інформаційне повідомлення, необхідно просто виконати на ній подвійне клацання ЛКМ. Якщо в полі ID стояло OK, то автоматично в проект додасться функція з ім'ям OnBnClicked**Ok**(), якщо ж просто Button1, то - OnBnClicked**Button1**().

Якщо таких кнопок багато, то іноді буває важко візуально зорієнтуватися, для яких дій кожна з них призначена. Персональні ID однозначно вирішують цю проблему. Лістинг функції, що виконується при натисненні на кнопку, приведений нижче.

```
void CPage1::OnBnClickedOk()
{
AfxMessageBox ("Тестове повідомлення");
}
```

На рис. 2.8 показаний результат виконання команди виведення інформаційного повідомлення *AfxMessageBox*, розміщеного у функції OnBnClickedOk().

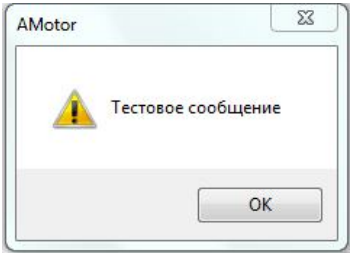


Рисунок 2.8 - Інформаційне повідомлення, отримане при натисненні на кнопку OK на формі діалогу

2.4.2 Прапор установки опцій. Перетягнемо на форму діалогу елемент Check box. Аналогічно, як було розглянуто для кнопки, можна задати ім'я напису, що знаходиться поряд з елементом Check Box: при виділеному елементі набрати необхідний напис на клавіатурі або задати його у вікні властивостей в полі Caption.

Елемент Check Box може набувати два логічні значення: Істина, TRUE (прапор встановлений) і Брехня, FALSE (прапор знятий). Для того, щоб прапори станів не лише візуально відображались на формі діалогу, але і були доступні програмі, необхідно значення цих станів присвоїти змінній.

Створимо змінну логічного типу і зв'яжемо її з елементом Check Box. Для цього необхідно натиснути праву кнопку миші (ПКМ) на елементі Check Box і в контекстному меню вибрати пункт Add variable (рис. 2.9).

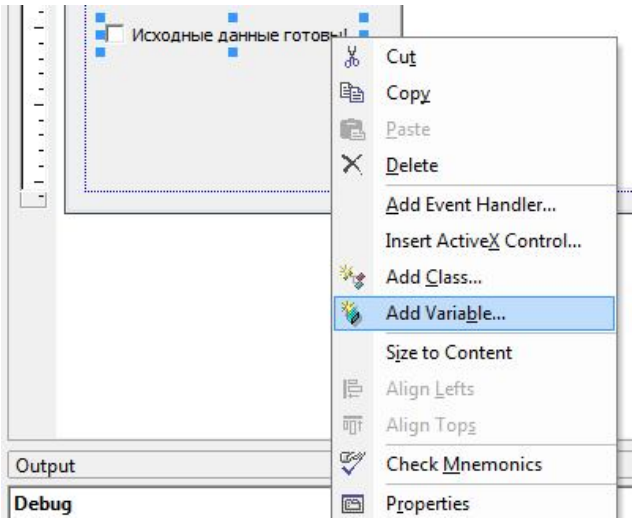


Рисунок 2.9 - Контекстне меню створення змінної

Відкриється вікно створення змінної в якому необхідно вказати тип даних (value), тип змінної (BOOL) і ім'я змінної. Ім'я усіх змінних, що зв'язують елементи управління, прийнято починати з префіксу "m_". (рис. 2.10).

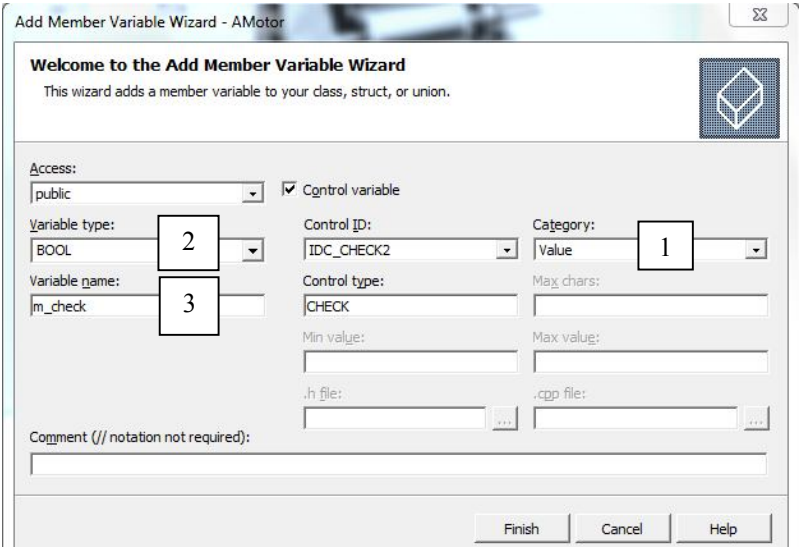


Рисунок 2.10 - Вікно створення змінної m_check

Для елементу типу "прапор" була створена змінна булевого типу з ім'ям m_check. Опис змінної заноситься автоматично в тіло класу проекту, що знаходиться в заголовному файлі.

Ініціалізація змінної m_check здійснюється в конструкторі класу діалогу, що знаходиться в програмному *cpp* файлі:

```
CAMotorDlg::CAMotorDlg(CWnd* pParent /*=NULL*/)
: CDialog(CAMotorDlg::IDD, pParent)
, m_check(FALSE)
```

За умовчанням, значення змінної встановлюється як FALSE. Це означає, що при запуску додатка прапор не буде встановлений. Якщо замість FALSE вписати TRUE в дужках ініціалізації змінної m_check, то при запуску програми прапор вже буде встановлений.

Але це було передача значень із змінної у вікно. Для здійснення зворотної дії, тобто програмного визначення прапора стану, необхідно просто перевірити те значення, яке зберігає в собі змінна m_check.

Розглянемо приклад, в якому при установці прапора кнопка ОК активуватиметься, а при його знятті - ставати неактивною.

Спершу, зробимо кнопку ОК неактивною. Для цього її необхідно виділити ЛКМ і в полі Disabled вікна властивостей встановити параметр True.

Зв'яжемо елемент прапор з реакцією на натиснення ЛКМ. Для цього, виділивши ПКМ елемент Check Box, в контекстному меню вибрати пункт Add Event Handler (рис. 2.11). У вікні, що відкрилося, просто натиснути на кнопку Add and Edit (рис. 2.12). Автоматично буде створена функція для обробки цієї події і буде виконаний перехід в тіло функції :

```
void CAMotorDlg::OnBnClickedCheck1()
{
// TODO: Add your control notification handler code here
}
```

Вставимо програмний код, який аналізує поточний стан прапора і по результату порівняння виконує активацію або деактивацію кнопки ОК.

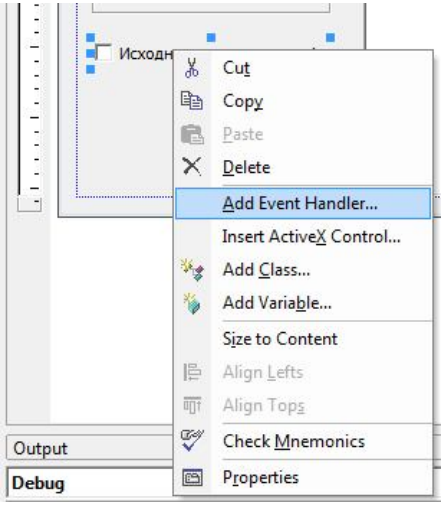


Рисунок 2.11 - Контекстне меню прив'язки подій

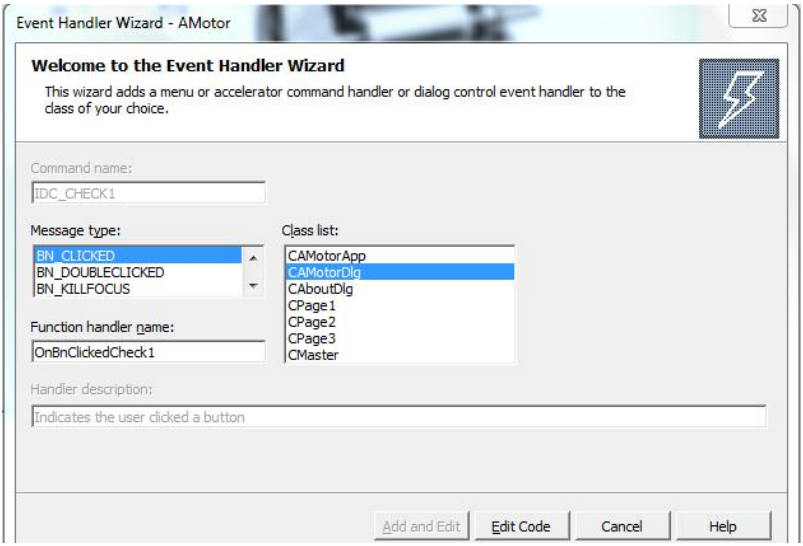


Рисунок 2.12 - Вікно створення функції обробки системних повідомлень

```
void CAMotorDlg::OnBnClickedCheck1()
{
UpdateData(); //Оновлення стану елементів вікна
switch (m_check)//Перевірка стану елементу m_check
{
case TRUE: // Якщо істина (прапор встановлений)
        //Активувати кнопку //IDOK
        GetDlgItem(IDOK) ->EnableWindow(SW_SHOW);
        break;
case FALSE: // Якщо брехня (прапор знятий)
        //Деактивувати кнопку IDOK
        GetDlgItem(IDOK) ->EnableWindow(SW_HIDE);
        break;
UpdateData(FALSE); //Відновити (перемалювати) вікно
}
}//кінець функції
```

При обміні даними між вікном і програмою розрахунковий код слід розміщувати між двома ключовими рядками функції UpdateData() з параметром TRUE спочатку (часто опускається) і FALSE наприкінці. Перша команда переписує значення з елементу вікна в пов'язану з ним змінну, друга команда (з параметром FALSE), навпаки, переписує значення змінної у вікно і перемальовує останнє.

Для активації кнопки ОК ми спочатку звертаємося до неї по ідентифікаційному номеру командою *GetDlgItem()*, а потім командою *EnableWindow()* встановлюємо значення SW_SHOW для активації і SW_HIDE для деактивації. Таким чином, ми програмно міняємо значення поля Disabled з вікна властивостей кнопки.

2.4.3 Текстове поле. Для створення текстового поля необхідно перетягнути елемент Edit control з панелі інструментів. Зв'язуємо елемент управління із змінною. Натисненням ПКМ викликаємо контекстне меню, в якому вибирається пункт Add variable. У вікні (рис. 2.13), що з'явилося, вибираємо категорію Value, необхідний тип змінної (у

прикладі - *float*) і задаємо ім'я змінної m_P (як раніше згадувалося, ім'я змінної повинне починатися з префіксу "m_"). Аналогічно створимо друге текстове поле із змінною m_n типу *float*.

Ініціалізація текстового поля даними (за умовчанням 0 для цифрових значень і пропуск для рядка) здійснюється в конструкторі класу діалогу :

```
CAMotorDlg::CAMotorDlg(CWnd* pParent /*=NULL*/)
: CDialog(CAMotorDlg::IDD, pParent)
, m_Pn(0) // ініціалізація текстового поля (потужність, кВт)
, m_n(0) // ініціалізація текстового поля (швидкість, об/хв)
, m_data(FALSE) // ініціалізація елементу Check Box
```

Як видно, до раніше створеної змінної прапору станів додалися дві нових змінних: m_P і m_n. Якщо замість нулів в дужках вписати необхідні чисельні значення, то відповідні змінні приймуть ці величини при запуску діалогу і з'являться в текстових полях.

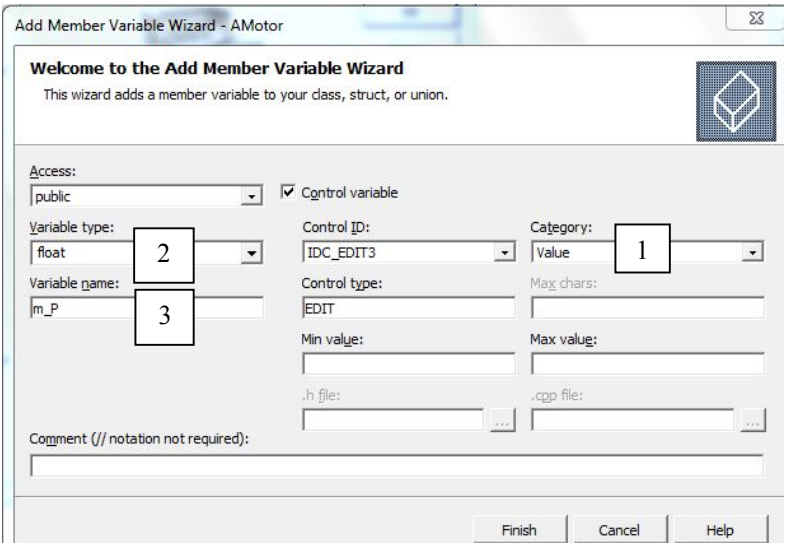


Рисунок 2.13 - Створення змінної елементу Edit control

Передача даних з текстового поля в пов'язану з ним змінну здійснюється за допомогою функції UpdateData(), а із змінної в текстове поле - функцією UpdateData(FALSE).

2.4.4 Поле із списком. Поле із списком створюється перетягненням елементу Combo box з панелі інструментів на форму діалогу. Зв'язуємо елемент управління із змінною. Натисненням ПКМ викликаємо контекстне меню, в якому вибирається пункт Add variable. У вікні (рис. 2.14), що з'явилося, категорію і тип змінної не міняємо, а задаємо лише її ім'я. У прикладі - це m_combo.

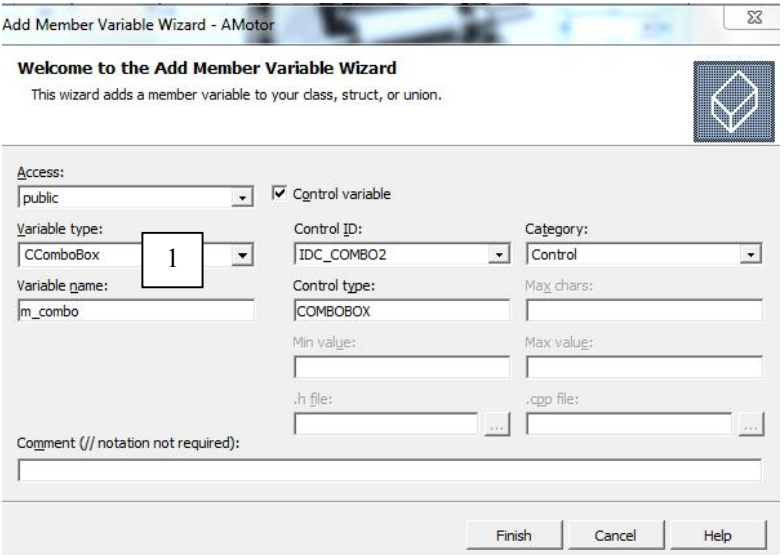


Рисунок 2.14 - Створення змінної для поля із списком

Ініціалізація елементи Combo box здійснюється у вікні властивостей. Для цього виділяємо ЛКМ поле із списком на формі діалогу, і через крапку з комою вносимо необхідні значення до поля Data (рис.

2.15). Також необхідно параметр сортування в полі Sort встановити в положення False.

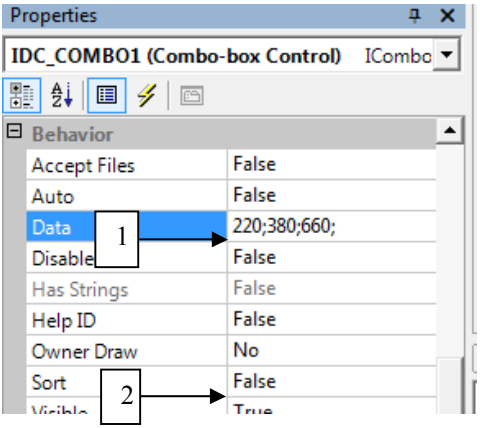


Рисунок 2.15 - Ініціалізація елементу Combo box

Для завдання висоти випадного списку необхідно клацнути на стрілці і розтягнути поле вниз за допомогою розмірного маркера (рис. 2.16).

Рисунок 2.16 - Завдання висоти випадного списку Combo box

Якщо зараз протестувати програму, то випадний список надасть користувачеві для вибору заздалегідь введені значення напруги (рис. 2.17).

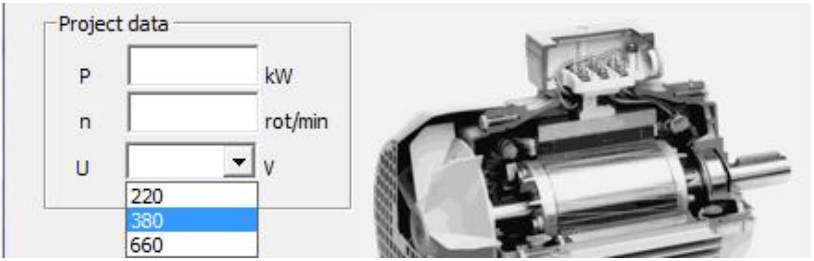


Рисунок 2.17 - Перевірка роботи випадного списку

При виконанні програми можна визначити номер вибраного рядка. Вміст полів отримати не можна. У прикладі в полі із списком відображається величина напруги, і присвоїти програмній змінній вибране значення можна двома способами:

а) використовуючи перемикач switch

```
UpdateData(); //Передаємо дані з вікна в пов'язані змінні
//отримуємо номер вибраного рядка в полі
int index = m_combo.GetCurSel();
switch (index) // перевіряємо номер вибраного рядка
{
case 0:
        U = 220;
        break;
case 1:
        U = 380;
        break;
case 2:
        U = 660;
        break;
}
```

б) звертаючись до елементу масиву з індексом, аналогічним вибраному в полі Combo box

```
UpdateData(); //Передаємо дані з вікна в пов'язані змінні
float dim_U[3] = {220;380;600}; //заздалегідь готуємо масив
int index = m_combo.GetCurSel(); //отримуємо номер вибраного рядка
U = dim_U[index];
```

2.4.5 Список. Список створюється перетяганням елементу List box з панелі інструментів на форму діалогу. Зв'язуємо елемент управління із змінною. Натисненням ПКМ на елементі викликаємо контекстне меню, в якому вибирається пункт Add variable. У вікні (рис. 2.18), що з'явилося, категорію і тип змінної не міняємо, а задаємо лише її ім'я. У прикладі - це m_list.

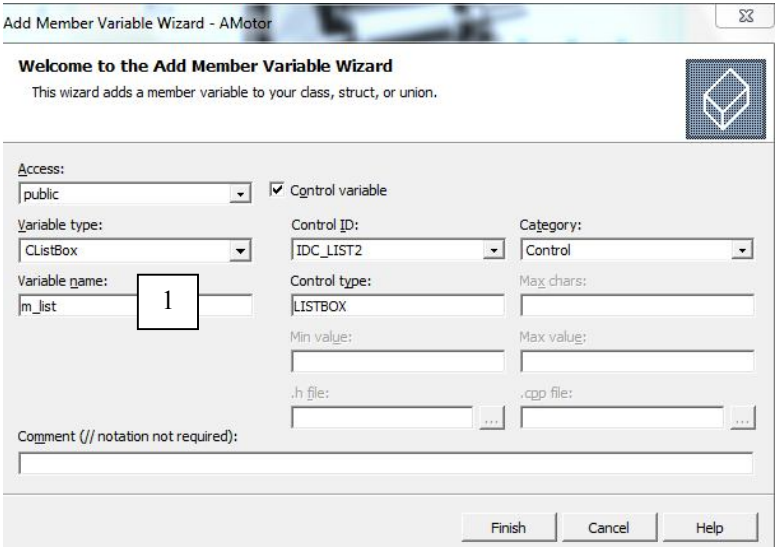


Рисунок 2.18 - Створення змінної елементу List box

Так само, як і в розглянутому в попередньому пункті елементу поля із списком, необхідно у вікні властивостей відключити сортування рядків у вікні властивостей, встановивши значення поля Sort в положення False.

У застосуваннях, що розглядаються в цьому навчальному посібнику, список використовуватиметься для виведення результатів розрахунку механічних і пускових характеристик асинхронного двигуна, тобто список динамічно заповнюватиметься рядками. Потрібно відмітити, що список можна створити із заздалегідь підготовленими рядками, для здійснення можливості їх вибору, як і в елементі Combo box. Проте тут це розглядатися не буде.

Нехай перед проектувальником стоїть завдання вивести N рядків із заздалегідь сформованого масиву A[] у список. Вирішимо її, як показано в лістингу нижче.

```
CString text; // оголошуємо строкову змінну
for (int i = 0; i <= N; i++) // організовуємо цикл від 0 до N
{
// форматування рядка виведення виду y(i) = A[i]
text.Format("y(%d) = %f", i, A[i]);
m_list.InsertString(i, text); // додавання рядка text у позицію i
}
UpdateData(FALSE); // оновлення діалогового вікна
```

Командою Format формується рядок *text*, в який замість символів %d (цілий тип) або %f (речовий тип) підставляються відповідні значення номера рядка *i* і значення елементу масиву *A[i]*, вказаних через кому. Функція *InsertString()* додає сформований рядок *text* в список з індексом *i*. Результат виконання приведеного коду може виглядати так, як показано на рис. 2.19.

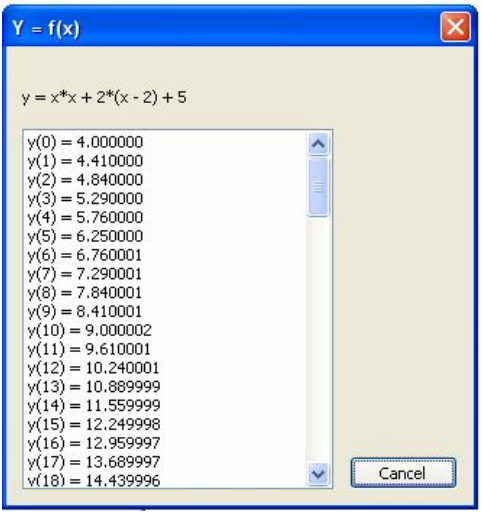


Рисунок 2.19 - Виведення рядків в елемент List box

2.4.6 Групуюча рамка. Рамка є найпростішим елементом оформлення інтерфейсу і служить для об'єднання пов'язаних загальним призначенням елементів управління. Для зміни назви рамки досить виділити її і набрати на клавіатурі необхідний текст або ж задати назву в полі Caption вікна властивостей.

2.4.7 Перемикач. Перемикач (радіокнопка) створюється перетягненням елементу Radio button з панелі інструментів на форму діалогу. Треба бути уважним в розміщенні елементів на формі діалогу - кнопки отримають порядкові номери, відповідні черговості їх створення. Для першого елементу Radio button у вікні властивостей встановлюємо значення поля Group в положення True. Зв'язуємо тільки перший елемент управління із змінною. Натисненням ПКМ на елементі викликаємо контекстне меню, в якому вибирається пункт Add variable.

У вікні (рис. 2.20), що з'явилося, задаємо категорію Value, тип змінної *int*, ім'я - m_radio.

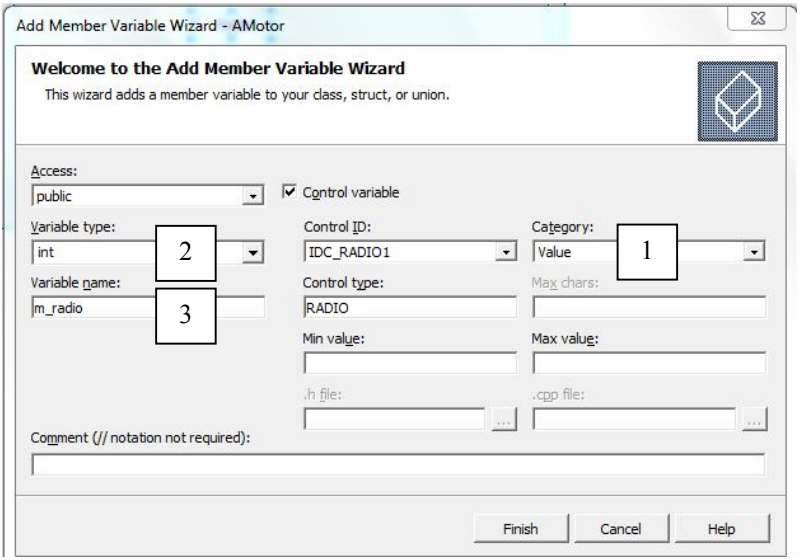


Рисунок 2.20 - Створення змінної елементу Radio button

При запуску застосування перший перемикач має бути вибраний. За умовчанням в конструкторі класу він ініціалізується нулем. Щоб виправити це, міняємо в секції ініціалізації *Radio buttion* 0 на 1:

```
CAMotorDlg::CAMotorDlg(CWnd* pParent /*=NULL*/)
: CDialog(CAMotorDlg::IDD, pParent)
, m_Pn(0) // ініціалізація текстового поля (потужність, кВт)
, m_n(0) // ініціалізація текстового поля (швидкість, об/хв)
, m_data(FALSE) // ініціалізація елементу Check Box
, m_radio (1) // ініціалізація елементу Radio button
```

У програмі можна визначити індекс вибраного у вікну перемикача таким чином:

```
int index;
UpdateData(); // Передача даних з вікна в пов'язані змінні
switch (m_radio) //Залежно від значення вибраного перемикача
{
case 0: // При виборі першого перемикача в групі
        index = 0;
        break;
case 1: // При виборі другого перемикача в групі
        index = 1;
        break;
}
```

2.4.8 Напис. Напис створюється перетяганням елементу Static text з панелі інструментів на форму діалогу. Найчастіше напис використовується тільки для записів пояснень, але його можна так само програмно змінювати які інші елементи. Для цього досить змінити стандартний ідентифікатор напису (усі створені написи мають один і той же ідентифікатор IDC_STATIC) на унікальний. У прикладі у вікні властивостей додамо до стандартного ID цифру 1 (рис. 2.21).

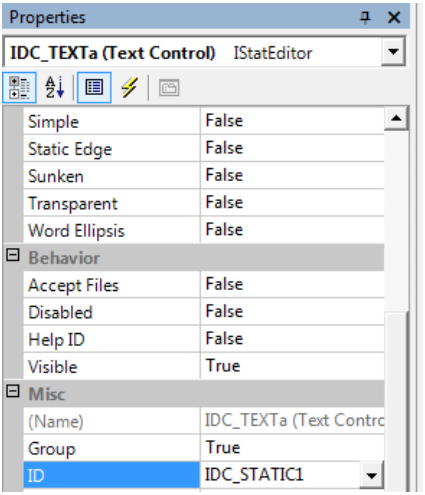


Рисунок 2.21 - Зміна стандартного ідентифікатора напису

Тепер для елементу Static можна створити змінну типу CString (рис. 2.22) і привласнювати їй потрібні значення. Головне, не забувати своєчасно вставляти функцію UpdateData(FALSE) перемальовування вікна після кожної зміни вмісту напису, інакше зроблені зміни не набудуть чинності (вони не будуть відображені у вікні).

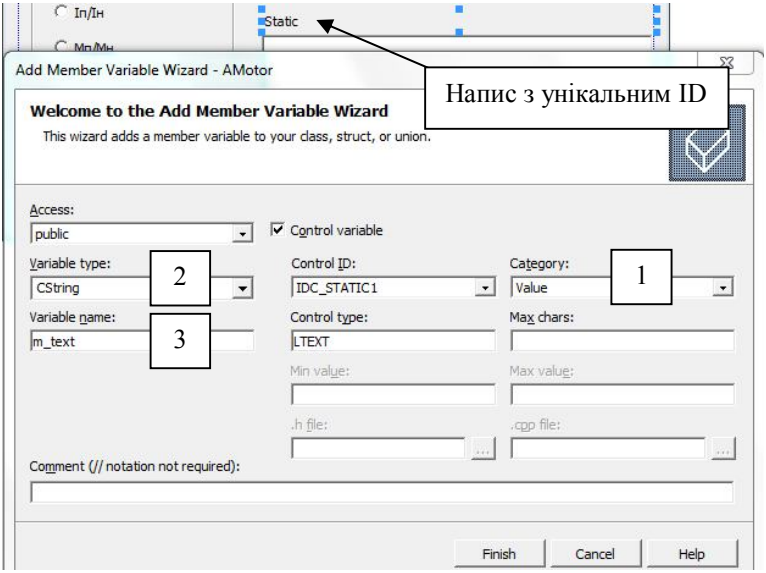


Рисунок 2.22 - Створення змінної для напису

2.4.9 Малюнок. Спочатку необхідно додати малюнок в проект. По-перше, малюнок має бути обов'язково збережений у форматі BMP в теці RES проекту. По-друге, в Visual Studio необхідно перейти на вкладку Resources і, натиснувши ПКМ на кореневій теці дерева ресурсів, в контекстному меню вибрати пункт Add Resource (рис. 2.23).

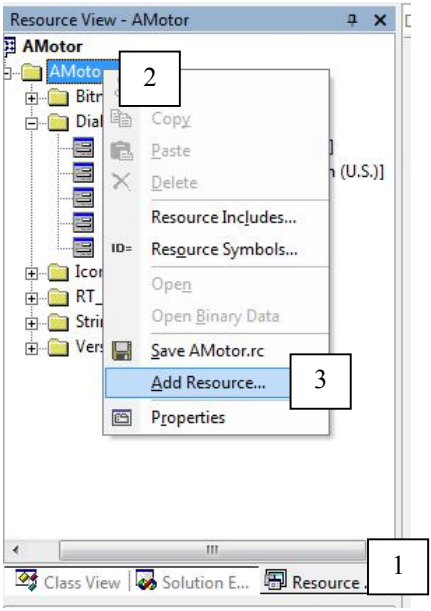


Рисунок 2.23 - Додавання нового ресурсу в проект

У вікні, що відкрилося, в дереві ліворуч вибрати Bitmap і натиснути на кнопку Import (рис. 2.24). Далі знайти збережений файл малюнка і додати його в проект.

Зверніть увагу: в дереві проектів з'явилася нова тека Bitmap, а в ній ресурс з ідентифікатором IDB_BITMAP1 - це і є наш малюнок (рис. 2.25). Доданий малюнок отримує унікальний ідентифікатор, який також можна побачити у вікні властивостей і при необхідності змінити. У прикладі стандартний ідентифікатор IDB_BITMAP замінений на IDB_MOTOR.

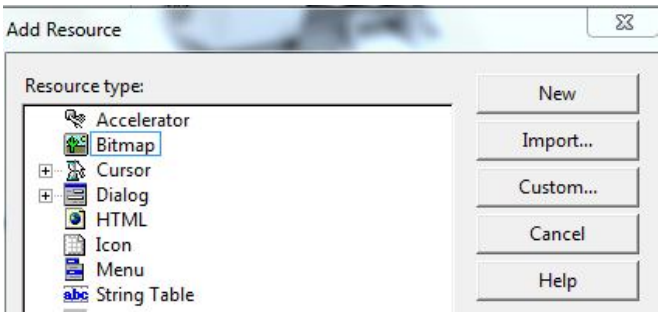


Рисунок 2.24 - Імпортування малюнка в проект

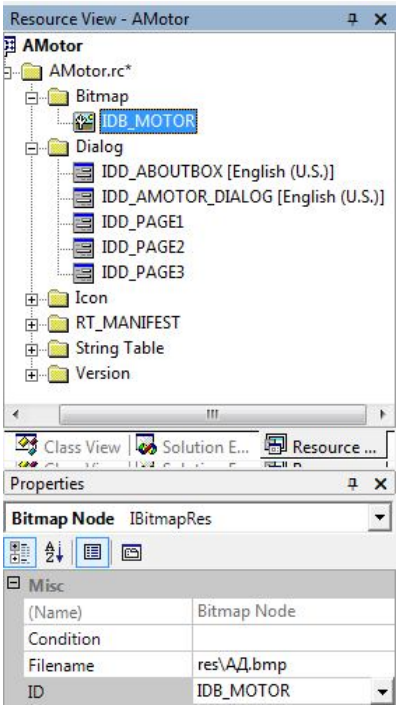


Рисунок 2.25 - Розміщення ресурсу Bitmap в дереві ресурсів проекту

Нарешті, після завершення підготовчих робіт, необхідно перетягнути елемент Picture Control з панелі інструментів на форму діалогу.

Не знімаючи виділення з елементу Picture Control, внести декілька змін у вікні властивостей (рис. 2.26) :

- змінити тип елементу в полі Type на Bitmap;

- вказати ідентифікатор малюнка в полі Image.

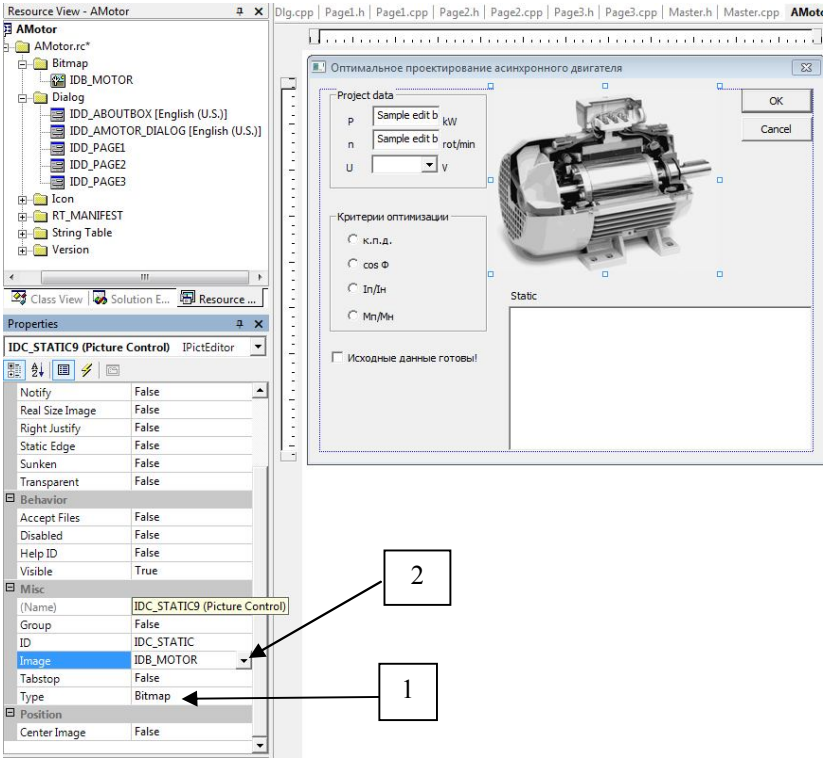


Рисунок 2.26 - Вікно властивостей елементу Picture Control

От і все, малюнок створений! Елемент Picture Control автоматично прийме розмір нашого малюнка.

2.5 Створення вікна-майстра і побудова графіка функції

Майстер (Wizard) є спеціалізованим вікном властивостей, вкладки якого мають кнопки "Назад", "Далі" і "Готово", призначені для внесення певного порядку до роботи користувача. Ця жорстка впорядкованість перетворює майстер в чудовий інструмент для організації проходження користувачем застосування послідовних етапів рішення комплексної задачі, зокрема, програми проектування електродвигуна.

Розглянемо на прикладі простий майстер, що складається з трьох сторінок, і який дозволяє розрахувати механічну характеристику асинхронного двигуна по відомій формулі Клосса.

На першій сторінці розмістимо інформаційні матеріали, на другій організуємо введення необхідних для розрахунку даних, а на третій - побудуємо графік залежності електромагнітного моменту від ковзання.

2.5.1 Створення сторінки властивостей. Для створення сторінки властивостей необхідно в Visual Studio перейти на вкладку Resource і, клацнувши ПКМ на теці Dialog, в контекстному меню вибрати пункт Insert Dialog (рис. 2.27).

У теці Dialog з'явиться новий запис з найменуванням IDD_DIALOG1 - це стандартний ідентифікатор діалогу. Для того, щоб було зручніше орієнтуватися в створюваних сторінках, тим більше, коли їх більше ніж дві, перейменуємо стандартний ідентифікатор на IDD_PAGE1 (поле ID вікна властивостей).

При виділеній формі створеної сторінки у вікні властивостей виконуємо наступні операції в строго вказаному нижче порядку (рис. 2.28):

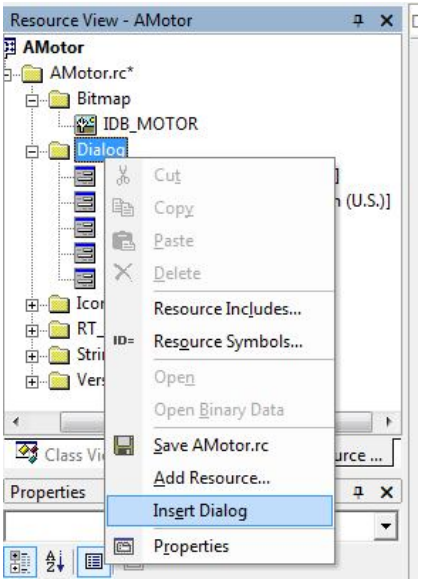


Рисунок 2.27 - Додавання нової форми діалогу в проект

а) видалити з форми сторінки автоматично додані елементи;

б) задати ім'я сторінки в полі Caption;

в) поле Border змінюємо з Dialog Frame на Thin.

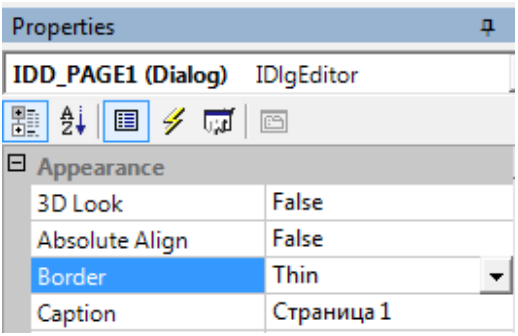


Рисунок 2.28 - Завдання властивостей сторінки майстра

Тепер прийшов час для сторінки властивостей створити власний клас. Для цього необхідно клацнути ПКМ на формі сторінки і в контекстному меню вибрати пункт Add Class (рис. 2.29).

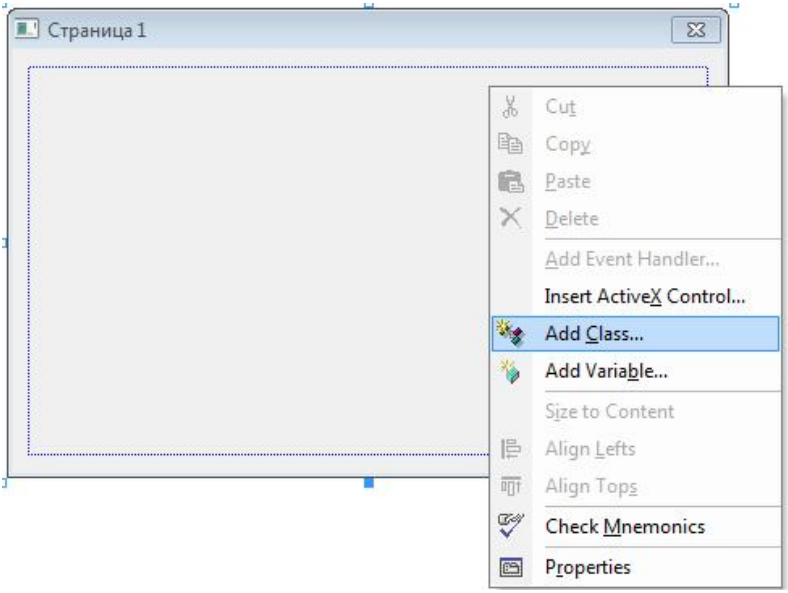


Рисунок 2.29 - Додавання в проект класу сторінки

У вікні, що відкрилося, задати ім'я нового класу CPage1 (повинно обов'язково починатися із заголовної англійської С) і базовий клас CProperty Page (рис. 2.30).

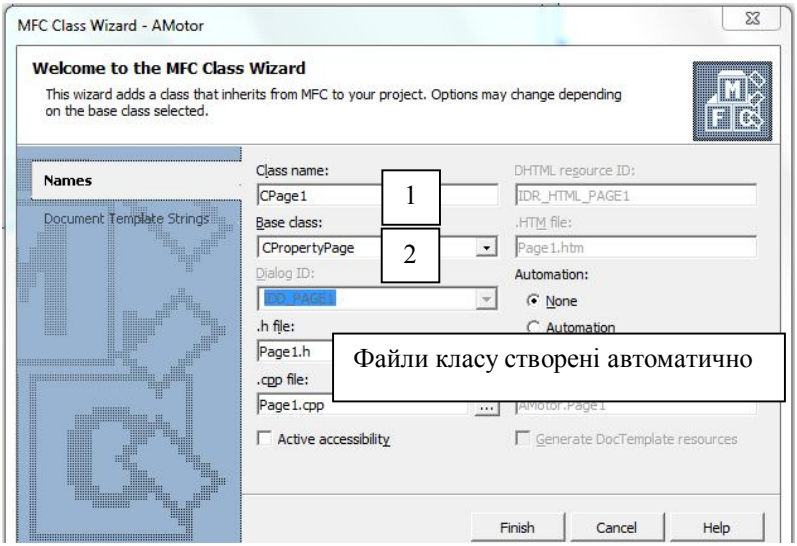


Рисунок 2.30 - Створення класу першої сторінки властивостей

Як видно з рис. 2.30, будуть автоматично створені два файли: Page1.h з описом класу сторінки і Page1.cpp для розміщення програмного коду, тобто реалізації класу.

Тепер необхідно виконати первинну ініціалізацію сторінки, а саме - визначити стан кнопок "Назад", "Далі" і "Готово" при запуску майстра.

На першій сторінці кнопка "Назад" має бути недоступною, а кнопка "Далі" - активною. Для здійснення цього треба перейти у файл Page1.cpp на вкладці Solution Explorer в Visual Studio, і натиснути на кнопку відкриття розширених функцій у вікні властивостей. Там знайти поле OnSetActive, і у випадному списку справа вибрати рядок <Add> OnSetActive (рис. 2.31).

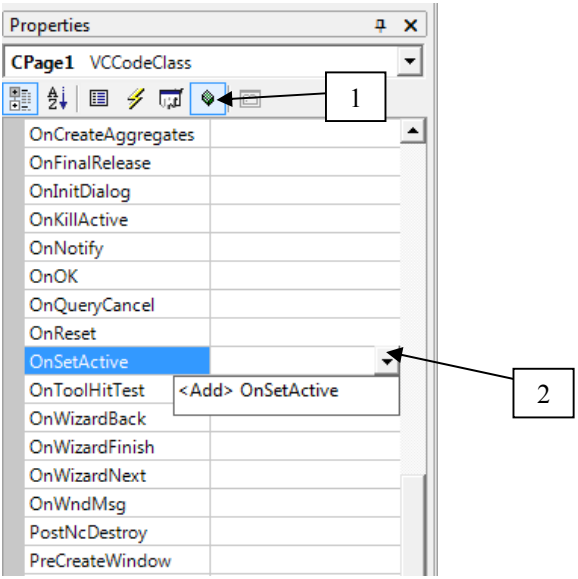


Рисунок 2.31 - Розширене вікно властивостей сторінки майстра

У файл Page1.cpp повинна автоматично додатися нова функція, вміст якої необхідно змінити так, як показано нижче:

```
BOOL CPage1::OnSetActive()
{
CPropertySheet* parent = (CPropertySheet*) GetParent();
parent ->SetWizardButtons(PSWIZB_NEXT);

return CPropertyPage::OnSetActive();
}
```

Аналогічні дії слід виконати для другої і третьої сторінок. Налаштування відрізнятиметься тільки установкою нових порядкових номерів сторінок і функцією OnSetActive().

Усі внутрішні сторінки майстра (у прикладі тільки одна - CPage2) повинні бути активованими кнопки "Назад" і "Далі":

```
BOOL CPage2::OnSetActive()
{
CPropertySheet* parent = (CPropertySheet*) GetParent();
parent ->SetWizardButtons(PSWIZB_BACK|PSWIZB_NEXT);

return CPropertyPage::OnSetActive();
}
```

На останній сторінці кнопку "Далі" необхідно заблокувати і зробити доступною кнопку "Готово" :

```
BOOL CPage3::OnSetActive()
{
CPropertySheet* parent = (CPropertySheet*) GetParent();
parent ->SetWizardButtons(PSWIZB_BACK|PSWIZB_FINISH);

return CPropertyPage::OnSetActive();
}
```

2.5.2. Створення класу-майстра. Тепер необхідно додати в проект клас, який об'єднає в собі створені сторінки. Для цього в Visual Studio в меню Project вибираємо пункт Add Class (рис. 2.32). У вікні, що з'явилося, вибираємо тип класу MFC Class і натискаємо кнопку Open (рис. 2.33).

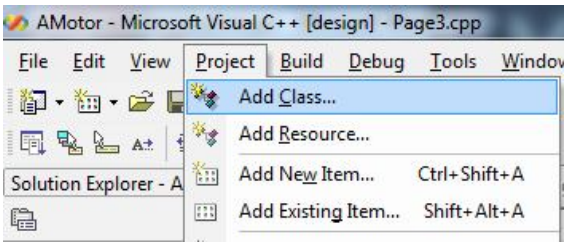


Рисунок 2.32 - Додавання нового класу в проект

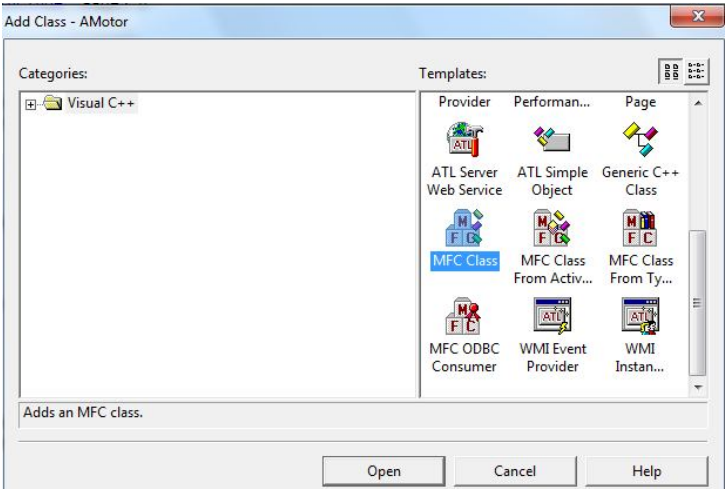


Рисунок 2.33 - Створення MFC класу

У вікні, що відкрилося, задаємо ім'я класу CMaster і базовий тип Property Sheet. При цьому будуть створено два файли: Master.h і Master.cpp (рис. 2.34).

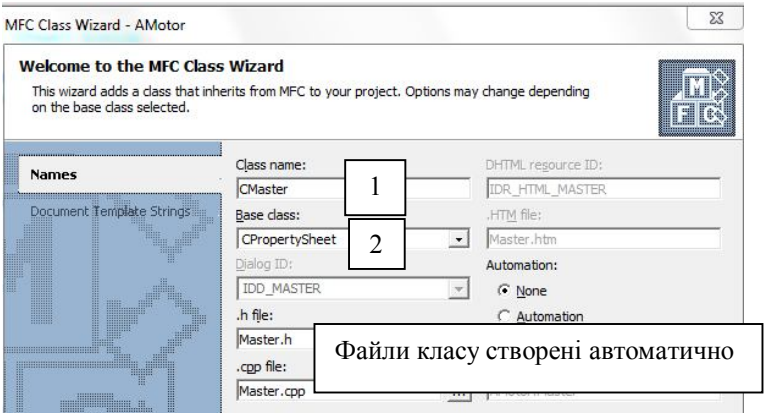


Рисунок 2.34 - Вікно створення класу-майстра

У Solution Explorer заходимо у файл Master.h і виконуємо реєстрацію сторінок :

а) вносяться зміни в секції директив препроцесору *#include* додаванням трьох рядків з включенням заголовних файлів сторінок майстра:

```
#include "Page 1.h"
#include "Page 2.h"
#include "Page 3.h"
```

б) створюємо змінні для трьох сторінок в секції *public* класу:

```
CPage1 m_page1;
CPage2 m_page2;
CPage3 m_page3;
```

У результаті, вміст файлу Master.h буде таким, як показано в лістингу нижче.

```
#pragma once
#include "Page 1.h"
#include "Page 2.h"
#include "Page 3.h"
// CMaster
class CMaster: public CPropertySheet
{
DECLARE_DYNAMIC(CMaster)
public:
CMaster(UINT nIDCaption,CWnd*pParentWnd=NULL,UINT iSelectPage = 0);
CMaster(LPCTSTR pszCaption, CWnd* pParentWnd = NULL, UINT iSelect-
Page = 0);
virtual ~CMaster();
CPage1 m_page1;
CPage2 m_page2;
CPage3 m_page3;
protected:
	DECLARE_MESSAGE_MAP()
};
```

Далі в Solution Explorer заходимо у файл Master.cpp і виконуємо додавання сторінок в майстер в конструкторові класу :

```
CMaster::CMaster(LPCTSTR pszCaption, CWnd* pParentWnd, UINT iSelect-
Page)
: CPropertySheet(pszCaption, pParentWnd, iSelectPage)
{
AddPage(&m_page1);
AddPage(&m_page2);
AddPage(&m_page3);
}
```

Отже, заготівлі сторінок сформовані. Тепер необхідно написати програмний код для запуску майстра. Передусім, для забезпечення доступу до класу майстра, додамо файл *Master.h* в cpp-файл проекту (у прикладі це *AMotorDlg.cpp*) командою *#include "Master.h"* на початку файлу.

Нехай, майстер відкриватиметься по натисненню на кнопку ОК головної форми діалогу. Виконавши в редакторові ресурсів подвійне клацання ЛКМ на кнопці ОК, переходимо у функцію, яка запускається на виконання при натисненні кнопки ОК (функція створюється у файлі проекту AMotorDlg.cpp).

Відредагуємо вміст цієї функції так, як показано нижче

```
void CAMotorDlg::OnBnClickedOk()
{
//Задаємо назву майстра
        CMaster PropSheet("Оптимальне проектування АД", this, 0);
//Перемикаємо відображення сторінки властивостей в режим "майстер"
        PropSheet.SetWizardMode();
//Запускаємо майстер
        int result = PropSheet.DoModal();
}
```

Якщо запустити застосування, то при натисненні на кнопку ОК відкриється майстер, показаний на рис. 2.35.

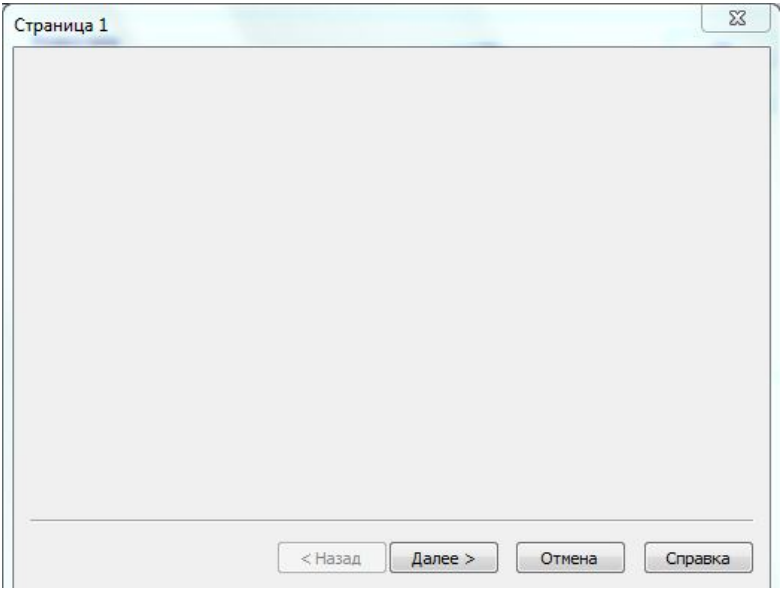


Рисунок 2.35 - Тестовий запуск застосування-майстра

2.5.3 Редагування першої сторінки майстра. Нехай на першій сторінці відображується інформаційне повідомлення про функції майстра і про майбутні розрахунки.

Перетягнемо на форму першої сторінки елемент Static text і в полі Caption вікна властивостей наберемо декілька рядків. Можна також додати рамку, скориставшись елементом Group box. Перша сторінка оформлена (рис. 2.36).

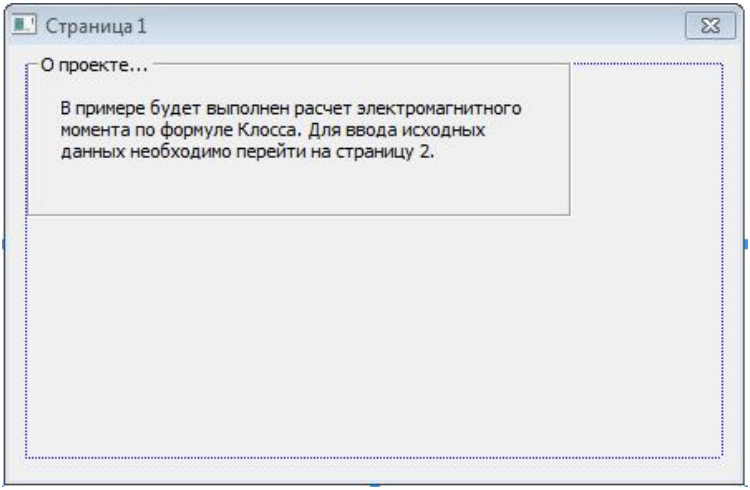


Рисунок 2.36 - Перша сторінка майстра

2.5.4 Редагування другої сторінки майстра. Електромагнітний момент, визначається по формулі Клосса:

$$M = \frac{2M_m}{\frac{s}{sm} + \frac{sm}{s}}.$$

Для виконання розрахунків необхідно задати значення критичного моменту M_m і критичного ковзання s_m, що ми і зробимо на другій сторінці.

Створимо на формі сторінки два елементи Edit Control і два написи Static Text як показано на рис. 2.37. Для першого текстового поля створимо змінну типу *float* з ім'ям m_M, а для другого - теж типу *float* з ім'ям m_s.

Тепер увага: розрахунки і побудова графіків виконуються на третій сторінці, тобто в іншому файлі! Організовуємо доступність даних, введених на сторінці 2 для сторінки 3. Механізм доступу детально описаний в п.2.8.5 і графічно продемонстрований на рис. 2.2 даного навчального посібника.

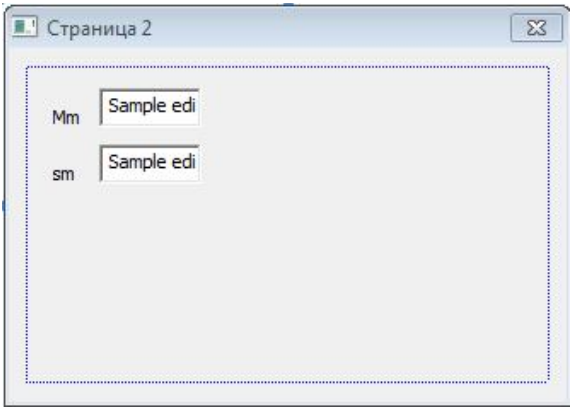


Рисунок 2.37 - Форма другої сторінки майстра

У Visual Studio перейдемо на вкладку Solution Explorer і створимо заголовний файл програми program.h. Для цього треба клацнути ПКМ на теці Header Files і в контекстному меню вибрати пункт Add, а потім New Item (рис. 2.38).

У вікні, що відкрилося, вказуємо тип файлу Header File і задаємо ім'я *program* в полі Name (рис. 2.39).

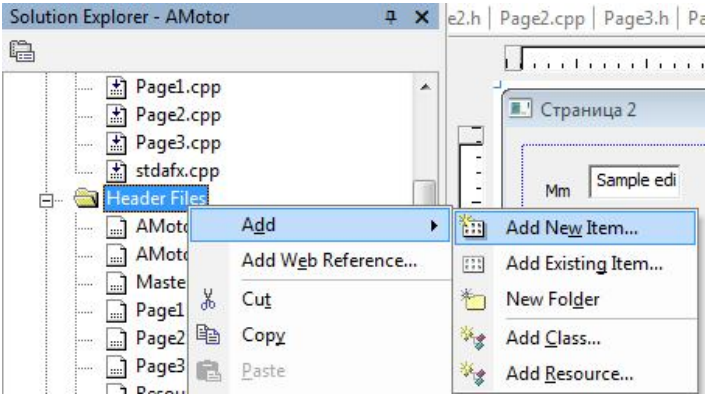


Рисунок 2.38 - Додавання нового файлу в проект

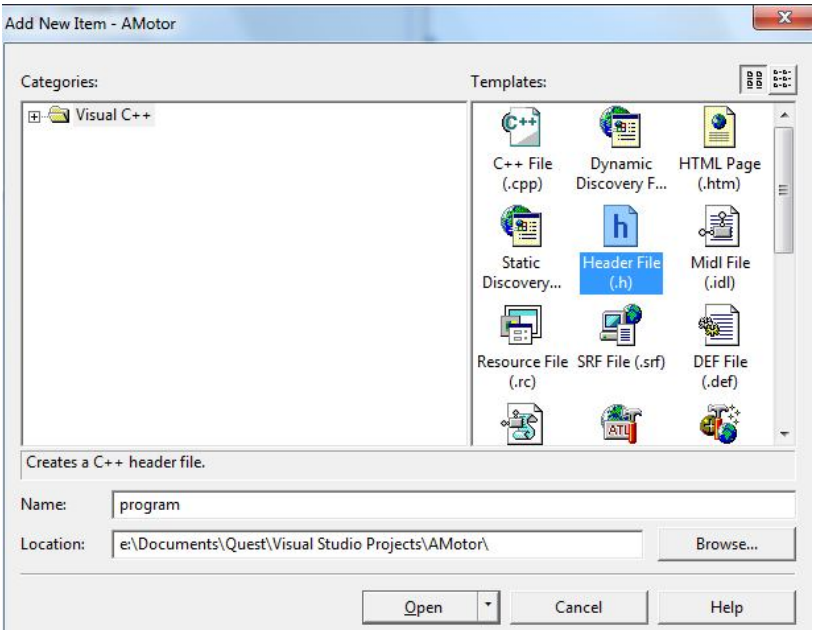


Рисунок 2.39 - Вікно створення заголовного файлу

У файл program.h вносяться рядки, наведені нижче:

```
//Program.h

float Mm; // максимальний момент
float sm; // критичне ковзання
float sx; // поточне ковзання
float smin; //мінімальне ковзання
float ds; //крок ковзання
float M[300]; // масив моменту
float s[300]; //масив ковзань
float Mmax; //максимальний момент
int Nmax; //кількість розрахункових точок
int imax; //індекс максимального значення моменту в масиві
```

Як видно, в заголовному файлі програми оголошуються усі використовувані в розрахунках і побудовах змінні.

У Visual Studio знову повертаємось на вкладку Solution Explorer і створимо вже відомим способом (див. рис. 2.38) розрахунковий файлу програми program.cpp.

У вікні, що відкрилося, вказуємо тип файлу C++ File і задаємо ім'я *program* в полі Name (рис. 2.40).

У файл program.cpp вноситься рядки, наведені нижче:

```
//Program.cpp
#include "stdafx.h"
#include "program.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

using namespace std;
```

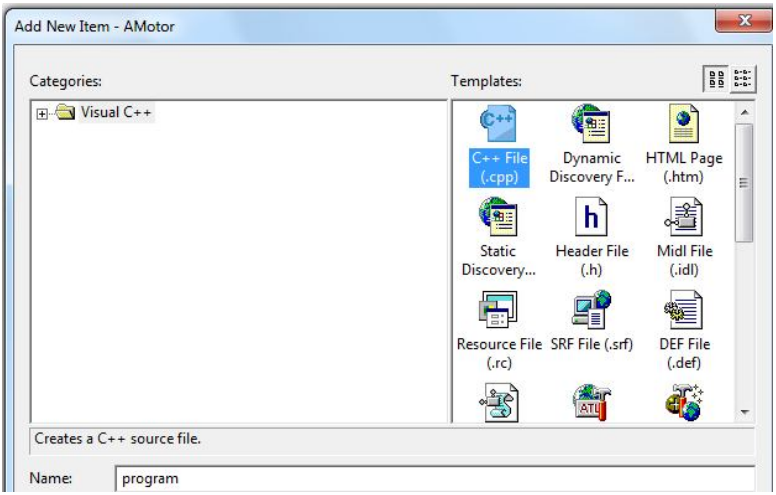


Рисунок 2.40 - Вікно створення cpp файлу

```
//Функція розрахунку електромагнітного моменту
void FindTorque()
{ //Ініціалізація змінних
smin = 0.01;
ds = 0.005;
sx = ds;
Nmax((1 - sx)/ds); //Визначення числа розрахункових точок
Mmax = 0;
//Обчислення електромагнітного моменту і внесення його в масив
for (int i = 0; i <= Nmax; i++)
{
        M[i] = 2*Mm/(sx/sm + sm/sx);
        s[i] = sx;
        sx += ds;
        if (M[i] > Mmax) //Визначення максимального значення масиву
        {
                Mmax = M[i];
                imax = i;
        }//кінець блоку порівняння
}//кінець циклу
}//кінець функції
```

Додаємо в проект заголовний файл *page2_ext.h*, який містить опис змінних і функції, що беруть участь в міжфайловому обміні сторінки 2:

```
//page2_ext.h
extern float Mm; // максимальний момент
extern float sm; // критичне ковзання
extern void FindTorque(); //розрахункова функція
```

У файлі Page2.cpp в секції *#include* включаємо файл зовнішніх змінних *page2_ext.h*:

```
#include "page 2_ext.h"
```

Тільки тепер змінні M_m і s_m оголошені у файлі *program.h*, а також функція *FindTorque()* стали доступними для другої сторінки майстра.

Нехай передача даних в розрахункову програму і обчислення механічної характеристики виконуються при натисненні на кнопку "Далі" майстра. Для цього переходимо у файл Page2.cpp і в розширених функціях вікна властивостей додаємо функцію *OnWizardNext* (рис. 2.41). У автоматично створену функцію вставляємо наступний код:

```
UpdateData(); // Переписуємо дані з вікна в змінні m_M і m_s
//Привласнюємо значення віконних змінних розрахунковим
Mm = m_M;
sm = m_s;
if (Mm <= 0 || sm <= 0) //Перевірка на введення неправильних значень
        AfxMessageBox ("Вхідні дані невірні або не введені");
else
{
        FindTorque(); //Виклик функції розрахунку моменту
        return CPropertyPage::OnWizardNext(); //Перехід на наступну стор.
}
```

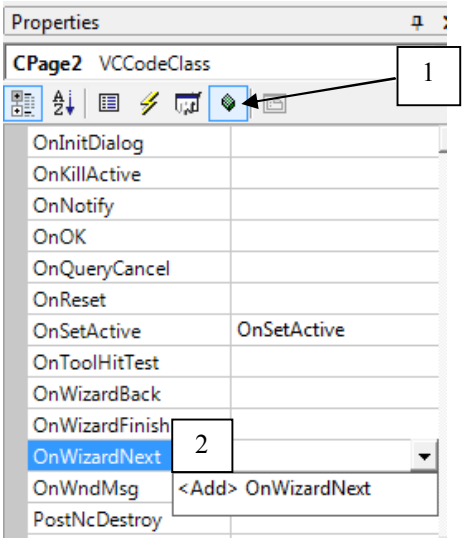


Рисунок 2.41 - Додавання функції для кнопки "Далі" майстра

З лістингу видно, що перехід на наступну сторінку програмно блокується, якщо були введені нульові або негативні значення критичного моменту або ковзання. Інформаційне повідомлення, а також зовнішній вигляд другої сторінки показані на рис. 2.42.

З рис. 2.42 також видно, що кнопка "Назад" тепер активна.

2.5.5 Побудова графіка функції. На третій сторінці буде виведений графік залежності електромагнітного моменту від ковзання, розрахованого на попередній сторінці.

Спочатку необхідно забезпечити доступ до розрахункових змінних і масивів, в яких зберігаються розраховані значення моменту і ковзання.

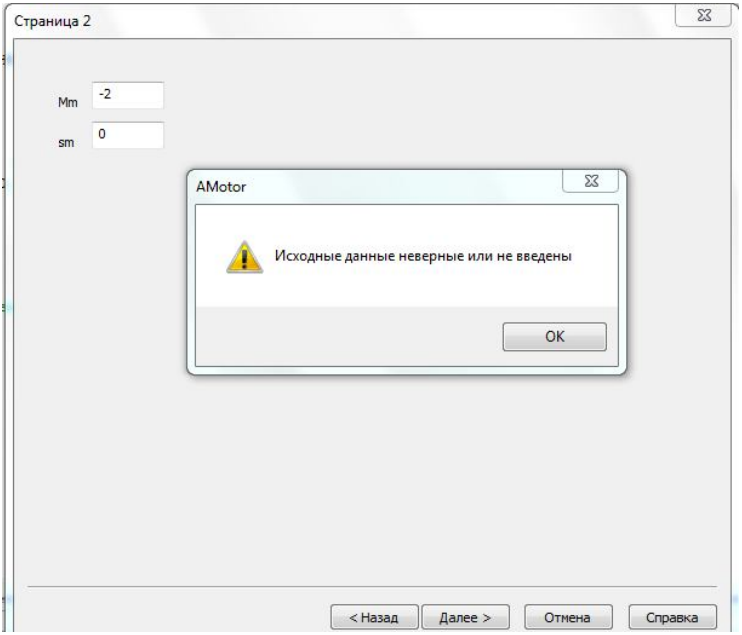


Рисунок 2.42 - Повідомлення, що викликається при невірних початкових даних

Для цього створимо файл зовнішніх змінних *page3_ext.h* і включимо його командою *#include* на початку файлу *Page3.cpp*. Лістинг файлу *page3_ext.h* наводиться нижче:

```
//page3_ext.h
extern float smin; //мінімальне ковзання
extern float Mmax; //максимальний момент
extern float M[300]; // масив моменту
extern float s[300]; //масив ковзань
extern int Nmax; //кількість розрахункових точок
extern int imax; //індекс максимального моменту в масиві
```

У лістингу як зовнішні оголошені тільки ті змінні, які беруть участь в міжфайловому обміні із сторінкою 3 і потрібні для побудови графіку.

Можливість побудови графіка забезпечується додаванням в проект файлів графічної бібліотеки *graph.h* і *graph.cpp*. Ці файли мають бути поміщені в кореневу теку проекту. Щоб підключити їх до ресурсів проекту, необхідно на вкладці Solution Explorer клацнути ПКМ для файлу *graph.h* на теці Header Files, для файлу *graph.cpp* - на теці Source Files, і в контекстному меню вибрати пункт Add, потім Add Existing Item (рис. 2.43).

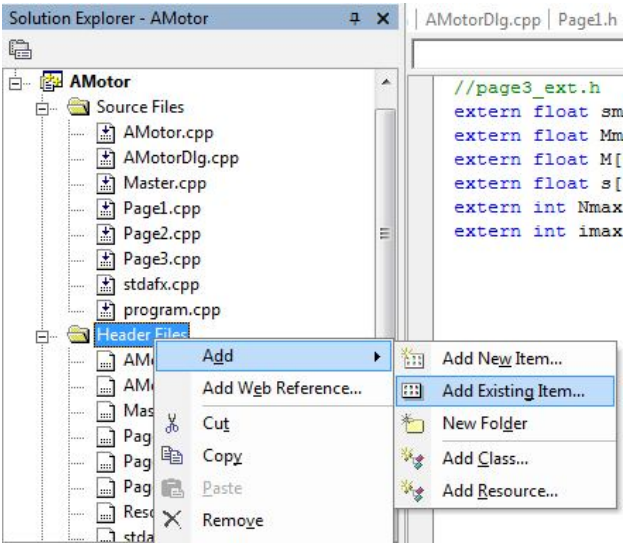


Рисунок 2.43 - Додавання існуючого файлу в проект

У вікні, що відкрилося, вибрати необхідний файл і додати його в проект.

Тепер треба перейти в заголовний файл третьої сторінки *Page3.h*, включити туди файл *graph.h* і створити змінні для побудови графіку.

У остаточному виді файл *Page3.h* повинен виглядати так:

```
#pragma once
#include "Graph.h" //Підключаємо графічну бібліотеку до діалогового вікна

// CPage3 dialog
class CPage3: public CPropertyPage
{
DECLARE_DYNAMIC(CPage3)
public:
CPage3();
virtual ~CPage3();
// Dialog Data
enum { IDD = IDD_PAGE3 };
protected:
virtual void DoDataExchange(CDataExchange* pDX);
DECLARE_MESSAGE_MAP()
public:
//Оголошення змінних графіка
double *m_pPlotItems;
LPG_FUNCTIONSTRUCT m_lpfs;
CGraph *m_pGraph;

virtual BOOL OnSetActive();
};
```

Створимо підпрограму побудови графіка *Draw_graph()*. Для цього переходимо в Visual Studio на вкладку Class View і після натиснення ПКМ на імені класу третьої сторінки CPage3 в контекстному меню вибираємо пункт Add, потім Add Function (рис. 2.44).

У вікні, що відкрилося, задаємо тип *void*, ім'я функції Draw_graph і в полі типу аргументу залишаємо порожнє місце (рис. 2.45).

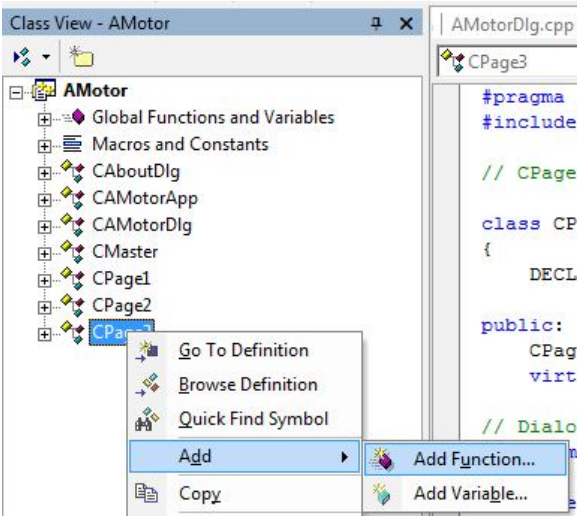


Рисунок 2.44 - Додавання нової функції в проект

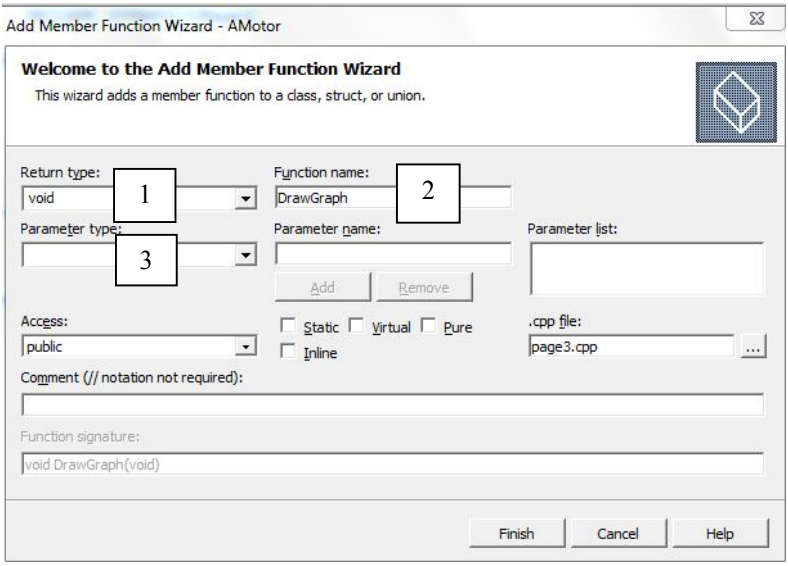


Рисунок 2.45 - Вікно налаштування нової функції

Код для функції Draw_graph:

```
CString s_y, s_x;
s_y = "M*"; // Вісь ординат
m_pGraph ->SetYLegendText(s_y);
s_x = "s"; // Вісь абсцис
m_pGraph ->SetXLegendText(s_x);
if(m_pPlotItems !=NULL)
{
        delete []m_pPlotItems;
        m_pPlotItems=NULL;
}
if(m_lpfs !=NULL)
{
        m_pGraph ->ClearFunction();
        delete m_lpfs;
}
UINT num_graphs = 1; // кількість графіків в одній системі координат
UINT numitems = Nmax; // кількість розрахункових точок
m_pPlotItems=new double[num_graphs*numitems*2];//кількість графіків
memset(m_pPlotItems, 0,4*numitems*2);
for(int i=0; i < numitems; i++) // заповнення масиву побудови даними
{
        (m_pPlotItems+i)[i]= (double)(s[i]);// дані аргументу
        (m_pPlotItems+i)[i+1]= (double)(M[i]); //дані функції
}
//установка ширини+висоти графіка (x0, y0, x1, y1)
m_lpfs =new G_FUNCTIONSTRUCT;
memset(m_lpfs, 0, sizeof(G_FUNCTIONSTRUCT));
m_lpfs ->FuncType=G_MULTIPLOTXY;
m_lpfs ->szGraphTitle="M = f(s)";
m_lpfs ->szXLegend="s";
m_lpfs ->szYLegend="M*";
m_lpfs ->xMax=1;
m_lpfs ->xMin=0;
m_lpfs ->yMax=Mmax;
m_lpfs ->yMin=0;
m_lpfs ->ChartType=G_LINECHART;
m_lpfs ->pPlotXYItems=m_pPlotItems;
m_lpfs ->num_PlotXYItems=numitems;
m_lpfs ->Const_1=1; //кількість графіків в одній системі координат
m_pGraph ->DoFunction(m_lpfs);
```

Для форматування графічного поля створюємо функцію SetGraph() типу *void*. Вставляємо код, як показано нижче.

```
void CPage3::SetGraph(void)
{
m_pGraph=new CGraph(this, 0,0,0,0, G_REDSCHEME);
m_lpfs=NULL;
m_pPlotItems=NULL;
// Установка теми графіка "синя"
m_pGraph ->SetColorScheme(G_BLUESCHEME, TRUE);
CString s_name;
s_name = "M* = f(s)"; // Назва графіка
m_pGraph ->SetGraphTitle(s_name);
//установка ширини+висоти графіка (x0, y0, x1, y1)
m_pGraph ->SetGraphSizePos(10,5,500,400);
}
```

Додаємо функцію отрисовки вікна *OnPaint()*. Пункт вибору повідомлення цієї функції вибирається у вікні властивостей файлу Page3.cpp в полі системних повідомлень WM_PAINT, як показано на рис. 2.46. Код цієї функції короткий і складається з одного рядка - виклику функції відображення графіку.

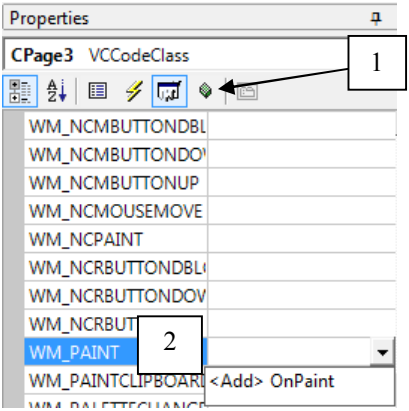


Рисунок 2.46 - Створення функції системних повідомлень OnPaint

Функція OnPaint() автоматично викликається всякий раз, коли графік перекривається яким-небудь об'єктом (вказівником миші, іншим вікном або коли поточне вікно згортається, а потім знову розгортається).

Лістинг функції OnPaint :

```
void CPage3::OnPaint()
{
CPaintDC dc(this); // device context for painting

m_pGraph ->PaintGraph();
}
```

Для очищення пам'яті при виході з програми додаємо функцію руйнування вікна графіка *OnDestroy()* при закритті сторінки. Функція додається у вікні властивостей файлу Page3.cpp в полі системних повідомлень WM_DESTROY так само, як і попередня функція *OnPaint()* (рис. 2.47).

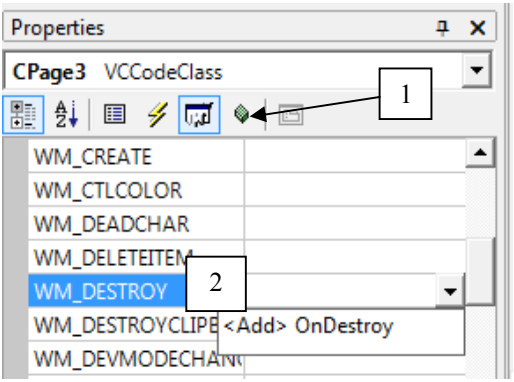


Рисунок 2.47 - Додавання функції очищення пам'яті при закритті вікна

Код функції *OnDestroy()* :

```
void CPage3::OnDestroy()
{
CPropertyPage::OnDestroy();
if (m_pGraph)
        delete m_pGraph;
if(m_lpfs)
        delete m_lpfs;
if(m_pPlotItems !=NULL)
        delete m_pPlotItems;
}
```

Останнє, що залишилося зробити - це створити функцію ініціалізації сторінки. Для створення такої функції треба у вікні властивостей файлу Page3.cpp в полі розширених функцій вибрати у випадному списку функцію <Add> OnInitDialog (рис. 2.48).

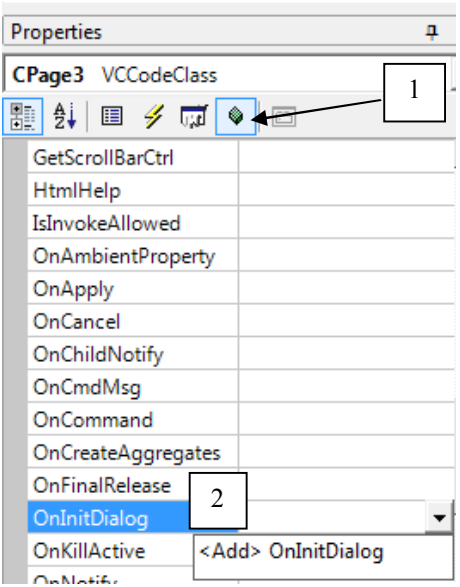


Рисунок 2.48 - Додавання функції ініціалізації сторінки

У функції OnInitDialog викликаються функції форматування поля графіка *SetGraph()* і відрисовки графіку *DrawGraph()*:

```
BOOL CPage3::OnInitDialog()
{CPropertyPage::OnInitDialog();
SetGraph();
Draw_graph();
return TRUE;
}
```

На рис. 2.49 показана третя сторінка майстра з побудованим графіком функції моменту від ковзання при $M_m = 2.2$ і $s_m = 0.3$.

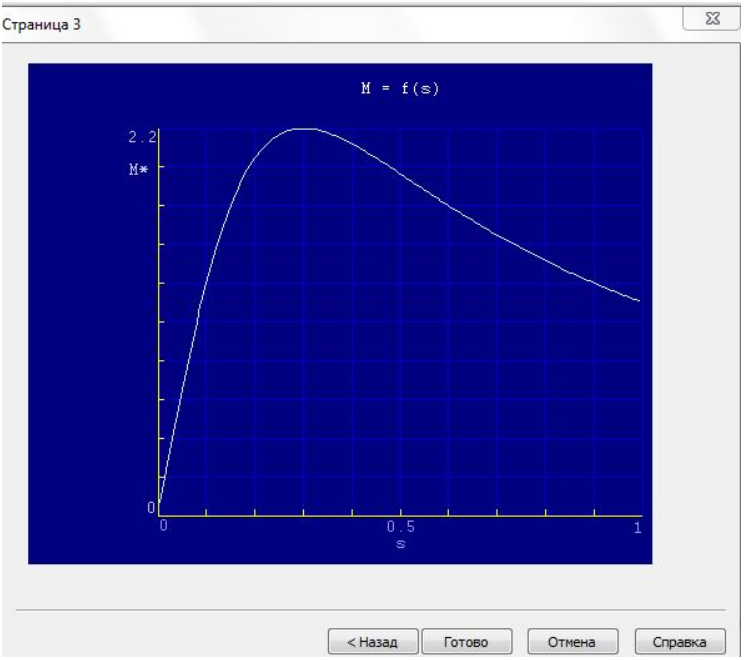


Рисунок 2.49 - Сторінка майстра з графіком M = f(s)

З рис. 2.49 видно, що кнопка "Далі" змінилася на кнопку "Готово". При натисненні на останню, майстер закриється.

Контрольні питання

1. Як створити діалогове застосування? Структура проекту в Visual Studio.

2. Як виконати ініціалізацію змінних діалогу?

3. Дати коротку характеристику одному з елементів управління:

- радіокнопка radio button;

- кнопка button;

- прапор check box;

- текстове поле Edit control;

- список List box;

- поле із списком Combo box;

- напис Static text.

4. За допомогою якої команди і як можна активувати/деактивувати елемент управління на формі діалогу?

5. Як додати на форму діалогового застосування малюнок?

6. За допомогою якої команди і як можна вивести інформаційне повідомлення?

7. Як створити застосування-майстер? Які класи при цьому створюються?

8. Створення і виклик майстра Wizard. Налаштування кнопок "Назад", "Далі", "Готово". Як обмежити до них доступ? Як організувати перехоплення повідомлення при натисненні на кнопку "Готово"?

9. Як підключити до проекту зовнішній "cpp"-файл з кодом розрахункової програми?

10. Як підключити заголовний файл? Його призначення.

11. Як забезпечити доступ до змінних, оголошених в іншому файлі?

Практичні завдання

1. Створити діалогове застосування, в якому в список List Box необхідно вивести результати знаходження коріння квадратного рівняння виду $y = ax^2 + bx + c$. Коефіцієнти рівняння a, b, c повинні вводитися в текстові поля. У разі негативного дискримінанта забезпечити виведення застережливого інформаційного повідомлення.

2. На формі діалогового застосування створити поле із списком Combo box, заповненого цілими числами від 1 до 10. При натисненні на кнопку "Прийняти" в напис Static text повинне виводитися розраховане значення швидкості поля в об/хв залежно від числа пар полюсів, обраного в полі Combo box.

3. На формі діалогу розмістити три перемикача radio button. При натисненні на кнопку "Прийняти" вивести інформаційне повідомлення з текстом: "Обраний пункт N", де замість N повинен підставлятися номер обраного у вікні перемикача.

4. У діалоговому застосуванні, що містить прапор check box і кнопку, остання повинна активуватися/деактивувати залежно від стану прапора.

5. Ґрунтуючись на матеріалі, викладеному в навчальному посібнику, на формі діалогу побудувати графік функції виду $y = ax^2 + bx + c$. Коефіцієнти рівняння a, b, c повинні вводитися в текстові поля на формі діалогу. Також необхідно створити два текстові поля, куди треба буде вводити мінімальне і максимальне значення аргументу. У разі негативного дискримінанту забезпечити виведення застережливого інформаційного повідомлення.

РОЗДІЛ 3

AUTOCAD І СЕРЕДОВИЩЕ ПРОЕКТУВАННЯ VISUAL LISP

Відповідно до загальної ідеології системи AutoCad, головне її призначення - не малювання креслень на комп'ютері (це призводить до падіння продуктивності праці конструктора в 2…3 рази), а створення на її основі спеціалізованої САПР певного класу виробів. Такі САПР різко, в 15…20 разів підвищують продуктивність праці проектувальника [3].

Аналіз роботи конструкторсько-технологічних служб ряду промислових підприємств дозволив встановити, що одна з найбільш трудомістких проектних процедур в ході конструкторсько-технологічному опрацюванню - розробка конструкторської документації ряду близьких по конструкції деталей і/або складальних одиниць, що відрізняються в основному своїми розмірними параметрами або варіантами виконання [3]. Ця процедура є трудомістким і нетворчим процесом з низькою продуктивністю і високою вірогідністю внесення помилок. Особливо часто вимагається випускати конструкторську документацію на засоби технологічного оснащення машинобудівного виробництва: лещата, кондуктори, прес-форми і так далі, причому підготовка цієї документації повинна вестися випереджаючими темпами для забезпечення часу на виготовлення засобів технологічного оснащення до моменту запуску виробу у виробництво.

Параметричне проектування дозволяє перетворити AutoCad, як засіб для малювання креслень на комп'ютері, в САПР певного класу виробів [4], надаючи можливість без втручання конструктора створю-

вати креслення виробу після проведення розрахунків в проектній програмі.

У навчальному посібнику спочатку розглянемо базові принципи роботи в середовищі автоматизованого проектування AutoCad, а потім перейдемо до вивчення основ мови програмування AutoLisp.

3.1 Графічне середовище САПР AUTOCAD: налаштування інтерфейсу

Перед розглядом системи автоматизованого проектування AutoCad приймемо наступні положення:

а) вивчення інтерфейсу AutoCad здійснюється на повністю англійській версії не нижче ніж 2004;

б) користувач вже має навички креслення в програмі AutoCad [5].

Завдання матеріалу, що викладається - організувати роботу проектувальника у напрямі створення САПР програм.

Налаштування робочої області. Підготовка робочої області починається з вікна налаштувань креслення. Викликати меню опцій можна, натиснувши ПКМ на робочій області і вибравши відповідний пункт Options контекстного меню.

Вкладка Display. Натиснути на кнопку Colors і для фону (background) вибрати нейтральний колір, наприклад RGB (254, 252, 240). Crosshair size (розмір перехрестя) встановити рівним 5.

Вкладка Drafting. Вибрати невеликий розмір маркера AutoSnap Marker Size, вибрати червоний колір маркера. Розмір Aperture Size також вибрати невеликим. Якщо розмір перехрестя буде великим, то при

досить щільному кресленні з'являється можливість помилкового виділення близько розташованих ліній.

Формат документу. Форматування документу визначається призначенням креслення. Приведений нижче опис відноситься до машинобудівного креслення. Помилкове форматування документу може привести до повного спотворення креслення, генерованого скриптовыми командами. Для виклику вікна форматування документу необхідно на робочій області натиснути ПКМ, утримуючи клавішу Shift. У контекстному меню вибрати пункт "Osnap Settings".

Вкладка Snap and Grid. У полях Snap X spacing, Snap Y spacing, Grid X spacing, Grid Y spacing встановити значення 1. Поля Snap відповідають за розмірність сітки прив'язки, а поля Grid - за розмірність сітки креслення. При розмірі поля рівному 1, поле креслення перетворюється на "міліметрівку".

Вкладка Object Snap. На вкладці усі прапори мають бути зняті. Ця сторінка є налаштуванням автоматичної прив'язки. Для автоматизованого креслення автоматична прив'язка є перешкодою.

Одиниці креслення. Налаштування розмірності креслення здійснюється з меню Format командою Units. На рис. 3.1 показані налаштування вікна.

Стиль документу. Налаштування стилю документу дозволяє перетворити креслення до виду, що відповідає вимогам ЄСКД. Різні налаштування стилю викликаються з меню Format.

Текст. Пункт меню Format - Text Style (стиль тексту). Вибираємо шрифт isocp.shx, вказуємо висоту тексту Height 0, кут нахилу Oblique Angle 15, інтервал Width Factor 1 і створюємо новий стиль тексту з назвою "ЄСКД", натиснувши на кнопку "New".

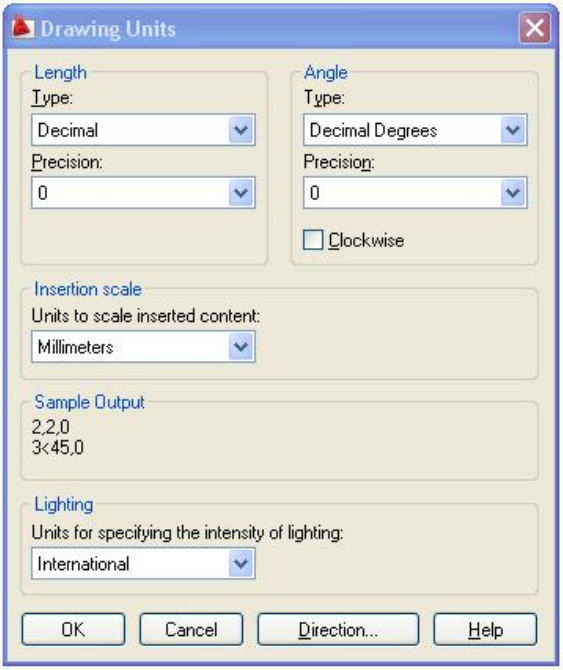


Рисунок 3.1 - Вікно налаштувань розмірності креслення

Розмірні лінії. Пункт меню Format - Dimension Style (стиль розміру). Створюємо новий стиль з назвою "ЄСКД", натиснувши на кнопку "New" і вибравши як основу стиль ISO - 25.

Налаштування вікон ліній (Lines), символів і стрілок (Symbols and Arrows), розмірних написів (Text) показані на рис. 3.2 - 3.4 відповідно. Значення одиниць виставлення розміру Primary Units встановити Decimal, точність 0.

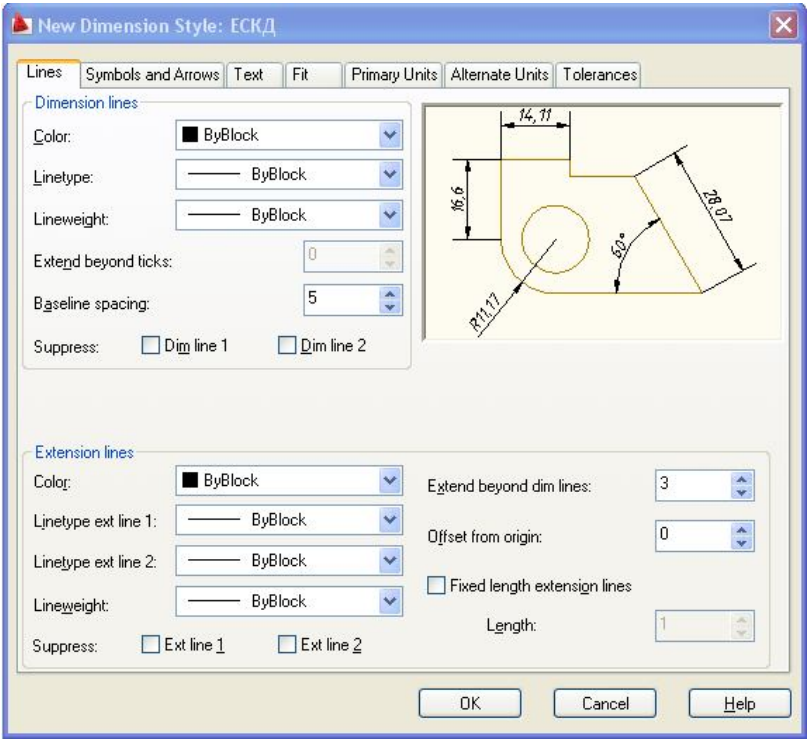


Рисунок 3.2 - Налаштування стилю розмірних ліній

Межі креслення. Межі креслення задаються для двох цілей:

1) встановити межі показу сітки креслення;

2) встановити межі показу креслення командою ZOOM ALL.

Як правило, межі креслення задають рівними розмірам листа креслення:

A1 - 594x840;

A2 - 420x594;

A3 - 297x420;

A4 - 210x297.

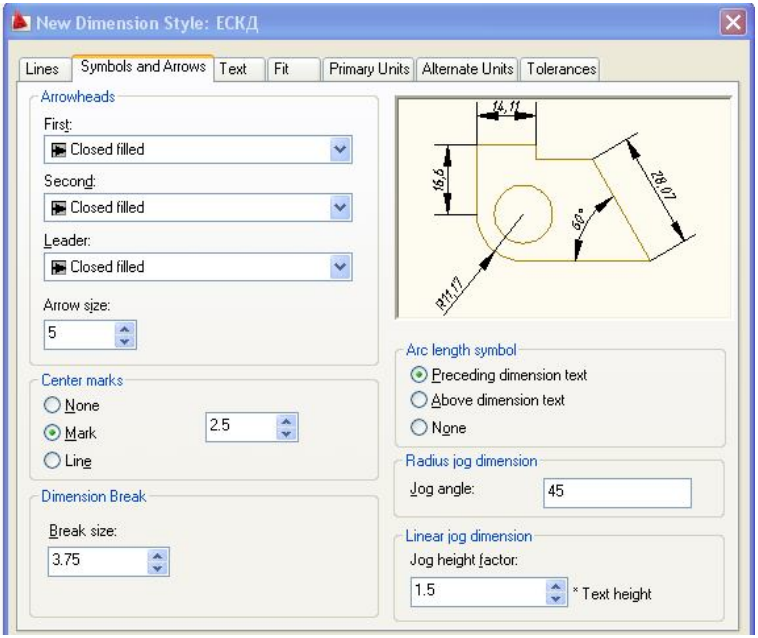


Рисунок 3.3 - Налаштування стилю символів і стрілок

Для завдання меж креслення треба з меню Format вибрати пункт Drawing Limits і в командному рядку задати координати нижнього лівого і верхнього правого кутів прямокутника, що обмежує креслення. Наприклад, для формату А1 буде такий запис: 0,0 [пропуск] 594,840 [пропуск].

Створення шарів і шаблон документу. При підготовці креслень усі використовувані типи ліній мають бути розміщені в окремих шарах. Для створення шарів вибрати з меню Format пункт Layers або натиснувши на відповідну піктограму на панелі інструментів.

У таблиці. 3.1 показані властивості створюваних шарів.

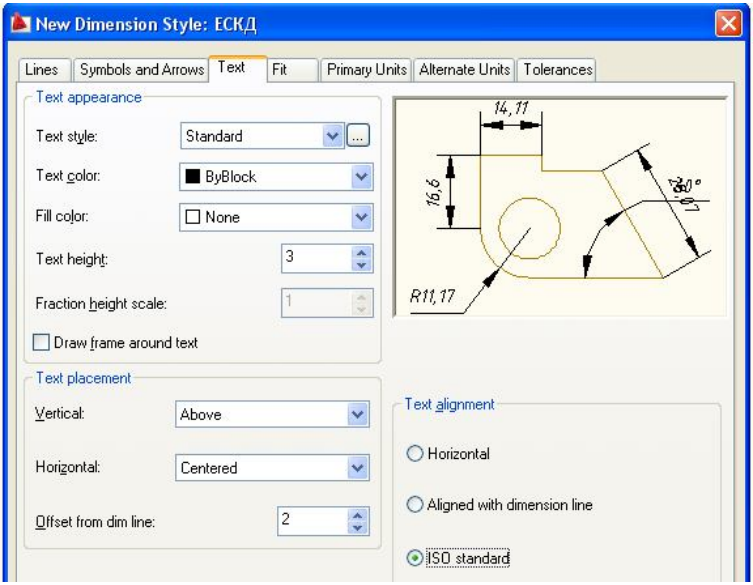
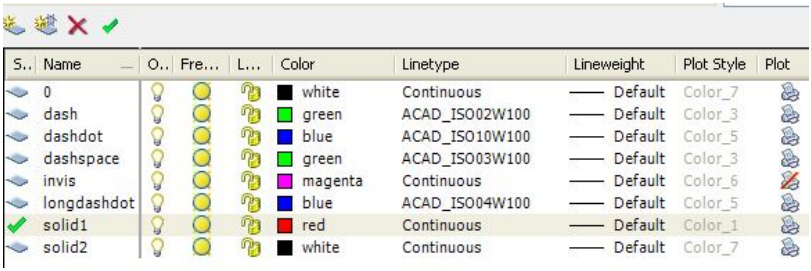


Рисунок 3.4 - Налаштування стилю розмірних написів

Таблиця 3.1 - Властивості шарів креслення

Ім'я шару	Призначення лінії	Колір шару	Тип лінії	Друк
solid1	основна	червоний	Continuous	Так
solid2	допоміжна	чорний	Continuous	Так
dashdot	штрих-пунктир	синій	ACAD_ISO10W100	Так
longdashdot	довгий штрих-пунктир	синій	ACAD_ISO04W100	Так
dash	пунктир	зелений	ACAD_ISO02W100	Так
dashspace	довгий пунктир	зелений	ACAD_ISO03W100	Так
invisible	невидима	малиновий	Continuous	Ні

Кольори шарів слід вибирати із стандартної 9-кольорової палітри швидкого доступу AutoCad. Приклад вікна налаштувань шарів показаний на рис. 3.5. Тоді, при налаштуванні стилю друку, буде простіше для відповідних кольорів шару призначати товщину ліній ніж шукати потрібний колір серед 255 можливих.



S..	Name	O..	Fre...	L...	Color	Linetype	Lineweight	Plot Style	Plot
	0				white	Continuous	Default	Color_7	
	dash				green	ACAD_ISO02W100	Default	Color_3	
	dashdot				blue	ACAD_ISO10W100	Default	Color_5	
	dashspace				green	ACAD_ISO03W100	Default	Color_3	
	invis				magenta	Continuous	Default	Color_6	
	longdashdot				blue	ACAD_ISO04W100	Default	Color_5	
✓	solid1				red	Continuous	Default	Color_1	
	solid2				white	Continuous	Default	Color_7	

Рисунок 3.5 - Шари креслення

Товщина ліній Lineweight має бути заданою за умовчанням, тобто нульовий. Товщина ліній задаватиметься залежно від кольору шару при налаштуваннях друку документу, що буде розглянуто нижче.

Щоб не втратити виконані налаштування, треба зберегти порожній файл AutoCad у вигляді шаблону Template, що має розширення DWT.

Гарячі клавіші. Для швидкого застосування певних функцій AutoCad корисно пам'ятати призначення "гарячих" клавіш, що допомагають не лише прискорити роботу над кресленням, але і забезпечити безперервність операції креслення. Наприклад, включення/виключення режимів прив'язки, ортогонального режиму безпосередньо в процесі побудови лінії. Список клавіш швидкого доступу приведений нижче:

F1 - довідка;

F2 - історія командного рядка;

F3 - налаштування прив'язок (аналогічно Shift + ПКМ, Osnap Settings.);

F7 - включити/вимкнути сітку на робочому полі креслення в межах лімітів;

F8 - включити/вимкнути режим прямокутного креслення;

F9 - включити/вимкнути режим прив'язки до сітки;

F10 - включити/вимкнути режим полярних координат.

Варто також пам'ятати, що натиснення клавіші Space (пропуск) еквівалентно натисненню клавіші Enter.

Друк документу. Перед друком документу необхідно виконати налаштування кольору і товщини ліній креслення, які визначаються створеними шарами. Для цього необхідно створити новий стиль друку.

При виборі пункту меню File - "Plot…" (викреслювання) у вікні, що відкрилося, в секції Plot style table (таблиця стилів), вибрати пункт "New". У майстрові, що з'явився, встановити перемикач в положення "Start from scratch" і натиснути "Next". Вводимо ім'я нового стилю друку і, не міняючи нічого в наступному вікні, натискаємо "Finish". У полі Plot style table буде вибраний створений стиль і, крім того, активується піктограма налаштувань стилю. При натисненні на піктограму відкривається вікно налаштування стилів.

На вкладці Form View для кожного кольору створених шарів встановлюємо колір друку Color і товщину лінії Lineweight. Слід звернути увагу, що у вікні Form View, в перших строчках, знаходиться палітра кольорів швидкого доступу, що цілком виправдовує її застосування для вибору кольору шарів.

3.2 Програмування в AutoCad на мові AutoLisp: основні функції і методи

Розглянемо мову Лісп, створену американським вченим Джоном Маккарті в 1957 р. в Масачусетському технологічному інституті, і що знайшов широке застосування. Назва мови означає **LIS**t **P**rocessing. Інтерпретатор мови Лісп вбудований в САПР AutoCad і носить назву AutoLisp (АвтоЛісп).

Лісп відрізняється високою компактністю - ядро інтерпретатора займає 4.10 Кб, функціональним стилем програмування і використанням зворотної польської нотації.

Основне поняття мови Лісп - список. Список - перелік атомів або списків, відокремлених один від одного пропусками і записаних у дужках. Списки можуть бути вкладеними. Атом в Ліспі - це простий (на відміну від списку) тип даних: число, символьний рядок, функція. У Ліспі немає відмінності між текстом програми і оброблюваними нею даними.

У інших мовах (наприклад, в C++) програма і дані жорстко розділені. У Ліспі ж і дані, і текст програми є списками. Список, що є програмою, включає атоми-функції, які і викликаються при виконанні програми. Те, що з програмою можна працювати, як з даними, і визначає можливість динамічної модифікації тексту програми, що є властивістю мов штучного інтелекту. Відмінність списку-програми від списку даних полягає в тому, що в Ліспі за умовчанням будь-який список є програмою, і інтерпретатор намагатиметься її виконати. Якщо список - не програма, а дані, потрібно явно відключити його інтерпретацію. З того, що текст програми - теж список, витікає необхідність використання спеціальної системи його запису - зворотній польській нотації

(названа так унаслідок винаходу її польським математиком Яном Лукасевичем). Виклик будь-якої функції в цій нотації записується як список наступного вигляду:

$$\left({}^{з}\ddot{i}\ '\ddot{y}\ \_\acute{o}\acute{o}\acute{i}\hat{e}\ddot{o}^{3}\text{¿} \quad a_1\, a_2 \ldots a_n \right),$$

де $a_1\, a_2 \ldots a_n$ - аргументи функції.

Наприклад, якщо функція складання двох чисел має ім'я "+", то операція 2+3 запишеться як (+ 2 3). Як аргументи можуть фігурувати інші функції, що дозволяє записувати скільки завгодно складні формули в зворотній польській нотації.

Нехай дана функція $f(x,y) = \left(\frac{2x+3}{y-1} \right) \cdot (y - x)$. В зворотному польському записі вона придбає наступного вигляду:

(* (/ (+ (* (* (/ (+ (* 2 x) 3) (- y 1)) (- y x)).

Аналогічна нотація використовувалася раніше в програмованих калькуляторах із стеком типу МК52, МК61, ТІ59 і інших.

Розглянемо конкретну реалізацію мови Лісп - вбудований в САПР AutoCad інтерпретатор мови AutoLisp (АвтоЛісп) в середовищі Visual Lisp [3]. Вибір цієї рідкісної мови як вбудованої для такої популярної САПР програми викликаний тим, що список команд, яким є код на Лісп - оптимальний спосіб представлення графічної інформації, а також легкістю реалізації і невеликими розмірами інтерпретатора. Проте слід зазначити відносно велику трудомісткість розробки програм на АвтоЛіспі. Тим часом, програми на АвтоЛіспі відрізняються надзвичайно високою надійністю; за багато років роботи помилок в роботі інтерпретатора не виявлено.

Команди на Автоліспі записуються у вигляді рядків, поміщених в круглі дужки і починаються з ключового слова "command". АвтоЛісп не випадково має приставку "Авто" (від Автокад) - усі команди Автокад сумісні з командами АвтоЛісп. Якщо виникає питання, як правильно, наприклад, написати рядок для побудови лінії, треба у вікні Автокад намалювати лінію, а потім в історії командного рядка подивитися, яка послідовність дій виконувалася - те ж саме можна використовувати в коді АвтоЛісп. І навпаки, команди АвтоЛісп, набрані в командному рядку Автокад, будуть коректно виконані.

Створення шарів. Для того, щоб зрозуміти правила синтаксису мови АвтоЛісп, розглянемо програму створення шарів. Перед написанням коду зробимо декілька зауважень:

- крапка з комою служить для позначення коментаря;

- натисненням клавіші Enter розділяти рядки один від одного не можна - Enter, так само як і Пропуск, є службовими клавішами і служать для введення команд. Отже в коді програми не повинно бути зайвих символів пропуску і Enter.

- підпрограма оголошується командою *defun*;

- почало і кінець підпрограми полягають в круглі дужки.

Приведений код набирається в редакторові AutoLisp Editor, який викликається з меню Tools AutoCad. Після збереження створюється файл з розширенням "LSP". Для завантаження коду на виконання необхідно з меню Tools вибрати пункт Load Application і відкрити збережений файл, натиснувши на кнопку Load.

При поверненні в простір AutoCad можна переконатися в тому, що було створено сім нових шарів. Підпрограму можна використовувати не лише при створенні програм в AutoLisp, але і при створенні

звичайних креслень. Порядок записаних команд повністю аналогічний тим діям, які ми б виконували, працюючи в меню AutoCad за допомогою миші та клавіатури.

```
(defun Format()
;червоний колір (індекс 1), шар основної лінії з ім'ям шару solid1
(command "_LAYER" "NEW" "solid 1" "color" "1" "solid 1")
;чорний колір (індекс 7), шар допоміжної лінії з ім'ям solid2
;команда "_LAYER" другий раз не записується
(command "NEW" "solid 2" "color" "7" "solid 2")
;синій колір (індекс 5), шар штрих-пунктирной лінії dashdot
;командою "Ltype" завантажується штрих-пунктирная лінія, що має
умовне по;значення "ACAD_ISO10W100" в базі AutoCad.
(command "NEW" "dashdot" "color" "5" "dashdot" "Ltype"
"ACAD_ISO10W100" "dashdot")
;синій колір (індекс 5), шар довгої штрих-пунктирной лінії longdashdot
(command "NEW" "longdashdot" "color" "5" "longdashdot" "Ltype"
"ACAD_ISO04W100" "longdashdot")
;зелений колір (індекс 3), шар короткої пунктирної лінії dash
(command "NEW" "dash" "color" "3" "dash" "Ltype" "ACAD_ISO02W100"
"dash")
;зелений колір (індекс 3), шар довгої пунктирної лінії longdash
(command "NEW" "longdash" "color" "3" "longdash" "Ltype"
"ACAD_ISO03W100" "longdash")
;малиновий колір (індекс 6), шар невидимої суцільної лінії invisible
;команда "plot" дозволяє/забороняє доступ до принтера
(command "NEW" "invisible" "color" "6" "invisible" "plot" "No" "invisible")
;вибір командою "make" шару основної лінії як поточна
;два символи подвійних лапок, наступних один за одним без Пропуску
;виконують програмне натиснення клавіші Enter
(command "make" "solid 1" "")
;завершуюча дужка подрограммы Format
)
;виклик підпрограми створення шарів на виконання
(Format)
```

Побудова лінії. Вводимо позначення:

x1, y1 - координати початку лінії;

x2, y2 - координати кінця лінії.

При програмуванні в редакторові AutoLisp слід замінювати введені вище позначення координат цифрами. Розглянемо побудову лінії чотирма способами.

а) побудова лінії за точним завданням координат

```
(command "_LINE" "x 1, y1" "x 2, y2" "")
```

б) побудова ламаної лінії з трьох точок (лінія з продовженням)

```
(command "_LINE" "x 1, y1" "x 2, y2" "x 3, y3" "")
```

в) побудова лінії по змінних

Координати початку лінії привласнюються змінній p1, координати кінця лінії - p2. Для зведення двох координат в один запис використовується команда *list* (список). Команда привласнення в AutoLisp завжди починається з ключового слова *setq*. Операція привласнення полягає в круглі дужки:

```
;створюємо пари точок початку і кінця лінії
(setq p1(list x1 y1) p2(list x1 y1))
;будуємо лінію
(command "_LINE" p1 p2 "")
```

г) побудова лінії по координатах, що завантажуються з файлу

Вводяться нові команди:

Відкриття файлу - *open*;

Читання рядка з файлу - *read_line*;

Перетворення рядка в дійсне число - *atof*;

Закриття файлу - *close*.

Після відкриття файлу командою *read_line* прочитуються рядки, в яких повинні зберігатися координати точок. Завантаження виконува-

тиметься послідовно рядок за рядком стільки раз, скільки разів була написана команда *read_line*.

Вміст текстового файлу *data.txt*, що містить координати двох точок, повинен виглядати таким чином (пам'ятаємо, що замість позначень x1, y1 і так далі мають бути цифри):

```
x1
y1
x2
y3
```

Код програми на AutoLisp побудови лінії по координатах двох точок, завантажених з файлу:

```
(setq fo(open"E:\\data.txt" "r"));відкриваємо текстовий файл для читання
;прочитуємо значення координати x1 в змінну px1
(setq px1(atof(read_line fo)))
;прочитуємо значення координати y1 в змінну py1
(setq py1(atof(read_line fo)))
;прочитуємо значення координати x2 в змінну px2
(setq px2(atof(read_line fo)))
;прочитуємо значення координати y2 в змінну py2
(setq py2(atof(read_line fo)))
;закриваємо файл
(close fo)
;створюємо точкові пари
;заносимо координати початку лінії в змінну p1
;заносимо координати кінця  лінії в змінну p2
(setq p1(list px1 py1) p2(list px2 py2))
;будуємо лінію
(command "_LINE" p1 p2 "")
```

Побудова кола. Задаємо координати центру x1, y1 і радіус R. Звернути увагу: подвійні лапки у кінці команди побудови кола, як це було в команді побудови лінії, не ставляться:

```
(command "_CIRCLE" "x 1, y1" "R")
```

Побудова дуги. Виконуємо побудову дуги по координатах центру x1, y1 (індекс "c"), координатам початкової точки дуги x2, y2 і координатам кінцевої точки x3, y3

((command "_ARC" "c" "x 1, y1" "x 2, y2" "x 3, y3")

Побудова фаски. Побудова фаски виконується в три етапи (рис. 3.6):

а) задається висота фаски командою Distance (m і n - цифри);

б) виконується команда побудови фаски командою "_CHAMFER"; так само, як і при звичайному кресленні, ця команда виконується двічі - для верхньої і нижньої лінії (x1...x3, y1…y3 - цифри);

в) після побудови фаски будується лінія її межі.

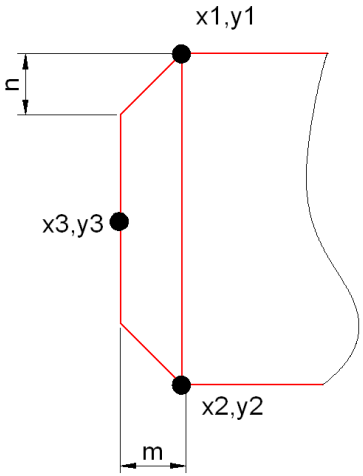


Рисунок 3.6 - До побудови фаски

```
;задаємо висоту фаски
(command "_CHAMFER" "Distance" "m" "n")
;верхня фаска
(command "_CHAMFER" "x 1, y1" "x 3, y3")
;нижня фаска
(command "_CHAMFER" "x 2, y2" "x 3, y3")
;будуємо лінію
(command "_LINE" "x 1, y1" "x 2, y2" "")
```

Побудова округлення. Побудова округлення виконується в п'ять етапів (рис. 3.7) :

а) задається радіус округлення командою Radius;

б) виконується команда побудови округлення командою "_FILLET" для верхньої лінії (при цьому вертикальна лінія стирається);

в) відновлюється вертикальна лінія;

г) виконується команда побудови округлення для нижньої лінії;

д) відновлюється вертикальна лінія.

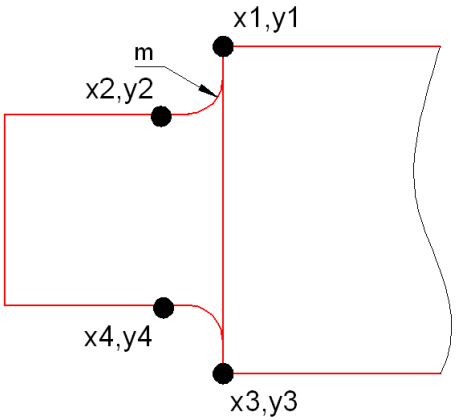


Рисунок 3.7 - До побудови округлення.

```
;задаємо радіус округлення величиною m
(command "_FILLET" "Radius" "m")
;виконуємо округлення верхньої лінії
(command "_FILLET" "x 1, y1" "x 2, y2")
;відновлюємо вертикальну лінію
(command "_LINE" "x 1, y1" "x 3, y3")
;виконуємо округлення нижньої лінії
(command "_FILLET" "x 3, y3" "x 4, y4")
;відновлюємо вертикальну лінію
(command "_LINE" "x 1, y1" "x 3, y3")
```

Виконання штрихування. Штрихування виконується командою "_HATCH". У параметрах задають кут штрихування "a" в градусах, густина "b" і координати точки x1, y1 внутрішньої області:

```
;побудова штрихування
(command "_BHATCH" "P" "USER" "a" "b" "N" "x 1, y1" "")
```

Виділення об'єктів. Виділення можна виконати двома способами: перший полягає у виділенні тільки тих об'єктів, які повністю потрапляють всередину сітки; другий спосіб дозволяє створити сітку, яка виділяє усі об'єкти, які перетинає. У команді створення сітки вказують координати лівого нижнього і правого верхнього кутів (так само, як і при побудові прямокутника).

а) виділення внутрішніх об'єктів сітки

```
(command "_WINDOW" "x 1, y1" "x 2, y2" "")
```

б) виділення об'єктів, що перетинаються сіткою

```
(command "_CROSSING" "x 1, y1" "x 2, y2" "")
```

Видалення об'єктів (стирання). Віддаляються об'єкти, виділені сіткою:

```
(command "_ERASE" "_CROSSING" "x 1, y1" "x 2, y2" "")
```

Усікання. Команда працює таким чином: сіткою виділяється область, в яку потрапляють об'єкти, що модифікуються. Далі вказуються координати точок на тих об'єктах, які мають бути видалені (рис. 3.8):

```
;усікання двох об'єктів по координатах x3, y3 і x4, y4
(command "_TRIM" "_CROSSING" "x 1, y1" "x 2, y2" "" "x 3, y3" "x 4, y4"
"")
```

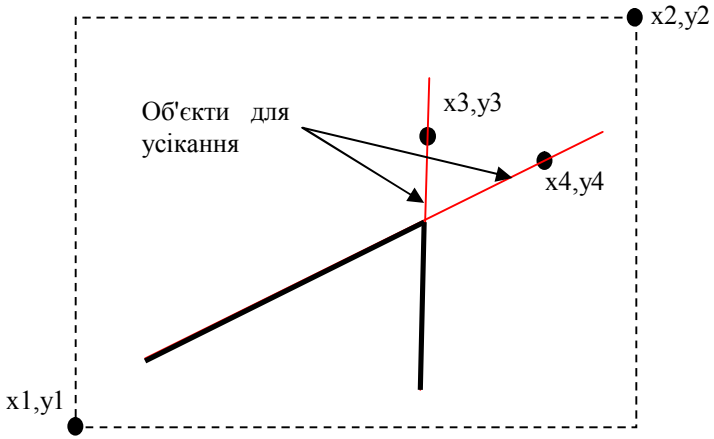


Рисунок 3.8 - Усікання об'єктів креслення

Звернути увагу: після координат x2, y2 команди "_CROSSING" записується пара подвійних лапок, що означає кінець виконання операції побудови сітки. Тільки після цього виробляється вказівка точок відсікання. Повністю ідентичний порядок дій буде при звичайному, неавтоматизованому, виконанні команди усікання TRIM з панелі інструментів AutoCad.

Створення блоку. При створенні блоку командою "_BLOCK" вказується ім'я створюваного блоку Name, координати базової точки блоку x1, y1, які потім в команді вставки блоку використовуватися для здійснення прив'язки, а також сітка "_WINDOW" з координатами x2, y2 і x3, y3, що повністю охоплює елементи, що включаються в блок:

```
((command "_BLOCK" "Name" "x 1, y1" "_WINDOW" "x 2, y2" "x 3, y3" "")
```

Вставка створеного раніше блоку здійснюється командою "_INSERT", з вказівкою імені блоку і координат точки вставки x1, y1. У цій координаті розміститься точка тієї частини блоку, яка була вказана як базова при його створенні:

```
(command "_INSERT" "Name" "x 1, y1" "" "" "")
```

Робота з масивами. Розглянемо використання полярного масиву, який, наприклад, буде потрібний для побудови листів статора і ротора.

Команда побудови масиву "_ARRAY" сіткою "_WINDOW" з координатами x1, y1 і x2, y2 покриває область, в якій знаходяться об'єкти, що поміщаються в масив. Команда "_POLAR" дозволяє створити полярний масив виділених елементів навколо кола з центром x3, y3, кількістю "Number", рівномірно розподілених в секторі кола, що має внутрішній кут "A" градусів і "Y" (або немає - "N"), що повертаються при копіюванні.

```
((command "_ARRAY" "_WINDOW" "x 1, y1" "x 2, y2" "" "_POLAR" "x 3, y3"
"Number" "A" "Y")
```

Виставлення розмірів:

а) лінійний розмір з виміром. Виставлення розміру виконується командою "_DIMLINEAR" з вказівкою координат початку, кінця лінії і довільні координати точки в необхідному місці розміщення напису (рис. 3.9) :

(command "_DIMLINEAR" "x 1, y1" "x 2, y2" "x 3, y3")

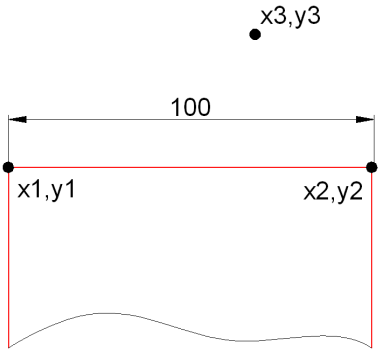


Рисунок 3.9 - Виставлення лінійного розміру

б) із зміною напису. Синтаксис команди схожий на виставлення лінійних розмірів, але відрізняється додаванням напису командою "TEXT" і наступним розміщенням самого напису в лапках:

(command "_DIMLINEAR" "x 1, y1" "x 2, y2" "TEXT" "%%c60" "x 3, y3")

При написанні тексту напису для введення спеціальних креслярських символів використовуються наступні позначення:

%%c - знак діаметру ∅;

%%d - знак градуса о;

%%p - знак ±;

%%% - знак відсотка %.

в)діаметральний розмір. Для виставлення діаметрального розміру після команди "_DIMDIAMETR" вказуємо координати протилежних квадрантів кола:

(command "_DIMDIAMETR" "x 1, y1" "x 2, y2")

г) радіус. Для вказівки радіусу кола або дуги після команди "_DIMRADIUS" вказуємо координати протилежних квадрант кола або крайніх точок дуги :

(command "_DIMRADIUS" "x 1, y1" "x 2, y2")

Виведення написів. Для виведення написів в полі креслення використовується команда "_TEXT" з вказівкою координати розміщення першого символу напису :

(command "_TEXT" "x 1, y1" "" "" "текст напису")

Службові команди AutoCad. Службові команди AutoCad застосовуються для організації відображення креслення. Розглянемо найбільш поширені.

Завдання меж креслення (області показу сітки) з координатами x1, y1 лівого нижнього і x2, y2 правого верхнього кутів:

(command "_LIMITS" "" "x 1, y1" "x 2, y2" "")

Збільшення до розмірів меж креслення:

(command "_ZOOM" "_ALL")

Збільшення усіх об'єктів креслення до розмірів робочого простору:

(command "_ZOOM" "_E")

3.3 Параметричне проектування в AutoCad на мові AutoLisp

У проектній програмі, написаній на мові С++ розраховуються параметри асинхронного двигуна з короткозамкненим ротором. Можна поставити завдання створення програму на мові AutoLisp для побудови вузла електродвигуна.

Розміри цього вузла, наприклад валу, залежатимуть від потужності машини, але порядок побудови і набір ліній креслення буде однаковим.

Якщо сформувати код побудови ліній креслення не по жорстко заданих координатах, а по змінних, яким привласнюватимуться значення розрахункових довжин, то завдання автоматичної побудови креслення буде вирішено. Саме таке креслення, пов'язане з автоматичною побудовою однотипних деталей, називається параметричним.

3.3.1 Основні математичні операції в AutoLisp. При розгляді 2D-проекций деталі видно, що вони можуть бути розбиті на елементарні графічні примітиви (рис. 3.10). Кожен примітив однозначно визначається координатами своїх базових точок: початковою і кінцевою точок відрізку; початковою, кінцевою точок і центру дуги. При формуванні коду побудови параметричної деталі одна точка креслення вибирається як базова, а інші обчислюються як зміщення відносно базової.

Для параметричної деталі, показаної на рис. 3.10 базовою буде точка 0, а координати інших точок обчислюються. Так, координати точки 1 будуть рівні: x1 = x0 – L12/2, y1 = y0 + L13/2. Як видно, у виразу використано три арифметичні операції: складання, віднімання і ділення.

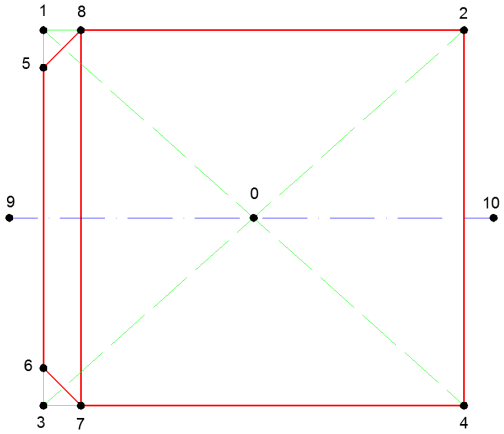


Рисунок 3.10 - Деталь, підготовлена до параметричного креслення

При записі в AutoLisp (табл. 3.2) цей вираз буде записаний в зворотній нотації:

- для x1: (- x0 (/ L12 2))
- для y1: (+ y0 (/ L13 2))

У мові AutoLisp також передбачена вбудована змінна PI, зі значенням, рівним числу π.

Для привласнення значень змінним використовується функція SETQ. Функція SETQ (**SET** by **Q**uote) міняє значення змінної, а не саму змінну. Аргументами функції є перелік пар "змінна" - "значення". Функція повертає результат останнього привласнення.

Наприклад, запис *(SETQ a 10)* поміщає число 10 в область пам'яті, на яку вказує змінна a і одночасно задає тип змінної *a*, - ціле число.

Таблиця 3.2 - Основні математичні операції AutoLisp

№ п/п	Ім'я функції	Математичний запис	Нотація AutoLisp
1	+	x + y	+ x y
2	-	x - y	- x y
3	/	x / y	/ x y
4	*	x * y	* x y
5	модуль	|x|	ABS(x)
6	квадратний корінь	$\sqrt{x}$	SQRT(x)
7	зведення до ступеня	x^y	EXPT(x y)
8	експонента	e^x	EXP(x)
9	натуральний логарифм	ln x	LOG(x)
10	синус	sin(x)	SIN(x)
11	косинус	cos(x)	COS(x)
12	арктангенс	arc tan(x)	ATAN(x)

Примітка: кути в тригонометричних функціях розраховуються в радіанах.

Можна в одній функції присвоїти декілька змінних:

```
(( SETQ a 10 b "213" )
```

Порядок виконання декількох привласнень у функції SETQ визначений зліва направо. Якщо ми напишемо:

```
( SETQ a 10 b ( + a  10 ))
```

то значенням змінної b стане 20.

3.3.2 Побудова параметричної деталі. Розглянемо приклад програми на мові AutoLisp для параметричної побудови деталі, зображеної на рис. 3.10. Координати точок записуватимемо у вигляді імені осі і порядкового номера точки, наприклад: x1, y1. Вважаємо, що довжини відрізків деталі L12, L13 відомі.

```
;Встановлюємо довільні координати центру креслення
(SETQ x0 200 y0 150)
;Встановлюємо значення розмірів L12 і L13
(SETQ L12 200 L13 100)
;знаходимо координати y9, y10 рівні y0
(SETQ y9 y0 y10 y0)
;знаходимо координату x1 = x0 - L12/2
(SETQ x1 (- x0 (/ L12 2)))
;це будуть також координати x5, x6, x3
(SETQ x5 x1 x6 x1 x3 x1)
;знаходимо координату y1 = y0 - L13/2
(SETQ y1 (- y0 (/ L13 2)))
;це будуть також координати y8, y2
(SETQ y8 y1 y2 y1)
;знаходимо координату x2 = x1 + L12
(SETQ x2 (+ x1 L12))
;це буде також координата x4
(SETQ x4 x2)
;знаходимо координату y3 = y1 + L13
(SETQ y3 (+ y1 L13))
;це будуть також координати y7, y4
(SETQ y7 y3 y4 y3)
;знаходимо координату x9 = x1 - 5
(SETQ x9 (- x1 5))
;знаходимо координату x10 = x2 + 5
(SETQ x10 (+ x2 5))
;задаємо висоту фаски F рівну 10
(SETQ F 10)
;знаходимо координату y5 = y1 + F
(SETQ y5 (+ y1 F))
;знаходимо координату y6 = y3 - F
(SETQ y6 (- y3 F))
;знаходимо координату x7
(SETQ x7 (+ x3 F))
```

```
;це буде також координата x8
(SETQ x8 x7)
;створюємо точки з парами координат
(SETQ p0 (LIST x0 y0)
p1 (LIST x1 y1)
p2 (LIST x2 y2)
p3 (LIST x3 y3)
p4 (LIST x4 y4)
p5 (LIST x5 y5)
p6 (LIST x6 y6)
p7 (LIST x7 y7)
p8 (LIST x8 y8)
p9 (LIST x9 y9)
p10 (LIST x10 y10)
)
;вибираємо шар основної лінії (вважаємо, що шари вже були створені)
(command "_LAYER" "Make" "solid 1" "")
;виконуємо побудову горизонтальних ліній
(command "_LINE" p8 p2 "")
(command "_LINE" p7 p4 "")
;виконуємо побудову вертикальних ліній
(command "_LINE" p5 p6 "")
(command "_LINE" p8 p7 "")
(command "_LINE" p2 p4 "")
;виконуємо побудову похилих ліній
(command "_LINE" p6 p7 "")
(command "_LINE" p5 p8 "")
;вибираємо шар осьової лінії
(command "_LAYER" "Make" "longdashdot" "")
;виконуємо побудову осьової лінії
(command "_LINE" p9 p10 "")
;команда "Показати усе"
(command "_ZOOM" "_E")
```

3.3.3 Приклад повної програми параметричного креслення. Розглянемо автоматизоване формування коду AutoLisp програми побудови параметричної деталі, розглянутої в попередньому розділі.

Нехай в деякій програмі проектування, написаній на мові C++, були набуті значень змінних типу float : L12, L13, F. Також нехай за-

дані координати центру креслення типу *int*: x0, y0. Складемо на мові C++ програму формування коду AutoLisp, приведеного в п.3.3.2, для побудови деталі, зображеної на рис. 3.10.

У файлі *program.cpp* створюємо нову функцію, в завдання якої входитиме формування рядків на мові AutoLisp для побудови лінії.

```
//відкриття файлу Drawing.lsp для виведення коду AutoLisp
ofstream df("Drawing.lsp");
//Побудова лінії
void line(int* x1, int* y1, int* x2, int* y2)
{
df << "(command \"_LINE\" \"" << *x1 << "," << *y1 << "\" \"" << *x2
<< "," << *y2 << "\" \"\")\n";
}
```

На цьому найскладніша частина програми закінчена.

Тепер можна створити функцію власне порядку викреслювання:

```
//Створення програми в AutoLisp побудови параметричної деталі
void Draw ()
{
void line(int*, int*, int*, int*); //оголошення прототипу функції
int x1, x2, x3, x4, x5, x6, x7, x8, x9, x10;
int y1, y2, y3, y4, y5, y6, y7, y8, y9, y10;
df << ";Побудова параметричної деталі\n";
df << ";\n";
//розрахунок координат точок деталі відносно базових x0, y0
//координати x
x1 = x0 - L12/2;
x2 = x1 + L12;
x3 = x1;
x4 = x2;
x5 = x1;
x6 = x1;
x7 = x1 + F;
x8 = x7;
x9 = x1 - 5;
x10 = x2 + 5;
```

```
//координати y
y1 = y0 - L13/2;
y2 = y1;
y3 = y1 + L13;
y4 = y3;
y5 = y1 + F;
y6 = y3 - F;
y7 = y3;
y8 = y1;
y9 = y0;
y10 = y0;
//вибір шару основної лінії
df << "(command \"_LAYER\" \"Make\" \"solid 1\" \"\")"
//побудова горизонтальних ліній
line(&x8, &y8, &x2, &y2);
line(&x7, &y7, &x4, &y4);
//побудова вертикальних ліній
line(&x5, &y5, &x6, &y6);
line(&x7, &y7, &x8, &y8);
line(&x2, &y2, &x4, &y4);
//побудова осьової лінії
//вибір шару осьової лінії
df << "(command \"_LAYER\" \"Make\" \"longdashdot\" \"\")"
line(&x9, &y9, &x10, &y10);
df << ";Показати усе" << "\n";
df << "(command \"_ZOOM\" \"_E\")";
//закрити файл
df.close();
}
```

У наведеній функції усі змінні передаються по покажчику для прискорення обміну даними. Оператор "&" перед ім'ям змінної служить для отримання адреси змінної в пам'яті.

Як видно з лістингу, програма побудови параметричної деталі на C++ формується значно простіше, ніж безпосередньо в редакторові Visual Lisp. Завдання зводиться до розрахунку координат і виведення рядків побудови ліній по цих координатах у файл.

Контрольні питання

1. Як задати точність одиниць креслення?

2. Призначення і порядок створення шарів. Додавання нових типів ліній з бази даних. Завдання товщини лінії. Заборона виведення шару на друк.

3. Як задати стиль тексту і розмірних ліній? Чи можна задати за умовчанням розмір і нахил тексту?

4. Як настроїти друк документу? Який зв'язок між стилем друку і шарами креслення?

5. Арифметичні операції мови AutoLisp.

6. Розглянути чотири способи параметричної побудови лінії в AutoLisp. Яка сфера застосування кожного з розглянутих способів?

7. Побудова кола і дуги в AutoLisp.

8. Виконання штрихування в AutoLisp.

9. Операції виділення, видалення і усікання в AutoLisp.

10. Створення блоків і робота з масивами в AutoLisp.

11. Побудова фасок і округлень в AutoLisp.

12. Автоматичне виставлення розмірів і створення написів в AutoLisp. Зміна тексту розмірних ліній, що виводяться за умовчанням.

13. Особливості параметричного креслення. Завантаження координат деталі з файлів даних.

14. Створення шарів в AutoLisp. У чому полягає особливість створення усіх шарів, відмінних від першого? Вибір поточного шару.

Практичні завдання

1. Виконати форматування робочої області креслення розміром А3, горизонтальної орієнтації так, як описано в навчальному посібнику. Зберегти відформатований шаблон документу.

2. На мові AutoLisp створити програму параметричного викреслювання квадрата із стороною 200 мм за допомогою ліній. Лінії будувати методом завдання координат в списку.

3. На мові AutoLisp скласти програму створення трьох шарів для основної лінії, допоміжної і осьової. Основною лінією викреслити коло радіусом 100 мм. Накреслити осьові лінії. У шарі допоміжних ліній вивести напис "коло ∅ 200 мм".

4. На мові AutoLisp написати програму викреслювання прямокутника з розмірами 50 х 200 мм. З лівого боку прямокутника побудувати фаску заввишки 5 мм, а справа - 10 мм.

5. Написати програму AutoLisp побудови квадрата із стороною 100 мм. У квадрат вписати коло. Побудувати осьові лінії. Координати центру кола (точці перетину діагоналей квадрата) завантажити з файлу даних.

6. На мові AutoLisp скласти програму побудови рівностороннього трикутника з довжиною сторони 50 мм. Трикутник помістити в блок. Створити масив з трикутників, завантажених із створеного блоку: кількість рядків - 1, кількість стовпців - 5, відстань між трикутниками - 50 мм.

7. Написати програму AutoLisp побудови кола діаметром 100 мм. Побудувати осьові лінії. У першому квадранті виконати штрихування. Проставити розмір кола.

РОЗДІЛ 4

МЕТОДИ І ЗАСОБИ ОПТИМІЗАЦІЇ

У цьому розділі будуть розглянуті тільки ті методи і засоби оптимізації, які безпосередньо реалізовані в наступному розділі, присвяченому складанню повної програми оптимального проектування асинхронного двигуна з короткозамкненим ротором [6].

До них відносяться [7]:

- сплайн-інтерполяція табличних функцій;
- метод покоординатного спуску;
- метод деформованого багатогранника Нелдера-Міду;
- метод зовнішніх штрафних функцій.

Кожен з перерахованих методів є незамінним інструментом будь-який САПР програми.

4.1 Інтерполяція методом сплайнів

Сплайн (англ. spline - рейка, лінійка) - це визначена в деякій області $G \in R^n$ кусочно-поліноміальна функція. Таким чином, існує розбиття області G на підобласті таке, що усередині кожної підобласті сплайн є поліномом деякого ступеня N.

Інтерполяція за допомогою сплайнів називається сплайн-інтерполяцією. Інтерполяція сплайнами третього порядку - це швидкий, ефективний і стійкий спосіб інтерполяції функцій. Сплайн-інтерполяція є однією з альтернатив поліноміальної інтерполяції.

У основі сплайн-інтерполяції лежить наступний принцип. Інтервал інтерполяції розбивається на невеликі відрізки, на кожному з

яких функція задається поліномом третього ступеня. Коефіцієнти полінома підбираються так, щоб виконувалися певні умови (які саме, залежить від способу інтерполяції). Загальні для усіх типів сплайнів третього порядку вимоги - безперервність функції і, зрозуміло, проходження через вказані їй точки. Додатковими вимогами можуть бути лінійність функції між вузлами, безперервність вищих похідних і так далі.

Основними перевагами сплайн-інтерполяції є її стійкість і мала трудомісткість. Системи лінійних рівнянь, які вимагається вирішувати для побудови сплайнів, дуже добре обумовлені, що дозволяє отримувати коефіцієнти поліномів з високою точністю. В результаті, навіть при дуже великих N, обчислювальна схема не втрачає стійкості. Побудова таблиці коефіцієнтів сплайну вимагає O(N) операцій, а обчислення значення сплайну в заданій точці - усього лише O(log(N)).

4.1.1 Лінійний сплайн. Лінійний сплайн - це сплайн, складений з поліномів першого ступеня, тобто з відрізків прямих ліній. Точність інтерполяції лінійними сплайнами невисока, також слід зазначити, що вони не забезпечують безперервності навіть перших похідних. Проте в деяких випадках кусочно-лінійна апроксимація функції може виявитися прийнятніше, ніж апроксимація вищого порядку. Наприклад, лінійний сплайн зберігає монотонність переданого в нього набору точок.

На графіці (рис. 4.1) наведений приклад лінійного сплайну, що інтерполює функцію $f = \cos(0.5 \cdot \pi \cdot x)$ на відрізку [- 1, 1]. З технічної точки зору, лінійний сплайн реалізований, як сплайн третього порядку з коефіцієнтами при другому і третьому ступені x, який дорівнює нулю.

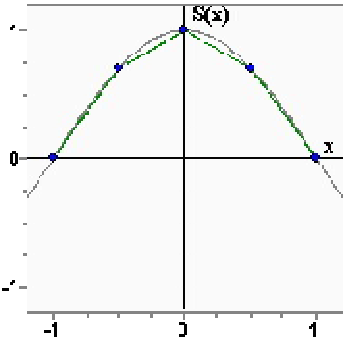


Рисунок 4.1 - Лінійний сплайн

4.1.2 Сплайн Ерміта. Сплайн Ерміта - це сплайн третього порядку, похідна якого набуває у вузлах сплайну заданих значень. У кожному вузлі сплайну Ерміту задано не лише значення функції, але і значення її першої похідної. Сплайн Ерміта має безперервну першу похідну, але друга похідна у нього розривна.

На графіці (рис. 4.2) наведений приклад сплайну Ерміта, що інтерполює функцію f = cos(0.5·π·x) на відрізку [- 1, 1]. Можна побачити, що точність інтерполяції значно краща, ніж у лінійного сплайну.

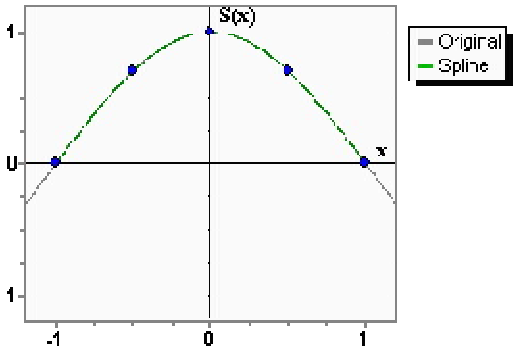


Рисунок 4.2 - Сплайн Ерміта

4.1.3 Кубічний сплайн. Усі сплайни, розглянуті на цій сторінці, є кубічними сплайнами - в тому сенсі, що вони є кусочно-кубічними функціями. Проте коли говорять "кубічний сплайн", то зазвичай мають на увазі конкретний вид кубічного сплайну, який виходить, якщо вимагати безперервності першої і другої похідних. Кубічний сплайн задається значеннями функції у вузлах і значеннями похідних на межі відрізку інтерполяції (або перших, або других похідних).

Якщо відоме точне значення першої похідної на обох межах, то такий сплайн називають фундаментальним. Похибка інтерполяції таким сплайном дорівнює $O(h^4)$.

Якщо значення першої (чи другої) похідної на межі невідоме, то можна задати так звані природні граничні умови S"(A) = 0, S"(B) = 0, і отримати природний сплайн. Погрішність інтерполяції природним сплайном складає $O(h^2)$. Максимум погрішності спостерігається в околицях граничних вузлів, у внутрішніх вузлах точність інтерполяції значно вища.

Ще одним видом граничної умови, яку можна використовувати, якщо невідомі граничні похідні функції, являється умова типу "сплайн, що завершується параболою". В цьому випадку граничний відрізок сплайну представляється поліномом другого ступеня замість третього (для внутрішніх відрізків як і раніше використовуються поліноми третього ступеня). У ряді випадків це забезпечує більш високу точність, чим природні граничні умови.

Нарешті, можна поєднувати різні типи граничних умов на різних межах. Зазвичай має сенс так робити, якщо у нас є тільки частина інформації про поведінку функції на межі (наприклад, похідна на лівій межі - і ніякій інформації про похідну на правій межі).

На графіці (рис. 4.3) наведений приклад кубічного сплайну, що інтерполює функцію f = cos(0.5·π·x) на відрізку [- 1, 1]. Можна бачити, що точність інтерполяції близька до точності Ермітового сплайну.

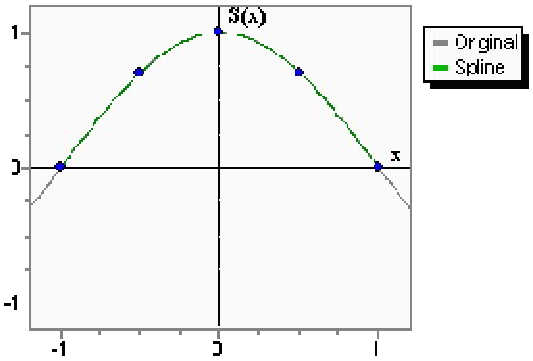


Рисунок 4.3 - Кубічний сплайн

4.1.4 Сплайн Акіми. Сплайн Акіми - це особливий вид сплайну, стійкий до викидів. Недоліком кубічних сплайнів є те, що вони схильні осцилювати в околицях точки, що істотно відрізняється від своїх сусідів. На графіці (рис. 4.4) приведений набір точок, що містить один викид. Зеленим кольором позначений кубічний сплайн з природними граничними умовами. На відрізках інтерполяції, що граничать з викидом, сплайн помітно відхиляється від інтерпольованої функції - позначається вплив викиду. Червоним кольором позначений сплайн Акіми. Можна бачити, що на відміну від кубічного сплайну, сплайн Акіми в меншій мірі схильний до впливу викидів. На відрізках, що граничать з викидом, практично відсутні ознаки осциляції.

Важливою властивістю сплайну Акіми є його локальність - значення функції на відрізку [xi, xi+1] залежать тільки від f_{i-2} , f_{i-1} , f_i , f_{i+1} ,

f_{i+2} , f_{i+3}. Другою властивістю, яку слід брати до уваги, є нелінійність інтерполяції сплайнами Акіми - результат інтерполяції суми двох функцій не дорівнює сумі інтерполяційних схем, побудованих на основі окремих функцій. Для побудови сплайну Акіми вимагається не менше 5 точок. У внутрішній області (тобто між x_2 і x_{N-3} при нумерації точок від 0 до N - 1 погрішність інтерполяції має порядок $O(h^2)$.

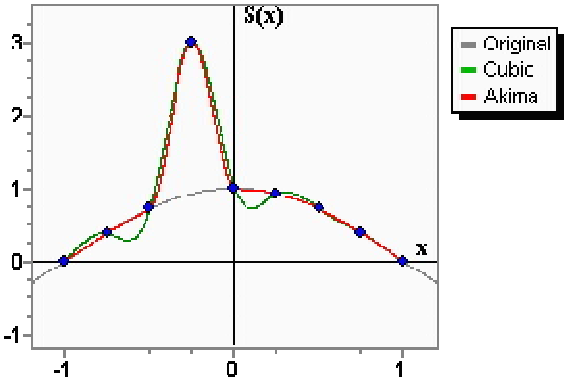


Рисунок 4.4 - Інтерполяція сплайнами кубічним і Акіми

4.1.5 Реалізація методу сплайн-інтерполяції. Поширеною є побудова на кожному відрізку $[x_i, x_{i+1}]$, i=0..n-1 кубічній функції. При цьому сплайн - шматкова функція, на кожному відрізку задана кубічною функцією, є кусково-безперервною, разом зі своєю першою і другою похідною.

Шукатимемо кубічний сплайн на кожному з часткових відрізків $[x_i, x_{i+1}]$ (рис. 4.5).

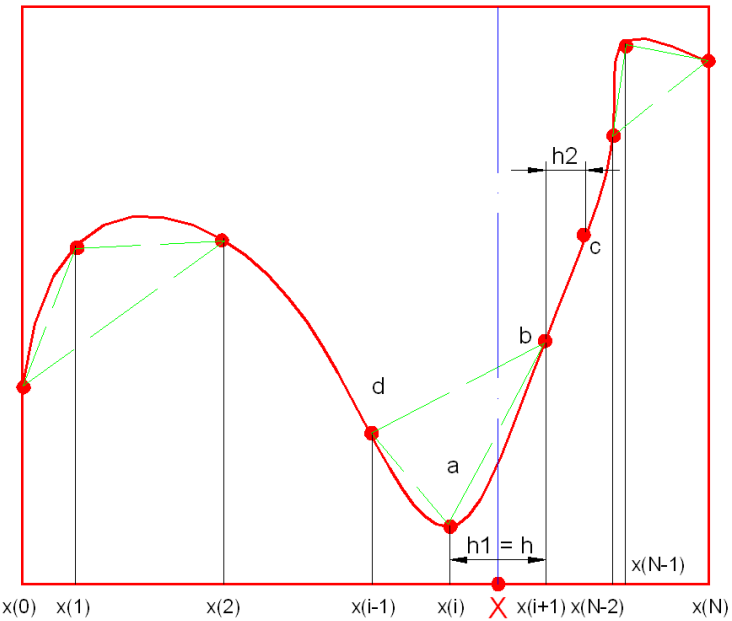


Рисунок 4.5 - Графічне представлення сплайн-інтерполяції

Для спрощення запису розрахункових формул введемо допоміжні змінні:

$a = x_i$, $ya = f(x_i)$;

$b = x_{i+1}$, $yb = f(x_{i+1})$;

$c = x_{i+2}$, $yc = f(x_{i+2})$;

$d = x_{i-1}$, $yd = f(x_{i-1})$;

$h = h_1 = b - a$:

$h_2 = c - b$;

для лівої межі (x_1, x_2, x_3)

$$m1 = \frac{4f(x_1) - f(x_2) - f(x_0)}{2h_1};$$

для правої межі (x_N, x_{N-1}, x_{N-2})

$$m2 = \frac{3f(x_N) + f(x_{N-2}) - 4f(x_{N-1})}{2h_2};$$

для середніх точок

$$m2 = \frac{yc - ya}{2h_2}.$$

Знаходимо складові сплайну:

$$s_1 = \frac{ya(b-x)^2\left[2(x-a)+h\right]}{h^3};$$

$$s_2 = \frac{yb(x-a)^2\left[2(b-x)+h\right]}{h^3}$$

$$s_3 = \frac{m1(b-x)^2(x-a)}{h^2};$$

$$s_4 = \frac{m2(x-a)^2(x-b)}{h^2}.$$

Результуюче значення функції в шуканій точці X дорівнюватиме сумі кускових сплайнів :

$$S = \sum_{i=1}^{4} s_i$$

Оголошення функції сплайн-інтерполяції:

```
extern float spline(float x, float* arg, float* func);
//x - шуканий аргумент, arg - значення аргументів, func - значення функції
```

Виклик функції сплайн-інтерполяції в програмі:

```
Hz1 = spline(Bz1, "St_Bz.dat", "St 2013Z.dat");
```

Програмний код інтерполяції методом сплайнів:

```
float spline(float x, char* f1, char* f2)
{
ifstream fileh(f1); //відкриття файлу аргументу
ifstream filem(f2); //відкриття файлу функції
fileh.seekg(0, ios::beg); //установка покажчика читання на адресу 0
filem.seekg(0, ios::beg);

char buffer[100]; //буфер для зберігання тимчасових даних
int line_count = 0; //лічильник рядків, завантажених з файлу
//оголошення допоміжних змінних
float a, b, c, d, ya, yb, yc, yd;
float m1, m2, S, s1, s2, s3, s4;
float h, h1, h2;
// заповнення масивів з файлу даних
while (fileh.good()) //доки не досягнутий кінець файлу
{
        fileh.getline(buffer, sizeof(buffer));//завантажити рядок аргументів
        arg[line_count] = atof(buffer); //збереження реч. числа в масив
        filem.getline(buffer, sizeof(buffer)); //завантажити рядок функцій
        func[line_count] = atof(buffer); //збереження реч. числа в масив
        //Примітка: ф-я atof() переводить рядок в речовий тип
        line_count++; //збільшення лічильника на 1
}
int N = line_count - 1; //кількість аргументів
for (int i=0; i<=N - 1; i++) //цикл від мін. до макс. числа аргументів
{
        if (x>= arg[i] && x < arg[i+1]) //якщо шукана точка в діапазоні
        {
                //розрахунок допоміжних змінних
                a = arg[i];
                b = arg[i+1];
                c = arg[i+2];
                d = arg[i - 1];
                ya = func[i];
                yb = func[i+1];
                yc = func[i+2];
                yd = func[i - 1];
                h = b - a;
                h1 = h;
                h2 = c - b;
                if (i<=1)
                m1 = (func[1]*4.-func[2]- func[0]*3.)*0.5/h1;
        else if (i>1)
```

```
            m1 = (yb - yd)*0.5/h1;
        if (i >= N - 1)
            m2 = (func[N]*3.+func[N - 2]+func[N - 1]*4.)*0.5/h2;
        else if (i< N - 1)
            m2 = (yc - ya)*0.5/h2;
        //розрахунок кускових сплайнів
        s1 = pow(2.*(x - a)+h)*ya/pow(h, 3);
        s2 = pow(2.*(b - x)+h)*yb/pow(h, 3);
        s3 = pow(x - a)*m1/(h*h);
        s4 = pow(x - b)*m2/(h*h);
        // розрахунок результуючого значення
        S = s1+s2+s3+s4;
        return S; //передача розрахованого значення в програму
        }// кінець перевірки умов if
        else if (x == arg[i+1]) //
        {
            S = func[i + 1];
            return S;
        }
} // кінець циклу
}// кінець функції spline()
```

Примітка: рядки відкриття файлів, виділені в лістингу напівжирним шрифтом треба прибрати, якщо функція сплайн-інтерполяції використовуватиметься для завантаження значень аргументів і функції з масивів. При цьому оголошення функції необхідно перетворити до наступного вигляду (N - кількість елементів в масиві):

```
float spline(float x, char* arg, char* func, int N);
```

Масиви значень аргументів *arg* і функцій *func* передаються у функцію *spline()* по покажчику. Наведена функція реалізації кубічної сплайн інтерполяції може бути застосована до будь-яких табличних залежностей. Оголошення файлів значень аргументу і функції як покажчиків на символьну змінну, робить функцію інваріантною до імен файлів, що містять вхідні дані.

4.2 Метод покоординатного спуску

У САПР вирішуються завдання оптимізації - вибору найкращого варіанту з усіх можливих. В процесі рішення задачі оптимізації зазвичай необхідно знайти оптимальні значення деяких параметрів, що визначають це завдання. При рішенні інженерних завдань їх прийнято називати проектними параметрами, а в економічних завданнях їх називають параметрами плану.

Як проектні параметри можуть бути лінійні розміри, швидкість, маса, температура деталі або виробу і тому подібне.

Кількість n проектних параметрів x_1, x_2,..., x_n характеризує розмірність (і міру складності) завдання оптимізації.

Вибір оптимального рішення або порівняння двох і більше альтернативних рішень проводиться за допомогою деякої залежної величини (функції), яка визначається проектними параметрами. Ця величина називається цільовою функцією (чи критерієм якості).

В процесі рішення задачі оптимізації мають бути знайдені такі значення проектних параметрів, при яких цільова функція має мінімум (чи максимум).

У загальному вигляді цільову функцію можна записати так:

$$u = f(x_1, x_2,..., x_n).$$

Приклади цільової функції - максимальна міцність або мінімальна маса конструкції, максимальна потужність двигуна, мінімальна собівартість продукції, максимальний прибуток і тому подібне.

Цільова функція не завжди може бути представлена у вигляді формули. Іноді вона може набувати тільки деяких дискретних значень, задаватися у вигляді таблиці. Цільових функцій може бути

декілька (наприклад, коли у виробу мають бути мінімальна вага, максимальна міцність і мінімальна собівартість). При цьому деякі з цих функцій можуть виявитися несумісними, тоді вводять пріоритет тієї або іншої цільової функції.

У багатьох випадках цільова функція не описана ніякою функцією. Є лише можливість визначення її значень в довільних точках даної області за допомогою обчислювального алгоритму або шляхом вимірів.

Завдання полягає в наближеному визначенні найменшого значення функції в усій області при відомих її значеннях в певних точках.

Для вирішення подібного завдання в області проектування G, в яких є мінімум цільової функції $u = f(x_1, x_2,..., x_n)$, можна ввести дискретну безліч точок (вузлів) шляхом розбиття інтервалів зміни параметрів $x_1, x_2,..., x_n$ на частини з кроками $h_1, h_2,..., h_n$. У отриманих вузлах можна вичислити значення цільової функції і серед цих значень знайти найменше.

Такий метод загального пошуку з суцільним перебором для вирішення багатовимірних завдань оптимізації не годиться із-за великого об'єму обчислення.

Тут потрібні спеціальні чисельні методи, засновані на цілеспрямованому пошуку. Одним з таких методів є метод покоординатного спуску (рис. 4.6).

Нехай вимагається знайти найменше значення цільової функції $u = f(x_1, x_2,\ldots, x_n)$. Як початкове наближення виберемо в n -мірному просторі деяку точку M_0 з координатами $x1^{(0)}, x2^{(0)},..., xn^{(0)}$. Зафіксуємо усі координати функції u, окрім першої. Тоді $u = f(x1, x2^{(0)},..., xn^{(0)})$ - функція однієї змінної x_1.

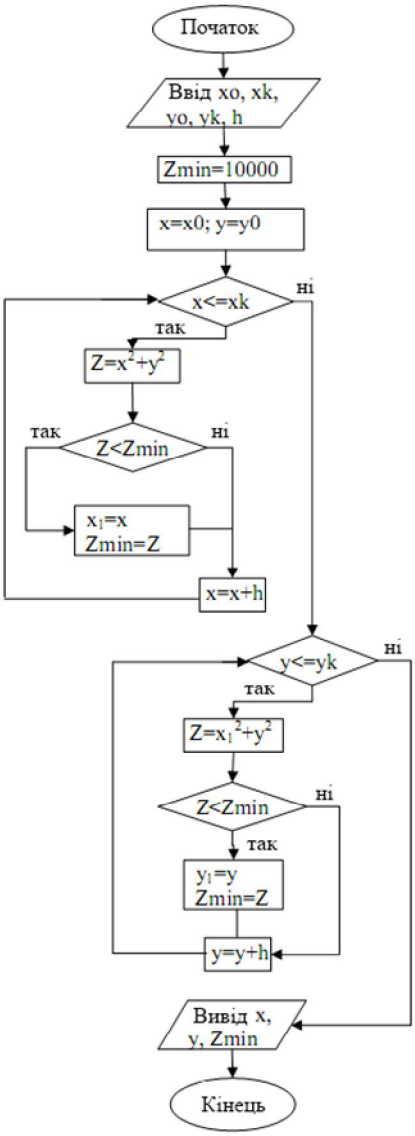


Рисунок 4.6 - Блок-схема алгоритму методу покоординатного спуску

Вирішуючи одновимірну задачу оптимізації для цієї функції, ми від точки M_0 переходимо до точки $M_1(x_1^{(1)}, x_2^{(0)},\ldots, x_n^{(0)})$, в якій функція u набуває найменшого значення по координаті x_1 при фіксованих інших координатах. У цьому полягає перший крок процесу оптимізації, що полягає в спуску по координаті x_1.

Зафіксуємо тепер усі координати, окрім x_2, і розглянемо функцію цієї змінної $u = f(x_1^{(1)}, x_2, x_3^{(0)}\ldots, x_n^{(0)})$. Знову вирішуючи одновимірну задачу оптимізації, знаходимо її найменше значення при $x_2 = x_2^{(1)}$, тобто в точці $M_2(x_1^{(1)}, x_2^{(1)}, x_3^{(0)}\ldots, x_n^{(0)})$.

Аналогічно проводиться спуск по координатах x_3, x_4,..., x_n, а потім процедура знову повторюється від x_1 до x_n і т. д.

Лістинг функції розрахунку оптимального значення заданого параметра асинхронного двигуна приведений нижче. У прикладі варіюються значення висоти і довжини повітряного проміжку. Оптимізація проводиться з використанням методу зовнішніх штрафних функцій, який буде розглянутий далі.

В результаті цього процесу виходить послідовність точок M_0, M_1,..., в яких значення цільової функції складають монотонно убиваючу послідовність $f(M_0) \geq f(M_1) \geq\ldots$ На будь-якому кроці k цей процес можна перервати і набути значення $f(M_k)$ як найменше значення цільової функції в даній області.

Таким чином, метод покоординатного спуску зводить завдання про знаходження найменшого значення функції багатьох змінних до багатократного рішення одновимірних завдань оптимізації по кожному параметру.

Важливо, щоб процес оптимізації сходився, тобто сходилася послідовність значень цільової функції $f(M_0)$, $f(M_1)$,... до найменшого її значення в цій області.

Для функції двох змінних очевидно, що метод не підходить у разі наявності зламів в лініях рівня. Це відповідає "яру" на поверхні. Тут можливий випадок, коли спуск по одній координаті приводить на "дно" яру. Тоді будь-який рух уздовж іншої координати веде до зростання функції, що відповідає підйому на "берег" яру.

Для гладких функцій при вдало вибраному початковому наближенні (у деякій околиці мінімуму) процес сходиться до мінімуму. До переваг методу покоординатного спуску відноситься також можливість використання простих алгоритмів одновимірної оптимізації.

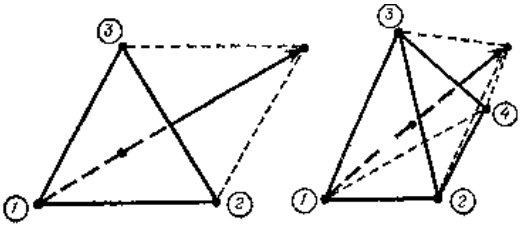
```
void koord()
{
float dld = 0.001;
float ddelta = 0.0001;
float ld_opt = ld;
float delta_opt = delta;
int step_ld, step_delta;
ld_min = lamda_min*tau;
ld_max = lamda_max*tau;
step_ld = int((ld_max - ld_min)/dld);
step_delta = int((delta_max - delta_min)/ddelta);
//перша координата оптимізації
ld = ld_min;
delta = delta_fix;
float fmin = 10000;
number = 0;
for (int i = 0; i <= step_ld;i++)
{
        auto_func(); //функція автоматичного розрахунку
        find_pay(); //розрахунок штрафів
        target(); //розрахунок уільової функції
```

```
        if (F < fmin)
        {
                fmin = F;
                ld_opt = ld;
        }
        ld += dld;
        number++;
}
ld = ld_opt;
//друга координата оптимізації
delta = delta_min;
for (int i = 0; i <= step_delta;i++)
{
        auto_func();
        find_pay();
        target();
        if (F < fmin)
        {
                fmin = F;
                delta_opt = delta;
        }
        delta += ddelta;
        number++;
}
delta = delta_opt;
//остаточний розрахунок
auto_func();
//Виведення даних на друк у файл
DataOutput(OPT_GEOM, "Data_optimum.txt");
}
```

4.3 Метод деформованого багатогранника Нелдера-Міда

Уперше метод деформованого багатогранника був запропонований Нелдером і Мідом. Вони запропонували метод пошуку, що виявився дуже ефективним і легко здійснюваним на ЕОМ. Щоб можна було оцінити стратегію Нелдера і Міда, коротко опишемо симплексний пошук. Регулярні многогранники в E^n є симплексами. Наприклад, як видно з рис 4.7, для випадку двох змінних регулярний симплекс є рівно-

стороннім трикутником (три точки); у разі трьох змінних регулярний симплекс є тетраедром (чотири точки) і так далі. На рисунку ① означає найбільше значення f(x). Стрілка вказує напрям найшвидшого поліпшення.



а) б)

Рисунок 4.7 - Регулярний симплекс для випадку двох (а) і трьох (б) незалежних змінних

При пошуку мінімуму цільової функції f(x) пробні вектори *x* можуть бути вибрані в точках E^n. Цільова функція може бути вичислена в кожній з вершин симплексу; з вершини, де цільова функція максимальна (точка А на рис. 4.7), проводиться проекція прямої через центр тяжіння симплексу. Потім точка А виключається і будується новий симплекс, званий відбитим, з колишніх точок, що залишилися, і однієї нової точки В, розташованої на проекції прямій на належній відстані від центру тяжіння. Продовження цієї процедури, в якій кожного разу викреслюється вершина, де цільова функція максимальна, а також використання правил зменшення розміру симплексу і запобігання циклічному руху в околиці екстремуму дозволяють здійснити пошук, що не використовує похідні і в якому величина кроку на будь-якому етапі k фіксована, а напрям пошуку можна змінювати.

У методі Нелдера і Міду мінімізується функція n незалежних змінних з використанням n+1 вершин деформованого багатогранника, в E^n. Кожна вершина може бути ідентифікована вектором x. Вершина (точка) в E^n, в якій значення f(x) максимально, проектується через центр тяжіння (центроїд) вершин, що залишилися. Поліпшені (нижчі) значення цільової функції знаходяться послідовною заміною точки з максимальним значенням f(x) на більш "хороші точки", поки не буде знайдений мінімум f(x).

Нехай $x_i^{(k)} = \left[x_{i1}^{(k)}, \ldots, x_{ij}^{(k)}, \ldots, x_{in}^{(k)}\right]^T, i = 1, \ldots, n+1,$ є i-ою вершиною (точкою) в E^n на k-му етапі пошуку, k=0, 1,..., і нехай значення цільової функції в $x^{(k)}_i$ дорівнює $f(x^{(k)}_i)$. Крім того, відмітимо ті вектори x многогранника, які дають максимальне і мінімальне значення f(x).

Визначимо

$$f\left(x_h^{(k)}\right) = \max\left\{f\left(x_1^{(k)}\right), \ldots, f\left(x_{n+1}^{(k)}\right)\right\},$$

де $x_h^{(k)} = x_i^{(k)}$,

$$f\left(x_l^{(k)}\right) = \min\left\{f\left(x_1^{(k)}\right), \ldots, f\left(x_{n+1}^{(k)}\right)\right\},$$

де $x_l^{(k)} = x_i^{(k)}$.

Оскільки многогранник в E^n складається з (n+1) вершин $x_1,...,$ x_{n+1}, нехай x_{n+2} буде центром тяжіння усіх вершин, виключаючи x_h.

Тоді координати цього центру визначаються формулою

$$x_{n+2,j}^{(k)} = \frac{1}{n}\left[\left(\sum_{i=1}^{n+1} x_{ij}^{(k)}\right) - x_{hj}^{(k)}\right], \quad j = 1, \ldots, n, \tag{4.1}$$

де індекс j означає координатний напрям.

Початковий багатогранник зазвичай вибирається у вигляді регулярного симплексу (але це не обов'язково) з точкою 1 як початок коор-

динат; можна початок координат помістити в центр тяжіння. Процедура пошуку вершини в E^n, в якій f(x) має краще значення, складається з наступних операцій:

1) Віддзеркалення - проектування x(k)h через центр тяжіння відповідно до співвідношення

$$x_{n+3}^{(k)} = x_{n+2}^{(k)} + \alpha\left(x_{n+2}^{(k)} - x_h^{(k)}\right), \tag{4.2}$$

де $\alpha > 0$ є коефіцієнтом віддзеркалення; $x_{n+2}^{(k)}$ - центр тяжіння, що обчислюється за формулою (1); $x_h^{(k)}$ - вершина, в якій функція f(x) приймає найбільше з n+1 значень на k -му етапі.

2) Розтягування. Ця операція полягає в наступному: якщо $f\left(x_{n+3}^{(k)}\right) \le f\left(x_l^{(k)}\right)$, то вектор $\left(x_{n+3}^{(k)} - x_{n+2}^{(k)}\right)$ розтягується відповідно до співвідношення

$$x_{n+4}^{(k)} = x_{n+2}^{(k)} + \gamma\left(x_{n+3}^{(k)} - x_{n+2}^{(k)}\right), \tag{4.3}$$

де $\gamma > 1$ є коефіцієнт розтягування. Якщо $f\left(x_{n+4}^{(k)}\right) < f\left(x_l^{(k)}\right)$, то $x_h^{(k)}$ замінюється на $x_{n+4}^{(k)}$ і процедура триває знову з операції 1 при k=k+1. Інакше $x_h^{(k)}$ замінюється на $x_{n+3}^{(k)}$ і також здійснюється перехід до операції 1 при k=k+1.

3) Стискування. Якщо $f\left(x_{n+3}^{(k)}\right) > f\left(x_i^{(k)}\right)$ для усіх i≠h, то вектор $\left(x_h^{(k)} - x_{n+2}^{(k)}\right)$ стискується відповідно до формули

$$x_{n+5}^{(k)} = x_{n+2}^{(k)} + \beta\left(x_h^{(k)} - x_{n+2}^{(k)}\right), \tag{4.4}$$

де $0<\beta<1$ є коефіцієнт стискування. Потім $x_h^{(k)}$ замінюємо на $x_{n+5}^{(k)}$ і повертаємося до операції 1 для продовження пошуку на (k+1)-му кроці.

4) Редукція. Якщо $f\left(x_{n+3}^{(k)}\right) > f\left(x_{h}^{(k)}\right)$, усі вектори $\left(x_{i}^{(k)} - x_{l}^{(k)}\right), i = 1, \ldots, n+1,$ зменшуються в 2 рази з відліком від $x_{l}^{(k)}$ відповідно до формули

$$x_{i}^{(k)} = x_{l}^{(k)} + 0{,}5\left(x_{i}^{(k)} - x_{l}^{(k)}\right), \quad i = 1, \ldots, n+1. \qquad (4.5)$$

Потім повертаємося до операції 1 для продовження пошуку на (k+1)-му кроці.

Критерій закінчення пошуку, використаний Нелдером і Мідом, полягав в перевірці умови

$$\left\{\frac{1}{n+1}\sum_{i=1}^{n+1}\left[f\left(x_{i}^{(k)}\right) - f\left(x_{n+2}^{(k)}\right)\right]^2\right\}^{1/2} \le \varepsilon, \qquad (4.6)$$

де ε - довільне мале число, а $f\left(x_{n+2}^{(k)}\right)$ - значення цільової функції в центрі тяжіння $x_{n+2}^{(k)}$.

На рис. 4.8 приведена блок-схема пошуку оптимуму методом деформованого багатогранника. Деформований багатогранник, в протилежність жорсткому симплексу адаптується до топографії цільової функції, витягуючись уздовж довгих похилих площин, змінюючи напрям в зігнутих западинах і стискуючись в околиці мінімуму.

Коефіцієнт віддзеркалення α використовується для проектування вершини з найбільшим значенням f(x) через центр тяжіння багатогранника, що деформується. Коефіцієнт γ вводиться для розтягування вектора пошуку у разі, якщо віддзеркалення дає вершину зі значенням f(x), меншим, ніж найменше значення f(x), отримане до віддзеркалення.

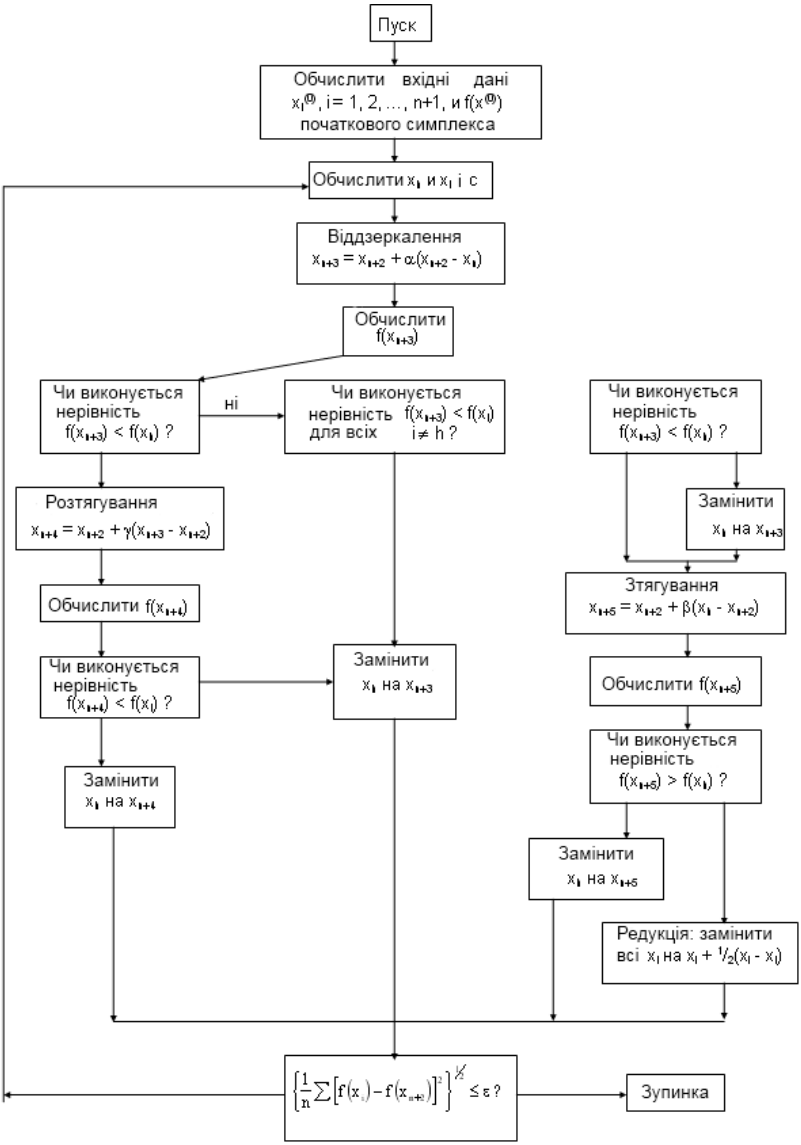


Рисунок 4.8 - Блок-схема методу деформованого багатогранника

Коефіцієнт стискування β використовується для зменшення вектора пошуку, якщо операція віддзеркалення не привела до вершини зі значенням f(x), меншим, ніж друге за величиною (після найбільшого) значення f(x), отримане до віддзеркалення.

Таким чином, за допомогою операцій розтягувань або стискування розміри і форма деформованого багатогранника, масштабуються так, щоб вони задовольняли топології вирішуваної задачі.

Нелдер і Мід показали, що при рішенні задачі з $\alpha=1$ вимагається менша кількість обчислень функції, чим при $\alpha<1$. Як задовільні значення параметрів β і γ при оптимізації без обмежень Нелдер і Мід рекомендували $\alpha=1$, $\beta=0,5$ і $\gamma=2$. При $0<\beta<0,4$ існує вірогідність того, що із-за сплощення багатогранника матиме місце передчасне закінчення процесу. При $\beta>0,6$ може знадобитись надлишкове кількість кроків і більше машинного часу для досягнення остаточного рішення.

Лістинг функції розрахунку оптимального значення заданого параметра асинхронного двигуна приведений нижче. У прикладі варіюються значення висоти і довжини повітряного проміжку.

```
//Реалізація методу дефомованого багатогранника (Нелдера - Міду)
//Знаходження мінімуму цільової функції
void optimize()
{
float x[7][3], xh[2], xL[2];
float fx[7], fxh, fxL;
float e = 0.001;
int di[3];
int n, m; //номери індексів перших трьох векторів, відмінних від h
void up(float [][3], float* dimfx);
void down(float [][3], float* dimxh, float* dimfx);
void reduce(float [][3], float* dimxL, float* dimfx);
void mirror(float [][3], float* dimxh, float* dimfx);
void center(float [][3], float* dimxh, float* dimfx);
```

```
void sort_x(float [][3], float* dimfx, float* fh);
void find_fxhL(float [][3], float* dimfx, float* dimxh, float* dimxL, float* fh,
float* fL, int* dim_index);

ld_min = lamda_min*tau;
ld_max = lamda_max*tau;
delta = delta_fix;
ld = ld_fix;
//Введення початкових точок симплексу
x[0][0] = ld_min;
x[0][1] = delta_min;
x[1][0] = ld_max;
x[1][1] = delta_max;
x[2][0] = ld_min;
x[2][1] = delta_max;
//Розрахунок вхідних точок
//x0
ld = x[0][0];
delta = x[0][1];
auto_func1();//функція автоматичного розрахунку двигуна
fx[0] = opt;
//x1
ld = x[1][0];
delta = x[1][1];
auto_func1();
fx[1] = opt;
//x2
ld = x[2][0];
delta = x[2][1];
auto_func1();
fx[2] = opt;
//Знаходження вектору з максимальним і мінімальним значенням функції
find_fxhL(x, fx, xh, xL, &fxh, &fxL, di);
n = di[0];
m = di[1];
//Визначення індексу hx
//sort_x(x, fx, &fxh);
float eps = 10000;
number = 0;
while (eps > e)// Доки не досягнута задана точність
{
//Знаходження центру симплексу
center(x, xh, fx);
```

```
//Розрахунок віддзеркалення і значення функції в точці //віддзеркалення
mirror(x, xh, fx);
if (fx[4] < fxh)
{
	//Розрахунок розтягування
	up(x, fx);
	if (fx[5] < fxL)
	{
		xh[0] = x[5][0];
		xh[1] = x[5][1];
	}
	else
	{
		xh[0] = x[4][0];
		xh[1] = x[4][1];
	}
}
else if ((fx[4] < fx[n]) && (fx[4] < fx[m]))
{
	if (fx[4] < fxh)
		//Розрахунок стискування
		down(x, xh, fx);
	else
	{
		xh[0] = x[4][0];
		xh[1] = x[4][1];
		//Розрахунок стискування
		down(x, xh, fx);
	}
	if (fx[6] > fxh)
	{
		//Редукція
		reduce(x, xL, fx);
		find_fxhL(x, fx, xh, xL, &fxh, &fxL, di);
		n = di[0];
		m = di[1];
	}
	else
	{
		xh[0] = x[6][0];
		xh[1] = x[6][1];
	}
}
```

```
else if ((fx[4] >= fx[n]) || (fx[4] >= fx[m]))
{
        xh[0] = x[4][0];
        xh[1] = x[4][1];
}
int sum = 0;
for (int i = 0; i <= 2; i++)
{
        sum +=(fx[i] - fx[3]);
}
eps = sqrt(sum/3);
number++;
}
//Остаточний розрахунок по знайдених оптимальних значеннях
ld = x[3][0];
delta = x[3][1];
auto_func1();
//Виведення даних на друк у файл
DataOutput(OPT_GEOM, "Data_optimum.txt");
}
//Розрахунок координат перших трьох елементів, знаходження min і max
void find_fxhL(float dimx[][3], float* dimfx, float* dimxh, float* dimxL,
float* fh, float* fL, int* dim_index)
{
float min = 1000;
int imin, imax;
float max = 0;
for (int i = 0; i <=2; i++)
{
        if (dimfx[i] < min)
        {
                min = dimfx[i];
                imin = i;
        }
        if (dimfx[i] > max)
        {
                max = dimfx[i];
                imax = i;
        }
}
//Занесення в масив dim_inex усі індекси, відмінні від h
int index = 0;
for (int i = 0; i<= 2; i++)
```

```
{
	if (i != imax)
	{
		dim_index[index] = i;
		index++;
	}
}
//Координати максимуму
dimxh[0] = dimx[imax][0];
dimxh[1] = dimx[imax][1];
*fh = dimfx[imax];
//Координати мінімуму
dimxL[0] = dimx[imin][0];
dimxL[1] = dimx[imin][1];
*fL = dimfx[imin];
}
//Впорядковування перших трьох елементів і знаходження xh
void sort_x(float dimx[][3], float* dimfx, float* fh)
{
float ftemp[2];
int ih = 0;
for (int i = 0; i <= 2; i++)
{
	if (dimfx[i] != *fh)
	{
		ftemp[ih] = dimfx[i];
		ih++;
	}
}
dimfx[0] = ftemp[0];
dimfx[1] = ftemp[1];
dimfx[2] = *fh;
}
//Розрахунок центру симплексу (# 3)
void center(float dimx[][3], float* dimxh, float* dimfx)
{
float sum;
for (int j = 0; j <=1; j++)
{
	sum = 0;
	for (int i = 0; i <=2; i++)
	{
		sum += dimx[i][j];
```

```
        }//i
        dimx[3][j] = 0.5*(sum - dimxh[j]);
}//j
ld = dimx[3][0];
delta = dimx[3][1];
auto_func1();
find_pay1();
target1();//Розрахунок цільової функції F
dimfx[3] = F;
}
//Розрахунок віддзеркалення (# 4)
void mirror(float dimx[][3], float* dimxh, float* dimfx)
{
float alfa = 1;
for (int j = 0; j<= 1; j++)
        dimx[4][j] = dimx[3][j] + alfa*(dimx[3][j] - dimxh[j]);

ld = dimx[4][0];
delta = dimx[4][1];
auto_func1();
find_pay1();
target1();//Розрахунок цільової функції F
dimfx[4] = F;
}
//Розрахунок розтягування симплексу (# 5)
void up(float dimx[][3], float* dimfx)
{
float gamma = 2.8;
for (int j = 0; j <= 1; j++)
        dimx[5][j] = dimx[3][j] + gamma*(dimx[4][j] - dimx[3][j]);

ld = dimx[5][0];
delta = dimx[5][1];
auto_func1();
find_pay1();
target1();//Розрахунок цільової функції F
dimfx[5] = F;
}
//Розрахунок стискування симплексу (# 6)
void down(float dimx[][3], float* dimxh, float* dimfx)
{
float betta = 0.5;
for (int j = 0; j <= 1; j++)
```

```
        dimx[6][j] = dimx[3][j] + betta*(dimxh[j] - dimx[3][j]);
ld = dimx[6][0];
delta = dimx[6][1];
auto_func();
find_pay1();
target1();//Розрахунок цільової функції F
dimfx[6] = F;
}
//Розрахунок редукції симплексу
void reduce(float dimx[][3], float* dimxL, float* dimfx)
{
for (int j = 0; j <= 1; j++)
{
        for (int i = 0; i <= 2; i++)
        {
                dimx[i][j] = dimxL[j] + 0.5*(dimx[i][j] - dimxL[j]);
        }//i
}//j
//Розрахунок нових вхідних точок
//x0
ld = dimx[0][0];
delta = dimx[0][1];
auto_func1();
find_pay1();
target1();
dimfx[0] = F;
//x1
if (dimx[1][0] > ld_max)
        dimx[1][0] = ld_max;
else if (dimx[1][0] < ld_min)
        dimx[1][0] = ld_min;
if (dimx[1][1] > delta_max)
        dimx[1][1] = delta_max;
else if (dimx[1][1] < delta_min)
        dimx[1][1] = delta_min;
ld = dimx[1][0];
delta = dimx[1][1];
auto_func1();
find_pay1();
target1();
dimfx[1] = F;
//x2
ld = dimx[2][0];
```

```
delta = dimx[2][1];
auto_func1();
find_pay1();
target1();
dimfx[2] = F;
}
```

4.4 Метод зовнішніх штрафних функцій

Завдання оптимізації бувають безумовні і умовні. Безумовне завдання оптимізації полягає у відшукуванні максимуму або мінімуму дійсної функції від n дійсних змінних і визначенні відповідних значень аргументів на деякій безлічі σ n -мірного простору.

Умовні завдання оптимізації (завдання з обмеженнями) - це такі завдання, при формулюванні яких задаються деякі умови (обмеження) на множині σ у вигляді сукупності деяких функцій, що задовольняють рівнянням або нерівностям.

В результаті обмежень область проектування σ, визначувана усіма n проектними параметрами, може бути істотно зменшена відповідно до фізичної суті завдання.

Обмеження-рівності виражають залежність між проектними параметрами, які повинні враховуватися при знаходженні рішення. Ці обмеження відбивають закони природи, наявність ресурсів і т. д.

Кількість m обмежень-рівності може бути довільним:

$g_1(x_1, x_2, \ldots, x_n) = 0;$

$g_2(x_1, x_2, \ldots, x_n) = 0;$

……;

$g_m(x_1, x_2, \ldots, x_n) = 0.$

У ряді випадків з цих співвідношень можна виразити одні проектні параметри через інші. Це дозволяє виключити деякі параметри з

процесу оптимізації і зменшити розмірність завдання, тобто полегшити її рішення.

Аналогічно вводяться обмеження-нерівності:

$\alpha_1 \le \gamma_1(x_1, x_2, \ldots, x_n) \le b_1$;

$\alpha_2 \le \gamma_2(x_1, x_2, \ldots, x_n) \le b_2$;

…… ;

$\alpha_k \le \gamma_k(x_1, x_2, \ldots, x_n) \le b_k$.

Якщо є обмеження, то оптимальне рішення може відповідати або локальному екстремуму (max або min) усередині області проектування, або значенню цільової функції на межі області.

Рішення умовних завдань оптимізації (завдань з обмеженнями) більш трудомісткі в порівнянні із завданнями безумовної оптимізації, оскільки обмеження типу рівностей або нерівностей вимагають їх урахування на кожному кроці оптимізації. Тому їх рішення прагнуть звести до рішення послідовності завдань безумовної оптимізації. На цьому і заснований метод штрафних функцій. Суть методу полягає в наступному.

Нехай $f(x_1, x_2, \ldots, x_n)$ - цільова функція, для якої треба знайти мінімум m в обмеженій області D ($x_1, x_2, \ldots, x_n \in D$). Замінимо це завдання завданням про безумовну оптимізацію (мінімізацію) однопараметричного сімейства функцій

$$F(x,\beta) = f(x) + \frac{1}{\beta}\varphi(x), x = \{x_1, x_2, ..., x_n\} \cdot \qquad (4.7)$$

При цьому додаткову (штрафну) функцію $\varphi(x)$ виберемо так, щоб при $\beta \to 0$ рішення допоміжної задачі прагнуло до рішення початкової або, принаймні, щоб їх мінімуми співпадали: $\min F(x, \beta) \to m$ при $\beta \to 0$.

Штрафна функція φ(x) повинна враховувати обмеження, які задаються при постановці завдання оптимізації. Якщо є обмеження-нерівності виду $g_j\ (x_1, x_2, \ldots, x_n) \geq 0\ \ (j = 1, 2, \ldots, J)$, то як штрафну можна узяти функцію, яка:

1) дорівнює нулю в усіх точках простору проектування, що задовольняють заданим обмеженням-нерівностям;

2) прагне до нескінченності в тих точках, в яких ці нерівності не виконуються.

Таким чином, при виконанні обмеженнь-нерівностей функції f(x) і F(x,β) мають один і той же мінімум.

Якщо хоч би одна нерівність не виконається, то допоміжна цільова функція F(x, β) отримує нескінченно великі добавки, і її значення стає далеким від мінімуму функції f(x). Іншими словами, при недотриманні обмеженнь-нерівностей на цільову функцію накладається "штраф". Звідси і термін "метод штрафних функцій".

Розглянемо випадок, коли в завданні оптимізації задані обмеження двох типів - рівність і нерівність:

$$\begin{aligned} &g_i(x) = 0, i = 1,2,...,I; \\ &h_j(x) > 0, j = 1,2,...,J; x = \{x_1, x_2, ..., x_n\} \end{aligned} \quad . \tag{4.8}$$

В цьому випадку як допоміжна цільова функція, для якої формується завдання безумовної оптимізації в усьому n-мірному просторі, приймають функцію

$$F(x,\beta) = f(x) + \frac{1}{\beta}\left\{\sum_{i=1}^{I} g_i^2(x) + \sum_{j=1}^{J} h_j^2(x)\left[1 - \text{sign}\ \ h_j(x)\right]\right\}, \beta > 0, \tag{4.9}$$

де sign(x) - "сигнум", тобто функція від дійсних змінних, дорівнює +1 для позитивних x, набуває значення нуль при x = 0 і дорівнює -1 для негативних x.

Тут узята така штрафна функція, що при виконанні умов (4.8) вона перетворюється на нуль. Якщо ж ці умови порушені ($g_i(x) \neq 0$, $h_j(x) < 0$ і $\text{sign}\, h_j(x) = -1$), то штрафна функція позитивна. Вона збільшує цільову функцію f(x) тим більше, чим більше порушуються умови (4.8).

При малих значеннях β поза областю D функція F(x, β) сильно зростає. Тому її мінімум може бути або усередині D, або зовні поблизу меж цієї області. У першому випадку мінімуми функцій F(x, β) і f(x) співпадають, оскільки додаткові члени в (4.9) дорівнюють нулю. Якщо мінімум функції F(x, β) знаходиться зовні D, то мінімум цільової функції f(x) лежить на межі D. При цьому можна побудувати таку послідовність $\beta_k \to 0$, що відповідна послідовність мінімумів функції F(x, β) прагнутиме до мінімуму функції f(x).

Таким чином, завдання оптимізації для цільової функції f(x) з обмеженнями (4.8) звелося до послідовності завдань безумовної оптимізації для допоміжної функції (4.9), рішення яких може бути проведене за допомогою методу координатного спуску. При цьому будується ітераційний процес при $\beta \to 0$.

Розглянемо алгоритм, що реалізовує метод штрафних функцій.

Набудемо довільного значення вагового коефіцієнта штрафу β. Позначимо як G максимальний компонент вектора обмежень.

Тоді цільову функцію двох змінних x, y можна записати у вигляді:

$$F(x,y) = f(x,y) + \beta \sum_{i=1}^{I\max} G_i, \qquad (4.10)$$

де I_{max} - кількість заданих обмежень.

Для визначення G усі обмеження нормуються відповідно до виразу:

$$\phi_{гі} = \frac{g_{гі} - [g_{гі}]_{доп}}{[g_{гі}]_{доп}}, i = 1,7. \qquad (4.11)$$

де $g_{гі}$, $[g_{гі}]_{доп}$ - поточне і допустиме значення контрольованого показника.

Тоді, відповідно до (4.9)

$$G_i = \left\{ \varphi_{гі} \frac{[1 \pm \text{sign}(\varphi_{гі})}{2} \right\}, \qquad (4.12)$$

де $\text{sign} = \begin{Bmatrix} +1 - \text{при } \varphi_{гі} > 0 \\ -1 - \text{при } \varphi_{гі} < 0 \end{Bmatrix}$.

Знак "+" відповідає обмеженням згори, а знак "-" - обмеженням знизу.

Тоді алгоритм пошуку оптимального значення функції можна представити у вигляді послідовності операцій:

1) формуємо програму пошуку оптимуму методом покоординатного спуску;

2) починаємо цикл зміни величини x; фіксуємо y; розраховуємо f(x);

3) розраховуємо компоненти вектора обмежень G у відповідність з (4.11), (4.12).

4) набуваємо значення цільової функції F за (4.10);

5) порівнюємо значення F з мінімумом;

6) повертаємось до п.2, поки не досягнуте максимальне значення x в циклі;

7) починаємо цикл зміни величини y; фіксуємо $x_{опт}$; розраховуємо f(y);

8) розраховуємо компоненти вектора обмежень G у відповідність з (4.12);

9) набуваємо значення цільової функції F за (4.10);

10) порівнюємо значення F з мінімумом;

11) повертаємося до п.7, поки не досягнуте максимальне значення у в циклі.

Після закінчення пошуку мінімуму функції методом покоординатного спуску, набудемо оптимального значення функції F при оптимальних значеннях аргументів x і y.

Реалізація методу зовнішніх штрафних функцій на мові C++, узята з проекту асинхронного двигуна з короткозамкненим ротором (буде детально розглянутий в наступному розділі), приведена нижче.

```
//Розрахунок штрафів для обмежень
void find_pay1()
{
// Коефіцієнт ld/tau max lamda
g1 = (lamda - lamda_max)/lamda_max;
sign = (g1 > 0)?1:- 1;
Gf[1] = g1*(1 + sign)/2;

//min lamda
g2 = (lamda - lamda_min)/lamda_min;
sign = (g2 > 0)?1:- 1;
Gf[2] = sign*g2*(1 - sign)/2;

// Індукція в повітряному проміжку max Bd
g1 = (Bd - Bd_max)/Bd_max;
sign = (g1 > 0)?1:- 1;
Gf[3] = g1*(1 + sign)/2;

//min Bd
g2 = (Bd - Bd_min)/Bd_min;
sign = (g2 > 0)?1:- 1;
Gf[4] = sign*g2*(1 - sign)/2;
```

```
//Лінійне навантаження max A
g1 = (A - A_max)/A_max;
sign = (g1 > 0)?1:- 1;
Gf[5] = g1*(1 + sign)/2;

//min A
g2 = (A - A_min)/A_min;
sign = (g2 > 0)?1:- 1;
Gf[6] = sign*g2*(1 - sign)/2;

//Індукція в зубцях статора max Bz1
g1 = (Bz1 - Bz1_max)/Bz1_max;
sign = (g1 > 0)?1:- 1;
Gf[7] = g1*(1 + sign)/2;

//min Bz1
g2 = (Bz1 - Bz1_min)/Bz1_min;
sign = (g2 > 0)?1:- 1;
Gf[8] = sign*g2*(1 - sign)/2;

//Індукція в зубцях ротора max Bz2
g1 = (Bz2 - Bz2_max)/Bz2_max;
sign = (g1 > 0)?1:- 1;
Gf[9] = g1*(1 + sign)/2;

//min Bz2
g2 = (Bz2 - Bz2_min)/Bz2_min;
sign = (g2 > 0)?1:- 1;
Gf[10] = sign*g2*(1 - sign)/2;

//Індукція в ярмі статора max Ba
g1 = (Ba - Ba_max)/Ba_max;
sign = (g1 > 0)?1:- 1;
Gf[11] = g1*(1 + sign)/2;

//min Ba
g2 = (Ba - Ba_min)/Ba_min;
sign = (g2 > 0)?1:- 1;
Gf[12] = sign*g2*(1 - sign)/2;

//Індукція в ярмі ротора max Bj
g1 = (Bj - Bj_max)/Bj_max;
sign = (g1 > 0)?1:- 1;
```

```
Gf[13] = g1*(1 + sign)/2;

//min Bj
g2 = (Bj - Bj_min)/Z2_min;
sign = (g2 > 0)?1:- 1;
Gf[14] = sign*g2*(1 - sign)/2;

//Густина струму обмотки статора max J1
g1 = (J1 - J1_max)/J1_max;
sign = (g1 > 0)?1:- 1;
Gf[15] = g1*(1 + sign)/2;

//min J1
g2 = (J1 - J1_min)/J1_min;
sign = (g2 > 0)?1:- 1;
Gf[16] = sign*g2*(1 - sign)/2;

//Густина струму обмотки ротора max J2
g1 = (J2 - J2_max)/J2_max;
sign = (g1 > 0)?1:- 1;
Gf[17] = g1*(1 + sign)/2;

//min J2
g2 = (J2 - Z2_min)/J2_min;
sign = (g2 > 0)?1:- 1;
Gf[18] = sign*g2*(1 - sign)/2;

//Коефіцієнт намагнічування  max km
g1 = (km - km_max)/km_max;
sign = (g1 > 0)?1:- 1;
Gf[19] = g1*(1 + sign)/2;

//min km
g2 = (km - km_min)/km_min;
sign = (g2 > 0)?1:- 1;
Gf[20] = sign*g2*(1 - sign)/2;

//Струм намагнічування  max Imo
g1 = (Imo - Imo_max)/Imo_max;
sign = (g1 > 0)?1:- 1;
Gf[21] = g1*(1 + sign)/2;
```

```
//min Imo
g2 = (Imo - Imo_min)/Imo_min;
sign = (g2 > 0)?1:- 1;
Gf[22] = sign*g2*(1 - sign)/2;

//Пусковий момент max Mp
g1 = (Mpo - Mp_max)/Mp_max;
sign = (g1 > 0)?1:- 1;
Gf[23] = g1*(1 + sign)/2;

//min Mp
g2 = (Mpo - Mp_min)/Mp_min;
sign = (g2 > 0)?1:- 1;
Gf[24] = sign*g2*(1 - sign)/2;

//Пусковий струм max Ip
g1 = (Ipo - Ip_max)/Ip_max;
sign = (g1 > 0)?1:- 1;
Gf[25] = g1*(1 + sign)/2;

//min Ip
g2 = (Ipo - Ip_min)/Ip_min;
sign = (g2 > 0)?1:- 1;
Gf[26] = sign*g2*(1 - sign)/2;

//Температура max dt1
g1 = (dt1 - dt1_max)/dt1_max;
sign = (g1 > 0)?1:- 1;
Gf[27] = g1*(1 + sign)/2;
//min dt1
g2 = (dt1 - dt1_min)/dt1_min;
sign = (g2 > 0)?1:- 1;
Gf[28] = sign*g2*(1 - sign)/2;

//Потік вентилятора Qv1 > Qv, що охолоджує
g2 = (Qv1 - Qv)/Qv;
sign = (g2 > 0)?1:- 1;
Gf[29] = sign*g2*(1 - sign)/2;

//Повітряний проміжок max delta
g1 = (delta - delta_max)/delta_max;
sign = (g1 > 0)?1:- 1;
Gf[30] = g1*(1 + sign)/2;
```

```
//min delta
g2 = (delta - delta_min)/delta_min;
sign = (g2 > 0)?1:- 1;
Gf[31] = sign*g2*(1 - sign)/2;

//Коефіцієнт заповнення паза max kzp
g1 = (kzp - kzp_max)/kzp_max;
sign = (g1 > 0)?1:- 1;
Gf[32] = g1*(1 + sign)/2;

//min kzp
g2 = (kzp - kzp_min)/kzp_min;
sign = (g2 > 0)?1:- 1;
Gf[33] = sign*g2*(1 - sign)/2;

Npay = 32;
}


//Розрахунок цільової функції для методу зовнішніх штрафних функцій
void target1()
{
float G = 0;
find_pay1();

for (int i = 0; i <= Npay; i++)
        G += Gf[i];

if (!KP1 || !KP2)
        F = 1000;
else
        F = opt + Kpay*G;
}
```

Контрольні питання

1. У чому полягає інтерполяція методом сплайнів? Який сплайн називається кубічним?

2. Різновиди сплайнів, вказати їх переважні сфери застосування.

3. Алгоритм і реалізація кубічної сплайн-інтерполяції. Організація доступу до даних, розміщених в текстових файлах.

4. Особливості методу покоординатного спуску, його переваги і недоліки. Застосування методу покоординатного спуску в рішенні задачі проектування асинхронного двигуна.

5. Алгоритм і реалізація методу покоординатного спуску. Розкрити характерні особливості методу.

6. Поняття умовної і безумовної оптимізації. Поняття обмежень і їх структура.

7. Призначення і реалізація методу зовнішніх штрафних функцій. Як правильно враховувати знак обмежень, що задаються?

8. Привести етапи знаходження оптимального значення цільової функції при використанні методу зовнішніх штрафних функцій.

Практичні завдання

1. Сформувати С++ програму в Visual Studio, забезпечуючи інтерполяцію методом сплайнів величини, аргументи і функції якої читаються з файлу даних. Як приклад можна використовувати табличні значення кривої намагнічування стали 2013 (дод. Д). При натисненні на кнопку "ОК", введене в перше поле Edit control значення магнітної індукції має бути передане в підпрограму сплайн-інтерполяції. У друге неактивне поле Edit control має бути виведене значення магнітної напруженості, набутого в результаті інтерполяції .

2. Сформувати на мові С++ програму в Visual Studio, що дозволяє знаходити екстремальне значення функції декількох змінних методом зовнішніх штрафних функцій з використанням методу покоординатного спуску. Рекомендується скласти програму знаходження максимуму електромагнітного моменту:

$$M = \frac{pmU_1^2 \frac{r_2'}{s}}{2\pi f_1 \left[\left(r_1 + \frac{r_2'}{s} \right)^2 + \left(x_1 + x_2' \right)^2 \right]},$$

де $p = 2$;

$m = 3$;

$U_1 = 380$ В;

$f_1 = 50$ Гц;

$r_1 = 0.4$ Ом;

$x_1 = 0{,}7$ Ом;

$r'_{2Б} = 0{,}27$ Ом;

$x'_{2Б} = 0{,}83$ Ом.

Приведений активний опір ротора знаходиться в прямій залежності від довжини повітряного проміжку l_δ, $r'_2 = l_\delta r'_{2Б}$. Діапазон варіювання $0 \le l_\delta \le 2$, $l_\delta(1) = 130$ мм.

Приведений індуктивний опір ротора, при постійному числі витків обмотки статора W_1, також знаходиться в прямій залежності від довжини повітряного проміжку l_δ, $x'_2 = l_\delta x'_{2Б}$.

Допустимий діапазон приведених опорів ротора :

$0{,}1 \le r'_2 \le 0{,}4$ Ом;

$0{,}4 \le x'_2 \le 1{,}2$ Ом.

Діапазон зміни ковзання $0{,}01 \le s \le 1$.

РОЗДІЛ 5

ПОВНИЙ ПРИКЛАД ОПТИМАЛЬНОГО ПРОЕКТУВАННЯ АСИНХРОННОГО ДВИГУНА З КОРОТКОЗАМКНЕНИМ РОТОРОМ

Даний розділ об'єднує в собі ті теоретичні відомості по оптимальному проектуванню електричних машин, які були розглянуті в навчальному посібнику раніше. Не показати, як використовувати отримані знання з практики, було б помилкою. Складемо програму автоматизованого проектування асинхронного двигуна з короткозамкненим ротором, починаючи з початкових даних і закінчуючи автоматизованою побудовою креслення валу.

Цей приклад слід оцінювати виключно з позиції учбового проектування і не переносити наведені положення на реальні проекти. Зокрема, це стосується вибору довжини і висоти повітряного проміжку як варійованих параметрів. При вивченні цього розділу рекомендується ознайомитися з літературою по оптимізації в техніці [8] і глибоко вивчити питання проектування електричних машин [9, 10].

Проект складатиметься з трьох основних частин:

а) розрахунку базової машини;

б) оптимізації базової машини;

в) генерації коду AutoLisp для створення креслення.

Усі розрахунки виконуються на мові C++ в середовищі Visual Studio. За роботу!

5.1 Неоптимізований розрахунок асинхронного двигуна

Проектування виконується за прикладом розрахунку асинхронного двигуна з короткозамкненим ротором, приведеному в [9]. Порядок розрахунку формується з послідовних виразів в 11 функціях проекту. Кожна функція викликається на відповідній сторінці майстра проекту при натисненні на кнопку «Прийняти», що буде розглянуто нижче.

Програму проектування слід почати з побудови головного програмного модуля. Він є діалоговим вікном з номінальними даними і трьома керуючими кнопками. У прикладі на рис. 5.1 - це "Project". Реакція на натиснення вказаної кнопки зв'язується із створенням діалогового вікна майстра, що складається з 11 сторінок. Функціональне призначення кожної сторінки відповідає порядку розрахунку у відповідних підрозділах прикладу в [9].

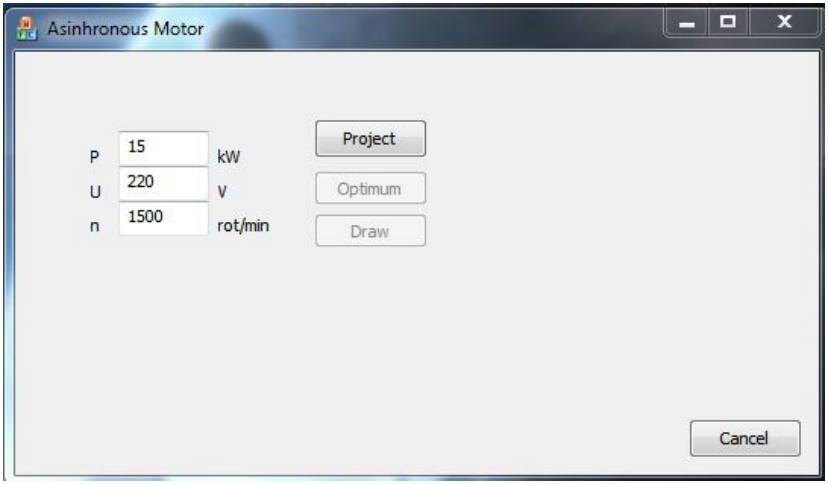


Рисунок 5.1 - Головне діалогове вікно програми

Кнопка "Optimum" викликає процедуру оптимізації.

Кнопка "Draw" викликає підпрограму створення коду параметричного креслення валу електродвигуна на мові AutoLisp.

Ці кнопки мають бути неактивними. Активація виконується після завершення розрахунку первинного неоптимізованого варіанту, тобто пов'язана з реакцією на кнопку "Готово" майстра.

Усі дійсні змінні проекту мають бути типу *float*. Геометричні розміри вносяться до вікон програми в [мм], потужність - в [кВт], переріз - [мм^2], лінійне навантаження - [А/мм], густина струму - [А/мм^2]. При зчитуванні цих даних з вікна командою UpdateData(), повинне виконуватися їх перетворення в систему СІ. Умовні позначення основних величин наведені у додатку З.

Усі розрахункові модулі програми розміщуються у файлі "program.cpp", змінні оголошуються у файлі "program.h". У заголовному файлі, передусім, необхідно включити бібліотеки для роботи з потоками, файлами і математичними функціями:

```
//program.h заголовний файл розрахунку оптимального АД
#include <iostream> //потоки
#include <fstream> //файли
#include <cmath> //математика
```

Початок файлу "program.cpp" повинен виглядати таким чином:

```
//program.cpp файл розрахунку оптимального АД
#include "stdafx.h"
#include "prog.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
```

```
//Прототипи зовнішніх функцій
extern float spline(float, char*, char*); //сплайн-інтерполяція з файлів
extern float spline(float x, float*, float*); //сплайн-інтерполяція з масивів
extern void auto_func(); //функція автоматичного розрахунку проекту
extern void DataOutput(int, char* f); //виведення даних у файл
extern void find_pay(); //реалізація методу штрафних функцій
extern void target(); //розрахунок цільової функції
extern void price(); //розрахунок вартості двигуна

using namespace std;
```

Як видно з приведеного лістингу, файл програми починається з оголошення прототипів допоміжних функцій. Ці функції потрібні як для автоматизації зчитування інформації з файлів даних, так і для розрахункових цілей.

Деякі з функцій вже були детально розглянуті в попередніх розділах:

- сплайн-інтерполяція: *spline()*;

- штрафна функція: *find_pay()*;

- розрахунок цільової функції з урахуванням штрафів: *target()*.

Функції оптимізації будуть розглянуті у відповідному розділі нижче (реалізують метод покоординатного спуску і розрахунок вартості двигуна).

Функція сплайн-інтерполяції оголошена двічі: перша, з символьними аргументами, призначена для роботи з файлами; друга, з дійсними аргументами - для роботи з дійсними масивами. Тобто, тут застосована одна з особливостей мови C++, яка носить назву поліморфізм. Таким чином, ми викликатимемо функцію *spline* як для файлів даних, так і для масивів, але з абсолютно однаковим зверненням.

Допоміжні функції, оголошені в секції прототипів, викликатимуться з головних функцій проекту. Метод покоординатного спуску не

викликатиметься з розрахункових модулів програми, а є незалежним програмним елементом.

Отже, при натисненні на кнопку Project головного вікна програми, викликається функція первинної ініціалізації *init()*, після чого слідує код запуску майстра, який складається з 11 сторінок. Код функції ініціалізації :

```
void init()
{
kf = 4/pi; //коефіцієнт форми поля
p = 60*f1/n; //кількість пар полюсів
dp = p*2; /кількість полюсів
omega = pi*n/30; //кутова швидкість поля
}//кінець функції init()
```

Останнє, що хотілося б відмітити перед початком роботи з майстром проекту - це мінімум автоматичних операцій на стадії розрахунку базової машини. Передбачається, що студент працює з підручником, вибираючи по таблицях і графіках необхідні коефіцієнти і величини, які потім вводить у вікна майстра. Звичайно ж, усі графіки і таблиці за бажанням можна занести у файли даних, і виконувати визначення величин методом сплайн-інтерполяції. Проте, ця робота вимагає великих витрат за часом, яке обмежене учбовим семестром, а наявність ручних операцій не перешкоджає розкриттю понять оптимального автоматизованого проектування.

Також не виконується розрахунок тих величин, які необхідно округлювати до цілих або із заданою точністю. Мета навчального посібника - не охопити усі трюки і методи програмування - це завдання іншого курсу, а застосувати мінімальну базу знань C++ для вирішення поставлених завдань.

5.1.1 Вибір головних розмірів. При виборі головних розмірів необхідно отримати відношення довжини повітряного проміжку до полюсного кроку $\lambda = l_\delta / \tau$, близьке до одиниці. Таке співвідношення, при фіксованому внутрішньому діаметрі статора D, дає можливість для варіювання l_δ в широкому діапазоні. Сторінка 1 майстра програми оформляється, як показано на рис. 5.2.

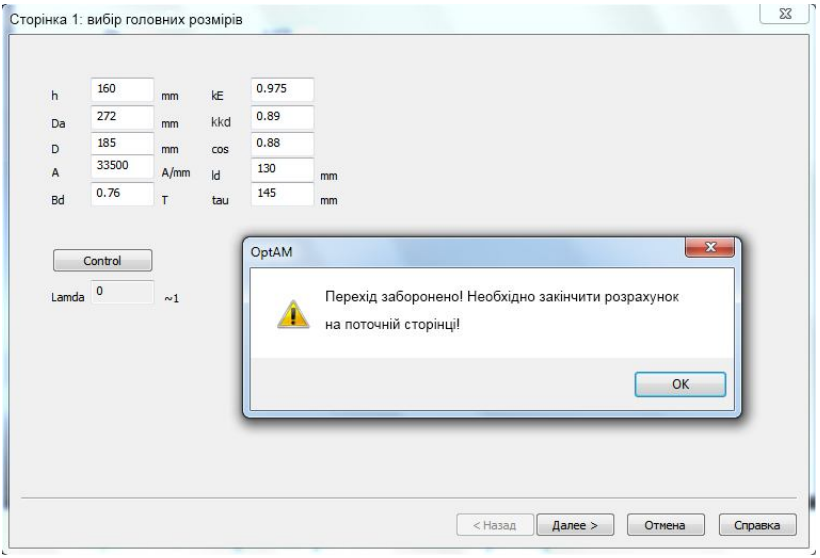


Рисунок 5.2 - Сторінка розрахунку головних розмірів

Заздалегідь розраховані значення l_δ і D округляються до цілих значень міліметра. Інші величини вибираються по рисунках і таблицях [9]. Натиснення на кнопку "Далі" має бути заблоковане, якщо не був виконаний розрахунок (кнопка "Control") на поточній сторінці, що повинне також супроводжуватися повідомленням програми (див. рис.

5.2). Неактивним має бути контрольне поле Lamda, в яке виводиться відношення $\lambda = l_\delta / \tau$.

При натисненні на кнопку Control майстра викликається наступна функція:

```
//Сторінка 1: вибір головних розмірів
void fpage1()
{
P1 = Pn*kE/(kkd*cosFi);
lamda = ld/tau;
}
```

У наведеній функції знаходиться розрахункова потужність і величина λ.

Підбір значень у вікні виконується до тих пір, поки значення λ не наблизиться до одиниці. Значення полюсного ділення $\tau = \frac{\pi D}{2p}$ визначається за простою формулою, проте результат розрахунку має бути округлений до цілих долів міліметра.

5.1.2 Визначення Z1, W1 і перерізу проводу обмотки статора. В усіх варіантах завдання, підготовлених для учбового проектування, потужність електродвигунів не перевищуватиме 15 кВт, що дає можливість застосувати одношарову обмотку. Це значно спростить майбутню програму оптимізації. При виборі Up звернути увагу на те, що це число має бути цілим. Значення W1 округляється до цілих значень міліметра. На рис. 5.3 показана сторінка майстра програми. При натисненні на кнопку "Apply" виробляється обчислення мінімального і максимального числа зубців статора за відомими формулами. До виконання цієї операції кнопка "Control" основній частині розрахунку на поточній сторінці має бути деактивованою.

Натиснення на кнопку "Далі" має бути заблоковане, якщо не був виконаний розрахунок (кнопка "Control") на поточній сторінці. Ця вимога відноситься до усіх сторінок майстра.

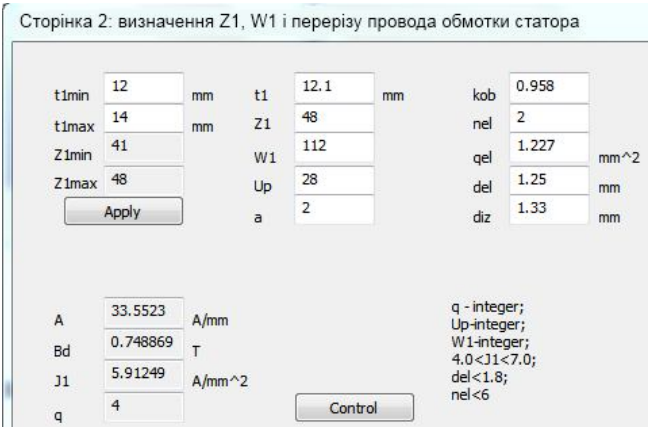


Рисунок 5.3 - Сторінка майстра за розрахунком обмотки статора

Текстові поля Z1max, Z1min вікна мають бути типу "Read only" (тільки для читання) і заповнюються автоматично після розрахунку діапазону числа зубців. При натисненні на кнопку Control викликається друга розрахункова функція програми :

```
//Сторінка 2: визначення Z1, W1 і перерізу проводу обмотки статора
void fpage2()
{
I1n = Pn/(m*U*cosFi*kkd); //номінальний струм
A = 2*I1n*W1*m/(pi*D); //лінійне навантаження
Fi = ke*U/(4*kB*W1*kob*f1); //магнітний потік
Bd = p*Fi/(D*ld); //індукція в повітряному проміжку
qef = nel*qel; //ефективний переріз провіду
J1 = I1n/(a*qef); //густина струму обмотки статора
q = Z1/(dp*m);
}
```

З коментарів до програми зрозумілі виконувані операції.

5.1.3 Розрахунок розмірів зубцевої зони статора і повітряного проміжку. При виконанні розрахунку, з метою отримання наочних результатів оптимізації, слід приймати наскільки можливо мінімальні значення магнітних індукцій B_{δ}, B_{z1}, B_a. Коефіцієнт заповнення паза повинен знаходитися в допустимому діапазоні. Для усіх варіантів розрахунку вибирати трапецієвидну конструкцію паза з постійним перерізом зубців (9, рис. 6-19, а). Клас ізоляції обмотки статора - F (155 ^{o}C).

Приклад оформлення сторінки майстра показаний на рис. 5.4.

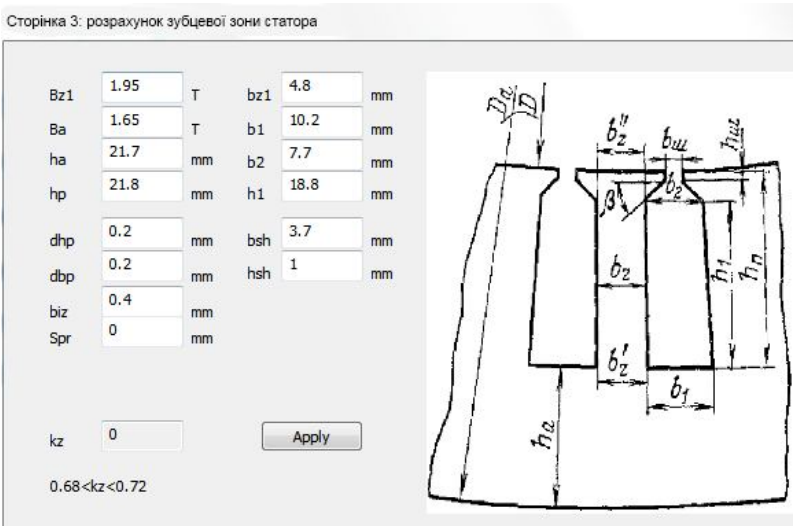


Рисунок 5.4 - Сторінка розрахунку зубцевої зони статора

Натиснення на кнопку Apply призводить до запуску третьої розрахункової функції, в якій визначаються розміри паза, площа ізоляції, площа паза і контрольна величина поточної сторінки - коефіцієнт заповнення паза. Останній повинен знаходитися в межах, вказаних в написі нижче вікна виведення значення коефіцієнту заповнення паза k_z.

```
//Сторінка 3: розрахунок зубцевої зони статора
void fpage3()
{
b11 = b1 - dbp;
b21 = b2 - dbp;
h11 = h1 - dhp;
Siz = biz*(2*hp1 + b1 + b2); // площа ізоляції
Sp1 = (b11 + b21)*h11/2 - Siz - Spr; // площа паза
kzp = diz*diz*Up*nel/Sp1; //коэфф. заповнення паза
}
```

5.1.4 Розрахунок ротора. Як і в п. 5.1.3, приймати наскільки можливо мінімальні значення магнітних індукцій B_{z1}, B_j. Розмір провода нижньої частини паза b_2 не повинен бути менше допустимого значення. Для всіх варіантів розрахунку вибирати грушовидний напівзакритий паз з постійним перерізом зубців (9, рис. 1, 6-27, а). Сторінка майстра показана на рис. 5.5.

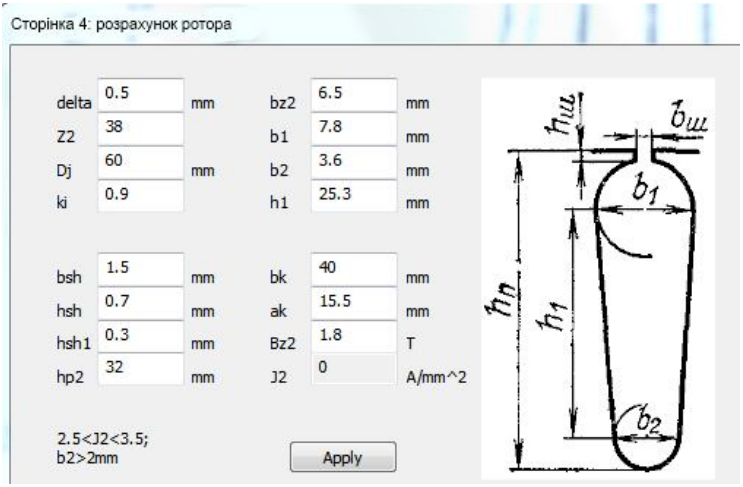


Рисунок 5.5 - Розрахунок ротора

При розрахунку паза ротора контрольними величинами є густина струму в роторі і мінімальний радіус нижньої частини паза. Функція розрахунку:

```
//Сторінка 4: розрахунок ротора
void fpage4()
{
D2 = D - delta; //зовнішній діаметр ротора
l2 = ld;//довжина пакету ротора
t2 = pi*D2/Z2; //зубцеве ділення
nui = 2*m*W1*kob/Z2;
I2 = ki*I1n*nui; //струм в стержні ротора
qc2 = pi*(b1r*b1r + b2r*b2r)/8 + 0.5*h1r*(b1r + b2r); //переріз стержня
J2 = I2/qc2; //густина струму в роторі
ddr = 2*sin(pi*p/Z2);
Ik = I2/ddr; //струм в к.з. кільці
Jk = 0.85*J2;//густина струму в к.з. кільці
qk = Ik/Jk; //переріз к.з. кільця
Dk = D2 - bk; //діаметр к.з. кільця
}
```

5.1.5 Розрахунок струму намагнічування. При розрахунку струму намагнічування значення магнітних напруженістей вибираються залежно від величини магнітних індукцій по таблицях [9]. Для зубців вибирається сталь 2013 (9, табл. П- 17), для ярма - також сталь 2013 (9, табл. П- 16). При оптимізації цей вибір робитиметься автоматично в програмі. У додатку Д для зручності також наведена таблиця залежності магнітної напруженості від величини індукції для сталі 2013.

Розрахункові поля km, Im, Imo - коефіцієнта насичення, струму намагнічування і його відносного значення є неактивними (*read only*).

Оформлення сторінки майстра за розрахунком струму намагнічування показане на рис. 5.6.

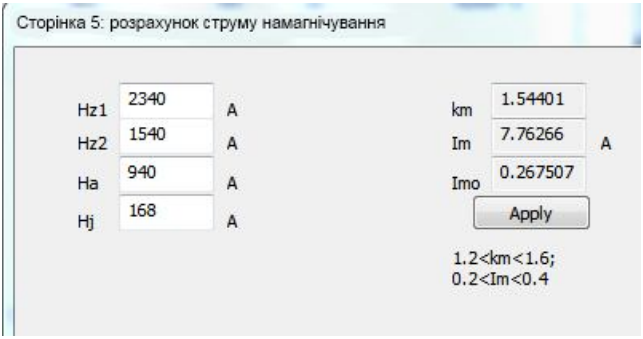


Рисунок 5.6 - Розрахунок струму намагнічування

Контрольними величинами поточної сторінки є значення коефіцієнта намагнічування і відносного значення струму намагнічування . Вони повинні знаходитися в межах, вказаних в написах пояснень на сторінці майстра.

Лістинг функції розрахунку приведений нижче.

```
//Сторінка 5: розрахунок струму намагнічування
void fpage5()
{
Bz1 = Bd*t1/(bz1*kc); //індукція в зубцях статора
Bz2 = Bd*t2/(bz2*kc); //індукція в зубцях ротора
Ba = Fi/(2*ha*ld*kc); //індукція в ярмі статора
hj1 = (2 + p)*(D2/2 - hp2)/(3.2*p);//спинка ротора (попереднє значення)
Bj = Fi/(2*hj1*ld*kc);//індукція в ярмі ротора
gamma = (bsh/delta)*(bsh/delta)/(5 + bsh/delta);
kd = t1/(t1 - gamma*delta);//коефіцієнт Картера
Fd = 1.59e06*Bd*kd*delta; //МРС повітряного проміжку
hz1 = hp;//висота зубця статора
Fz1 = 2*hz1*Hz1; //МРС в зубцях статора
hz2 = hp2 - 0.1*b2r; //висота зубця ротора
Fz2 = 2*hz2*Hz2; //МРС в зубцях ротора
kz = 1 + (Fz1 + Fz2)/Fd; //коэфф. насичення зубцевої зони
La = pi*(Da - ha)/dp; //середня магн. лінія ярма статора
```

```
hj = (D2 - Dj)/2 - hp2; //висота спинки ротора (остаточна)
Lj = pi*(Dj + hj)/dp; //середня магн. лінія ярма ротора
Fa = La*Ha; //МРС ярма статора
Fj = Lj*Hj; //МРС ярма ротора
Fc = Fd + Fz1 + Fz2 + Fa + Fj;//МРС на пару полюсів
km = Fc/Fd; //коэфф. насичення магнітного кола
Im = p*Fc/(0.9*m*W1*kob);//струм намагнічування
Imo = Im/I1n;//відносне значення струму намагнічування
}
```

5.1.6 Параметри робочого режиму. Розрахунок є одним з найбільш складних в синтаксичному відношенні. Слід бути уважним при складанні розрахункових формул коефіцієнтів розсіювання.

На рис. 5.7 показано вікно сторінки майстра до розрахунку параметрів робочого режиму. Також приведений лістинг функції розрахунку активних і індуктивних опорів обмоток статора і ротора.

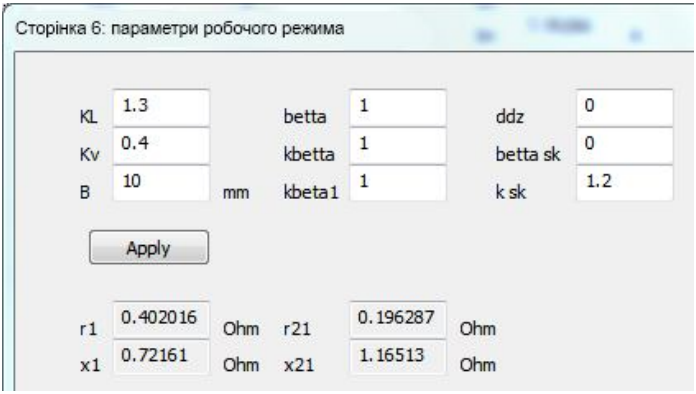


Рисунок 5.7 - Розрахунок параметрів робочого режиму

```
//Сторінка 6: параметри робочого режиму
void fpage6()
{
lp1 = ld;
bkt = pi*(D + hp)*betta1/dp;//котушка
ll1 = Kl*bkt + 2*B;//довжина лобової частини
lmid1 = 2*(lp1 + ll1);//середня довжина витка
L1 = lmid1*W1;//довжина обмотки
lv = Kv*bkt + B;//довжини вильоту лобової частини
//Активний опір статора
r1 = p115*L1/(qef*a);
r1o = r1*I1n/U; //відносний опір статора
//Активний опір ротора
rc = p115a*l2/qc2; //опір стержнів
rk = p115a*pi*Dk/(Z2*qk); //опір кілець
r2 = rc + 2*rk/(ddr*ddr);
r21 = r2*4*m*W1*W1*kob*kob/Z2;
r21o = r21*I1n/U;//відносний опір ротора
//Індуктивний опір статора
Lh1 = (b2 - bsh)/2;
//коэфф. пазового розсіяння
lamda_p = h1*kbetta/(3*b2) + kbetta1*(3*Lh1/(b2 + 2*bsh) + hsh/bsh);
//коэфф. лобового розсіяння
lamda_l = 0.34*q*(ll1 - 0.64*tau)/ld;
ksi = 2*ksk*kbetta - kob*kob*t2*t2(t1*t1);
//коэфф. диференціального розсіяння
lamda_d = t1*ksi/(12*delta*kd);
x1 = 15.8*f1*W1*W1*ld(100*100*100*p*q);
x1o = x1*I1n/U;
//Індуктивний опір ротора
Lh1r = hp2 - hsh2 - hshr1 - 0.1*b1r;
//коэфф. пазового розсіяння
lamda_p2 = Lh1r(3*b1r) + 0.66 - bsh2/(2*b1r) + hsh2/bsh2 +
1.12*hshr1*1e06/I2;
//коэфф. лобового розсіяння
lamda_l2 = 2.3*Dk*log(Z2*ld*ddr*ddr);
ksi2 = 1 + 0.2(pi*p/Z2) - ddz/(1 -(p/Z2));
//коэфф. диференціального розсіяння
lamda_d2 = t2*ksi2/(12*delta*kd);
x2 = 7.9*f1*ld*(lamda_p2 + lamda_l2 + lamda_d2)*1e-06;
x21 = x2*4*m*W1*W1*kob*kob/Z2;
x21o = x21*I1n/U;
}
```

5.1.7 Розрахунок втрат. Розрахунок не викликає труднощів. Приклад оформлення сторінки майстра показаний на рис. 5.8. Втрати в сталі виводяться у ватах.

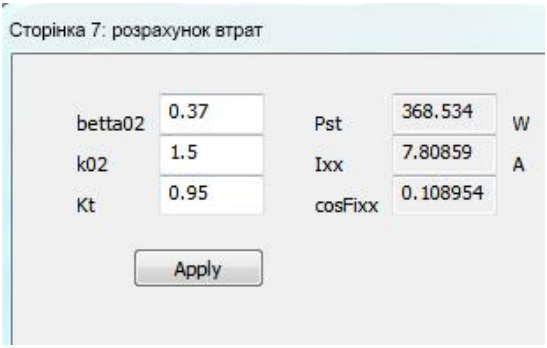


Рисунок 5.8 - Розрахунок втрат

Результатами розрахунку є визначення основних втрат, а також визначення струму холостого ходу і коефіцієнта потужності в режимі х.х.:

```
//Сторінка 7: розрахунок втрат
void fpage7()
{
ha = 0.5*(Da - D) - hp;
ma = pi*(Da - ha)*ha*ld*kc*gammaSt;
mz1 = hz1*bz1*Z1*ld*kc*gammaSt;
//втрати в сталі основні
Pst_main = 2.6*(1.6*Ba*Ba*ma + 1.8*Bz1*Bz1*mz1);
B02 = betta02*kd*Bd;
ppov2 = 0.5*k02*pow((Z1*n/10000)1.5)*B02*B02*t1*t1*1e06;
//втрати поверхневі
Ppov2 = ppov2*(t2 - bsh2)*Z2*ld;
Bpul2 = gamma*delta*Bz2/(2*t2);
mz2 = Z2*hz2*bz2*ld*kc*gammaSt;
//втрати пульсаційні
Ppul2 = 0.11*pow((Z1*n*Bpul2/1000)2)*mz2;
```

```
//втрати в сталі додаткові
Pst_dob = Ppov2 + Ppul2;
//повні втрати в сталі
Pst = Pst_main + Pst_dob;
//втрати механічні
Pmeh = Kt*n*n*Da*Da*Da*Da/100;
//втрати додаткові
Pdob = 0.005*Pn/kpd;
//втрати електричні в режимі х.х.
Pe10 = 3*Im*Im*r1;
//струм х.х.
Ixxa = (Pst + Pmeh + Pe10)/(m*U);
I0 = sqrt(Ixxa*Ixxa + Im*Im);
cosFi0 = Ixxa/Im;
}
```

5.1.8 Розрахунок робочих характеристик. Результатом розрахунку робочих характеристик є таблиця (9, табл. 6-26) для п'яти - шести значень ковзань в розрахунковому діапазоні. За результатами розрахунку необхідно знайти значення усіх величин, приведених в таблиці, для номінального ковзання. Знаходження номінальних даних здійснюється методом сплайн-інтерполяції за розрахованими даними. Будуються залежності робочих характеристик P_1, I_1, cosφ, ККД, ковзання s у функції від корисної потужності P_2 (9, рис. 6-44). Приклад сторінки майстра показаний на рис. 5.9.

Графік будується за допомогою графічної бібліотеки *Graph.h*. Реакція на елемент вибору типу *check box* (I1, s, kkd, cosFi) призводить до побудови відповідної графічної залежності.

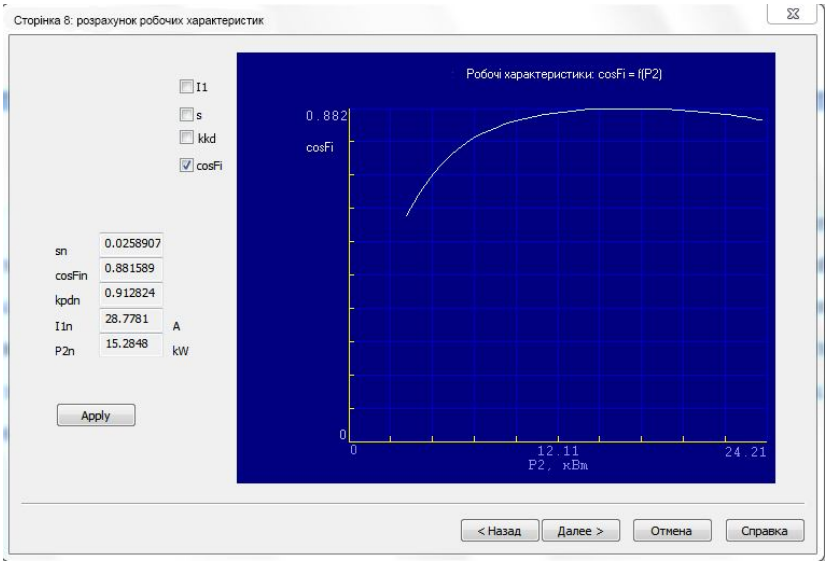


Рисунок 5.9 - Розрахунок робочих характеристик

```
//Сторінка 8: розрахунок робочих характеристик
void fpage8()
{
//відкриття файлів даних для запису
ofstream df_s("s.txt");//ковзання
ofstream df_P2("P 2.txt");//корисна потужність
ofstream df_kkd("kkd.txt");//к.п. д.
ofstream df_cos("cos.txt");//cosф
ofstream df_I1("I 1.txt");//струм статора I1
ofstream df_I21("I21.txt");//приведений струм ротора I21


//приведені параметри ротора
r12 = Pst_main/(m*Im*Im);
x12 = U/Im - x1;
c1 = 1 + x1/x12;
//активна складова струму х.х.
I0a(3*U);
```

```
//параметри схеми заміщення
aa1 = c1*c1;
bb1 = 0;
aa = c1*r1;
bb = c1*(x1 + c1*x21);
//постійні втрати (у сталі і механічні)
Pst_meh = Pst + Pmeh;
//номінальне ковзання
sn = r21o;
s = 0.2*sn;//початкове значення ковзання
ds = 0.001; //крок зміни ковзання
imax((2*sn - s)/ds);//кількість розрахункових точок
//відкриваємо цикл
for (int i = 0; i <= imax; i++)
{
	R = aa + aa1*r21/s;
	X = bb + bb1*r21/s;
	Z = sqrt(R*R + X*X);
	I211 = U/Z;
	I1a = I0a + I211*R/Z;
	I1r = Im + I211*X/Z;
	I1 = sqrt(I1a*I1a + I1r*I1r);
	I21 = c1*I211;
	P1r = 3*U*I1a;
	Pe1 = 3*I1*I1*r21;
	Pe2 = 3*I21*I21*r21;
	Pd = Pdob*(I1*I1/(I1n*I1n));
	Psum = Pst_meh + Pe1 + Pe2 + Pd;
	P2 = P1r - Psum;
	kkdr = 1 - Psum/P1r;
	cosFir = I1a/I1;
	//зберігаємо розраховані значення в масиви
	dimI1[i] = I1;//струм статора
	dimkkd[i] = kkdr;//к.п. д.
	dimcosFi[i] = cosFir;//cosФ
	dims[i] = s;//ковзання
	dimP2[i] = P2;//корисна потужність
	dimI21[i] = I21;//приведений струм ротора
	dimPe1[i] = Pe1;//втрати електричні статора
	dimPe2[i] = Pe2;//втрати електричні ротора
	dimPsum[i] = Psum;//втрати сумарні
```

```
        //запис розрахованих значень у файли даних
        df_s <<  s << "\n";
        df_kkd <<  kkdr << "\n";
        df_cos <<  cosFir << "\n";
        df_I1 <<  I1 << "\n";
        df_P2 <<  P2 << "\n";
        df_I21 << I21 << "\n";

        //збільшуємо ковзання
        s += ds;
}//кінець циклу

//Знаходження номінальних значень робочих характеристик
kkdn = spline(sn, dims, dimkkd);
cosFin = spline(sn, dims, dimcosFi);
I1nc = spline(sn, dims, dimI1);
P2n = spline(sn, dims, dimP2);
I21nc = spline(sn, dims, dimI21);
Pe1n = spline(sn, dims, dimPe1);
Pe2n = spline(sn, dims, dimPe2);
Psumn = spline(sn, dims, dimPsum);
Mn = p*m*I21nc*I21nc*r21*r21/(2*pi*f1*sn);
}
```

У лістингу функції розрахунку робочих характеристик усі розрахункові величини, залежні від ковзання, заносяться в масив і зберігаються у відповідних файлах даних. Після виходу з циклу по ковзанню, методом сплайн-інтерполяції знаходяться значення ККД, cosφ, I_1, P_2, I'_2, P_{e1}, P_{e2}, ΣP і $M_н$ при номінальному ковзанні.

5.1.9 Розрахунок пускових характеристик. Результатом розрахунку пускових характеристик є таблиця (9, табл. 6-28) для п'яти - шести значень ковзань в розрахунковому діапазоні. За результатами розрахунку знаходяться усі значення величин, приведених в таблиці, для ковзання s = 1. Розраховується значення критичного моменту.

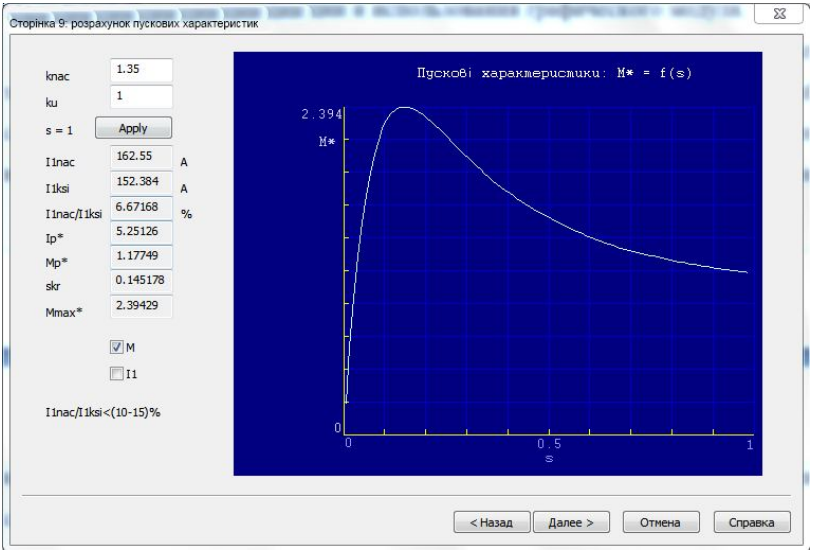


Рисунок 5.10 - Розрахунок пускових характеристик

Знаходження критичного моменту здійснюється шляхом пошуку максимуму в масиві розрахованих даних електромагнітного моменту. Після знаходження критичного моменту методом сплайн-інтерполяції визначається величина критичного ковзання. Виконується побудова залежності моменту M і струму I_1 у функції ковзання (9, рис. 6-67). Приклад сторінки майстра показаний на рис. 5.10. Графік будується за допомогою графічної бібліотеки *Graph.h*. Реакція на елемент вибору типу check box (M, I_1) призводить до побудови відповідної графічної залежності.

```
//Сторінка 9: розрахунок пускових характеристик
void fpage9()
{
//відкриття файлів даних
ofstream df_Mp("Mp.txt");//пусковий момент
ofstream df_I1p("I 1p.txt");//пусковий струм
ofstream df_s("sp.txt");//ковзання
hc = hp2 - hsh2 - hshr1;
ds = 0.01;//крок ковзання
s = sn*0.2;//початкове значення ковзання
imaxp((1 - s)/ds);//кількість розрахункових точок
//відкриття циклу
for (int i = 0; i <= imaxp+1; i++)
{
	//визначення параметрів з урахуванням витіснення струму
	ksip = 63.61*hc*sqrt(s);
	if (ksip < 1)
		fip = ksip/10;
	else if (ksip > 3)
		fip = ksip - 1;
	else
		fip = spline(ksip, "ksip.dat", "fip.dat");
	hr = hc/(1 + fip);
	br = b1r -(hr - 0.5*b1r)/h1r;
	qr = pi*b1r*b1r/8 + 0.5(hr - 0.5*b1r);
	kr = qc2/qr;
	Kr = 1 + rc*(kr - 1)/r2;
	//активний опір ротора з урахуванням витіснення струму
	r2ksi = Kr*r21;
	if (ksip > 4)
		kdp = 3*ksip/2;
	else
		kdp = spline(ksip, "ksipf.dat", "kdp.dat");
	//коэфф. пазового розсіяння з урахуванням витіснення струму
	lamda_p2ksi = (Lh1r*pow(3*b1r) + 0.66 - 0.5*bsh2/b1r)*kdp +
	hsh2/bsh2 + 1.12*hshr1*1e06/(6.5*I2);
	Kx = (lamda_p2ksi + lamda_l2 + lamda_d2)/(lamda_p2 +lamda_l2 +
	lamda_d2);
	//індуктивний опір ротора
	x21ksi = x21*Kx;
	//Врахування насичення
	//струм ротора без урахування насичення
	I21p = U/sqrt(pow((r1 + r2ksi/s)2) + pow((x1 + x21ksi)2));
```

```
CN = 0.64 + 2.5*sqrt(delta/(t1 + t2));
//струм статору з урахуванням насичення (попереднє знач.)
I1nac = knac*I21p;
//Розрахунок параметрів статора з урахуванням насичення
Fpmid = 0.7*I1nac*Up*(kbetta1 + ku*kob*Z1/Z2)/a;
Bfd = Fpmid*1e-06/(1.6*delta*CN);
xd = spline(Bfd, "Bfd.dat", "xd.dat");
c1p = (t1 - bsh)*(1 - xd);
h11p = hp - hsh - h1;
//коэфф. пазового розсіяння статора з урахуванням насичення
dlamda_p1 = (hsh + 0.58*h11p)*c1p/(bsh*(c1p + 1.5*bsh));
lamda_p1nac = lamda_p - dlamda_p1;
//коефф. дифф. розсіяння статора з урахуванням насичення
lamda_d1nac = lamda_d*xd;
//індуктивний опір статора з урахуванням насичення
x1nac = x1*(lamda_p1nac + lamda_l + lamda_d1nac)/(lamda_p +
lamda_l + lamda_d);
//Розрахунок параметрів ротора з урахуванням насичення
c2 = (t2 - bsh2)*(1 - xd);
dlamda_p2 = hsh2*c2/(bsh2*(bsh2 + c2));
//коефф. пазового розсіяння ротора з урахуванням насичення
lamda_p2nac = lamda_p2ksi - dlamda_p2;
//коефф. дифф. розсіяння ротора з урахуванням насичення
lamda_d2nac = lamda_d2*xd;
//індуктивний опір ротора з урахуванням насичення
x21nac = x21*(lamda_p2nac +lamda_l2 + lamda_d2nac)/(lamda_p2
+lamda_l2 + lamda_d2);
//Розрахунок пускового режиму
//опір взаємної індукції обмоток в пусковому режимі
x12p = x12*Fc/Fd;
c1pnac = 1 + x1nac/x12p;
app = r1 + c1pnac*r2ksi/s;
bpp = x1nac + c1pnac*x21nac;
//приведений струм ротора в пусковому режимі
I21p2 = U/sqrt(app*app + bpp*bpp);
//струм статора в пусковому режимі
I1p = I21p2*sqrt(app*app + pow((bpp + x12p), 2))/(c1pnac*x12p);
//Відносні значення
Ipo = I1p/I1n;//пусковий струм
Mpo = I21p2*I21p2*Kr*sn/(I21nc*I21nc*s);//пусковий момент

//Збереження розрахованих значень в масиви
dimMp[i] = Mpo;//момент
```

```
        dimI1p[i] = Ipo;//струм
        dimsp[i] = s;//ковзання

        //Запис значень у файли
        df_Mp << Mpo << "\n";//момент
        df_I1p << Ipo << "\n";//струм
        df_s << s << "\n";   //ковзання
        s += ds;
}//кінець циклу

//Визначення розрахункових величин в режимі к.з. (при s = 1) в отн. ед.
I1nac_s1 = I1nac;
I1ksi_s1 = I1p;
Ipo_s1 = Ipo;
Mpo_s1 = Mpo;
//Визначення розрахункових величин при критичному ковзанні
//Знаходимо індекс елементів при критичному ковзанні
//визначення по знаходженню індексу максимального моменту
int ikr = fmax(dimMp, imaxp+1);
Mmaxo = dimMp[ikr];//критичний момент
skr = dimsp[ikr];//критичне ковзання
persent = 100*(1 - I1nac_s1/I1ksi_s1);
if (persent < 0)
        persent *= -1;
}//кінець функції 9
```

5.1.10 Тепловий і вентиляційний розрахунки. Тепловий і вентиляційний розрахунки є важливими контрольними точками в розділі неоптимізованого проектування асинхронного двигуна (рис. 5.11).

Слід звернути увагу на середнє перевищення температури обмотки статора над температурою довкілля $\Delta\theta_1$ і витрати повітря $Q'_в$, яке забезпечується вентилятором.

Лістинг функції теплового і вентиляційного розрахунків:

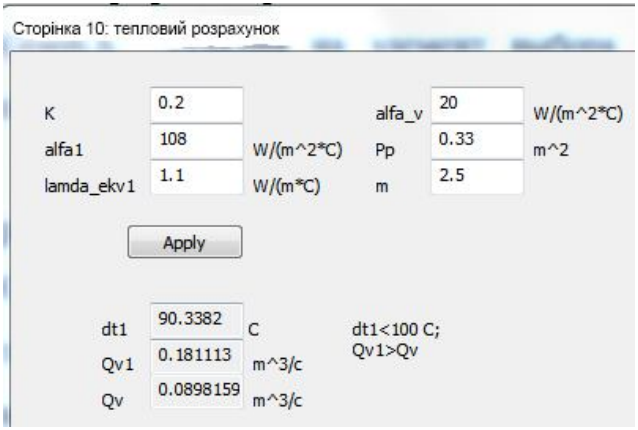


Рисунок 5.11 - Сторінка майстра : тепловий і вентиляційний розрахунки

```
//Сторінка 10: тепловий розрахунок
void fpage10()
{
//Тепловий розрахунок
Pe1n = m*I1nc*I1nc*r1;
Pe2n = m*I21nc*I21nc*r21;
Psumn = Pst_meh + Pe1n + Pe2n + Pdob;
Pep1 = kiz*Pe1n*2*ld/lmid1;
//перевищення темп. внутрішній поверхні сердечника статора
dtpov1 = K*(Pep1 + Pst_main)/(pi*D*ld*alfa1);
Pp1 = 2*hp + b1 + b2;
//перепад темп. в ізоляції пазової частини обмотки статора
dtizp = Pep1*(biz/lamda_ekv + (b1 + b2)/(16*lamda_ekv1))/(Z1*Pp1*ld);
Pelob1 = kiz*Pe1n*2*ll1/lmid1;
PPlob = Pp1;
bizlob = 0;
//перепад темп. по товщині ізоляції лобових частин
dtizlob=Pelob1*(bizlob/lamda_ekv+hp/(12*lamda_ekv1))/(2*Z1*PPlob*ll1);
```

```
//перевищення темп. зовнішньою поверхноси лобових частин
dtpovlob = K*Pelob1/(2*pi*D*lv*alfa1);
//середнє перевищення темп. обмотки статора усередині машини
dt11 = (dtpov1 + dtizp)*2*ld/lmid1 + (dtizlob + dtpovlob)*2*ll1/lmid1;
Psum1 = Psumn(Pe1n + Pe2n);
Psum1v = Psum1 -(Pep1 + Pst_main) - 0.9*Pmeh;
Sk = (pi*Da + 8*Pp)*(ld + 2*lv);
//перевищення темп. повітря усередині машини
dtv = Psum1v/(Sk*alfa_v);
//середнє перевищення темп. обмотки статора над темп. ОС
dt1 = dt11 + dtv;
        //Вентиляційний розрахунок
kmv = mm*sqrt(n*Da/100);
//Необхідна витрата повітря
Qv = kmv*Psum1v/(1100*dtv);
//Витрата повітря від зовнішнього вентилятора
Qv1 = 0.6*Da*Da*Da*n/100;
}
```

5.1.11 Механічний розрахунок валу. На сторінці механічного розрахунку валу (рис. 5.12) виконуються розрахунки валу на жорсткість, міцність і визначається критична частота обертання.

Правильно сконструйований вал повинен задовольняти вимогам по допустимих значеннях механічної напруги в перерізу східців для прийнятої марки стали 45, тобто не перевищувати 3600 МПа.

Після розрахунків валу на жорсткість і міцність встановлюються остаточні розміри, які мають бути вибрані відповідно до нормованих діаметрів циліндричних кінців валу [9].

Отримані в результаті розрахунку геометричні розміри валу увійдуть пізніше до функції автоматичного креслення, яка реалізується на мові AutoLisp з команд програми, написаної на C++. В лістингу нижче наводиться код функції механічного розрахунку валу з додатковим блоком підготовки до оптимізації і виведенням отриманих результатів неоптимізованого розрахунку в текстовий файл.

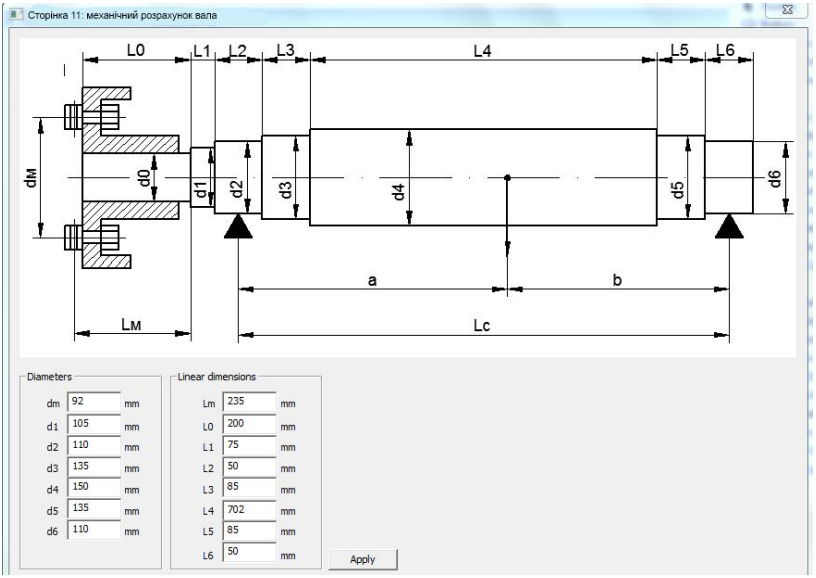


Рисунок 5.12 - Остання сторінка майстра : механічний розрахунок валу

```
void mehanical()
{
//Приведена сила тяжіння
mr = 6500*D2*D2*ld;
Gp = 9.81*mr;
//Прогин валу під серединою магнітопровода
Fp = 0.3*2*Mn/Dm;
//Прогин валу посередині сердечника
y1 = L2/2;
y2 = y1 + L3;
L41 = 0.55*L4;
L42 = L4 - L41;
y3 = y2 + L41;
x1 = L6/2;
x2 = x1 + L5;
x3 = x2 + L42;
```

```
J1 = pi*pow(d2, 4)/64;
J2 = pi*pow(d3, 4)/64;
J3 = pi*pow(d4, 4)/64;
J5 = pi*pow(d6, 4)/64;
J6 = pi*pow(d5, 4)/64;
J7 = pi*pow(d4, 4)/64;
Sa = pow(x1, 3)/J5+(pow(x2, 3) - pow(x1, 3))/J6+(pow(x3, 3) - pow(x2,
3))/J7;
Sb = pow(y1, 3)/J1+(pow(y2, 3) - pow(y1, 3))/J2+(pow(y3, 3) - pow(y2,
3))/J3;
a = x3;
b = y3;
Lc = y3 + x3;
E = 2.06e11;
fg = Gp(3*E*Lc*Lc);
//Прогин валу посередині магнітопровода ротора
c = L0/2 + L1 + L2/2;
S0 = y1*y1/J1 + (y2*y2 - y1*y1)/J2 + (y3*y3 - y2*y2)/J3;
fp = Fp*c(3*E*Lc*Lc);
//Первинне зміщення ротора
e0 = 0.1*delta + fg + fp;
//Початкова сила одностороннього магнітного тяжіння
T0 = 2.94*D2*ld*e0*1e05/delta;
//Прогин валу від сили T0
ft = fg*T0/Gp;
//Стале тяжіння
m = ft/e0;
fm = ft/(1 - m);
//Результуючий прогин
f = fm + fg + fp;
//Критична частота обертання
nk = 30*sqrt((1 - m)/fg);
//Розрахунок напруги в східцях валу
//Переріз А
//Момент, що вигинає
L = Lm
Mia = 2*Fp*L;
//Момент опору при вигині
Wa = 0.1*pow(d0, 3);
//Напруга в перерізі А
alfa = 0.6; //для нереверсивних машин
sigma_a = sqrt(Mia*Mia + 4*alfa*alfa*Mn*Mn)/Wa;
```

```
//Переріз В
//Момент, що вигинає
L = Lm + L1
Mib = 2*Fp*L;
//Момент опору при вигині
Wb = 0.1*pow(d1, 3);
//Напруга в перерізі В
sigma_b = sqrt(Mib*Mib + 4*alfa*alfa*Mn*Mn)/Wb;

//Переріз С
//Момент, що вигинає
L = Lm + L1 + L2/2;
T = T0/(1 - m);
Mic = 2*Fp*L(Gp + T)*y1/Lc;
//Момент опору при вигині
Wc = 0.1*pow(d2, 3);
//Напруга в перерізі С
sigma_c = sqrt(Mic*Mic + 4*alfa*alfa*Mn*Mn)/Wc;

//Переріз Д
//Момент, що вигинає
L = Lm + L1 + L2/2;
Mid = 2*Fp*L(Gp + T)*y2/Lc;
//Момент опору при вигині
Wd = 0.1*pow(d3, 3);
//Напруга в перерізі С
sigma_d = sqrt(Mid*Mid + 4*alfa*alfa*Mn*Mn)/Wd;

//Переріз Г
//Момент, що вигинає
L = Lm + L1 + L2/2;
Mig = (2*Fp*L + (Gp + T)*b)*x1/Lc;
//Момент опору при вигині
Wg = 0.1*pow(d6, 3);
//Напруга в перерізі С
sigma_g = sqrt(Mig*Mig + 4*alfa*alfa*Mn*Mn)/Wg;

//Переріз Ж
//Момент, що вигинає
L = Lm + L1 + L2/2;
Miz = (2*Fp*L + (Gp + T)*b)*x2/Lc;
//Момент опору при вигині
Wz = 0.1*pow(d5, 3);
```

```
//Напруга в перерізі C
sigma_z = sqrt(Miz*Miz + 4*alfa*alfa*Mn*Mn)/Wz;

//Визначення максимальної напруги
sigma_max = sigma_a;
sigma_max = (sigma_b > sigma_max)?sigma_b: sigma_max;
sigma_max = (sigma_c > sigma_max)?sigma_c: sigma_max;
sigma_max = (sigma_d > sigma_max)?sigma_d: sigma_max;
sigma_max = (sigma_g > sigma_max)?sigma_g: sigma_max;
sigma_max = (sigma_z > sigma_max)?sigma_z: sigma_max;

//Установка прапора механічного розрахунку
sigma_dop = 0.7*3600e05;
flag_meh = true; //механічний розрахунок задовільний
if (sigma_max >= sigma_dop)
        flag_meh = false;//механічний розрахунок незадовільний

//Виведення даних на друк у файл
DataOutput(BEGIN, "Data_first.txt");//файл даних неоптимиз. машини
//Створюємо копію номінальних значень базової машини
Bz1n = Bz1;
Bz2n = Bz2;
Ban = Ba;
Bjn = Bj;
J2n = J2;
I1n = I1nc;
Psum = Psumn;
P2 = P2n;
cosFir = cosFin;
//Підготовка даних до оптимізації
//Створюємо копію варійованих змінних базової машини
delta_fix = delta;
ld_fix = ld;
Up_fix = Up;
Z1_fix = Z1;

//розрахунок вартості базової машини
price();
SZ1 = SZ;//приведена вартість базової машини

}//Кінець функції механічного розрахунку валу
```

У функції механічного розрахунку також розміщений, як завершальний, код підготовки базової машини до оптимізації. У ньому виконується створення копій основних розрахункових величин базової машини, які підлягають порівнянню поточними розрахунками у циклі оптимізації. Якщо цього не зробити, то нові розраховані значення змінять вміст змінних, в яких зберігалися раніше розраховані величини.

Крім того, визначається вартість базової машини функцією *price()*, що детально розглянута в наступному пункті.

Також в лістингу присутній виклик функції *DataOutput()* для збереження усіх розрахункових величин в текстовому файлі. По першому ключовому параметру (у розрахунку базової машини це змінна типу *enum* - BEGIN) визначається поточна стадія розрахунку. Від ключового параметра залежить назва першого рядка текстового файлу.

Оскільки лістинг функції *DataOutput* великий, він винесений у додаток А.

При натисненні на кнопку "Готово" майстра, відбувається повернення до головного програмного модуля. При цьому стає активною кнопка оптимізації (рис. 5.13).

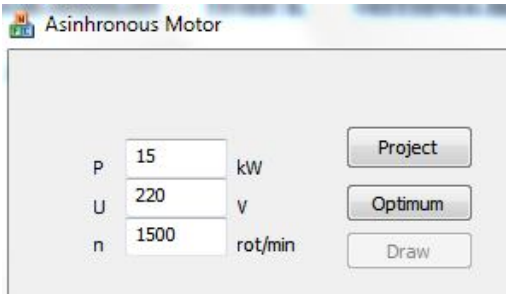


Рисунок 5.13 - Вікно головної програми після базового розрахунку

5.1.12 Розрахунок приведеної вартості електродвигуна. При порівнянні варіантів двигуна і виборі оптимального варіанту широко користуються поняттям "приведена вартість" - інтегральним показником, що враховує одночасно витрати на виробництво і експлуатацію двигуна:

$$З_{сум} = К_н Ц + С_э T\left(P_а + P_р\right),$$

де $К_н = 0,34$;

Ц - собівартість;

$С_э = 0,25$ грн./кВтч, вартість електроенергії;

$T = 1800$ ч, количесвто годин роботи в рік;

$P_а$ - сумарні активні втрати $P_{сум}$;

$P_р$ - реактивні втрати.

Реактивні втрати, приведені до активних, можна визначити як

$$P_р = k_{пп} \frac{P_{2н}}{\eta} \sin\varphi,$$

де $k_п = 0,12$ - економічний еквівалент реактивної енергії або коефіцієнт підвищення втрат.

Собівартість визначається сумарною вартістю матеріалів і витратами на виготовлення виробу :

$$Ц = k_{з.с.}\left[К_{ц1}\sum m_i G_i + \left(1 + К_{ц2}\right)\sum З_i G_i\right],$$

де $К_{ц1} = 1,29$;

m_i - вартість 1 кг матеріалу;

G_i - витрата матеріалів;

$К_{ц2} = 3,03$;

$З_i$ - витрати на обробку 1 кг матеріалу;

$k_{з.с.} = 1,05$.

Витрата матеріалів асинхронного двигуна з короткозамкненим ротором складається з витрат:

- електротехнічній сталі;
- конструкційній сталі;
- чавуну;
- міді;
- алюмінію;
- ізоляції.

Вартість одиниці матеріалу m_i і витрат на виготовлення 1 кг матеріалу приведена в табл. 5.1. Лістинг функції розрахунку приведеної вартості електродвигуна приведений в лістингу нижче. Функція складена так, що вона може викликатися як для розрахунку базової машини, так і в циклах оптимізації.

Таблиця 5.1 - Приблизна вартість матеріалів і витрат на їх виготовлення

Найменування	**m_i, грн./кг**	**$З_i$, грн./кг**
Сталь електротехнічна	21,0	1,55
Сталь конструкційна	4,65	6,75
Чавун	35	18,0
Мідь	26,45	6,3
Алюміній	11,8	0,55
Ізоляція	200,0	8,0

```
//Оптимізація приведеної вартості
void price()
{
Gcu = gammaCu*qef*lmid1*W1*m*1.04;
Gal = gammaAl*(Z2*qc2*ld*kc + 2*ak*bk*Dk)*1.2;
Gst = gammaSt*(Da*Da*ld*kc)*1.3;
Gfe = 1.16*Gst;
Gch = 0.74*Gst;
Giz = gammaIz*Siz*Z1*ld*1.4*1.85;
C = kzc*(Kc1(zcu*Gcu + zal*Gal + zst*Gst + zfe*Gfe + zch*Gch +
ziz*Giz));

SZ = Kn*C + CE*T*(Psum + 0.12*P2*sqrt(1 - cosFir*cosFir))/1000;
}
```

5.2 Автоматизований пошук оптимальних геометричних розмірів і параметрів асинхронного двигуна

Асинхронний двигун як об'єкт проектування характеризується сукупністю проектних даних, що визначають його конструктивну схему, геометричні розміри, обмотувальні дані і техніко-економічні показники. Усі ці дані повинні задовольняти великої кількості внутрішніх взаємних зв'язків у вигляді аналітичних співвідношень (рівнянь проектування), що визначаються за фізичними закономірностями, особливостями конструктивного виконання і технологічними чинниками.

Оскільки кожен двигун є елементом складної електромеханічної системи, то рівняння проектування повинні враховувати також зовнішні чинники: особливості схеми електропостачання, вимоги забезпечення експлуатаційної надійності в нормальних і аварійних режимах, характер механічного навантаження та ін. Для серійних двигунів ці чинники визначаються вимогами ДСТУ і враховуються у формі обмежень типу рівності або нерівностей.

Загальною особливістю рівнянь проектування є їх невизначеність, обумовлена невідповідністю числа рівнянь числу проектних даних. У зв'язку з цим, можна отримати велику кількість прийнятних варіантів двигуна, що задовольняли рівнянням проектування. Пошук одного з таких варіантів здійснюється ітераційним методом, суть якого полягає в наступному.

З усього різноманіття проектних даних виділяють деякі сукупності незалежних змінних, для яких задаються вхідні значення за даними аналогу або за допомогою наближених аналітичних співвідношень, отриманих на основі обробки статистичних даних раніше виконаних машин. Підставляючи вхідні значення незалежних змінних в рівняння проектування, визначають техніко-економічні показники двигуна і міру виконання всіх обмежень.

Якщо отримане рішення виявляється прийнятним, то виконується корекція незалежних змінних. Корекція може бути виконана на основі неформальних правил або за допомогою формалізованих процедур. Формалізовані процедури припускають різні способи перебору незалежних змінних і, як правило, вимагають великих витрат часу на пошук прийнятного варіанту. При використанні неформальних правил враховуються апріорні знання про характер впливу незалежних змінних на вихідні показники двигуна, що робить процес пошуку прийнятного варіанту проектних даних двигуна більш цілеспрямованим і, отже, менш трудомістким.

Безліч прийнятних варіантів двигуна утворюють область допустимих рішень. Для вибору кращого варіанту з цієї множини використовують критерії якості проектування. Завдання вибору в цьому випадку називається завданням оптимізації за певним критерієм. Якщо

критеріїв декілька, то для отримання рішення виробляють їх згортання, або здійснюють декомпозицію загального завдання багатокритеріальної оптимізації на декілька однокритеріальних завдань.

5.2.1 Математична модель асинхронного двигуна. Основу математичної моделі асинхронного двигуна складає учбова методика розрахунку [9]. Ця методика дозволяє задовольнити суперечливим вимогам, що пред'являються до моделі відносно точності і простоти, і в той же час забезпечує можливість безпосереднього використання результатів розрахунку на наступних етапах конструкторсько-технологічного проектування двигуна.

Проте наявність в учбовій методиці неформальних правил вибору ряду конструктивних параметрів обмежує її можливості при проведенні проектних досліджень і пошуку оптимального варіанту. Тому доцільно модель розширити, доповнивши її формалізованими процедурами вибору деяких конструктивних параметрів.

Зокрема, в модель включена оптимізаційна процедура вибору величини повітряного проміжку δ і довжина пакету статора $l_1 = l_\delta k_c$: Як критерій оптимізації приймається цільова функція з таблиці завдань на проект.

Завдання оптимізації параметрів δ, l_δ вирішується з урахуванням обмежень (табл. 5.2). Разом з функціональними обмеженнями в цьому завданні враховуються також параметричні обмеження, що визначають практично реалізацію зони пошуку параметрів оптимізації.

У табл. 5.2 обмежень мінімальних і максимальних значень контрольованих параметрів, виділених сірим кольором, вибираються по малюнках і таблицях, приведених в [9].

Таблиця 5.2 – Діапазон варіювання параметрів та обмеження

№ п/п	Од. вим.	MIN	Параметр	MAX	Обмеження
1	-		$I_п/I_н$		1, табл. 6-27
2	-		$M_п/M_н$		1, табл. 6-27
3	^oC	40	Δt_1	110	
4	$А/мм^2$	4	J_1	7	
5	$А/мм^2$	2,5	J_2	3,5	
6	Тл		B_δ		9, рис. 6-11, дисперсія 5 %
7	Тл	1,7	B_{z1}	1,9	
8	Тл	1,75	B_{z2}	1,85	
9	Тл	1,4	B_a	1,6	
10	Тл	0,8	B_j	1,25	
11	-		λ		9, рис. 6-14, а, дисперсія 5 %
12	-		Z_1		З t_1, 9, табл. 6-9, дисперсія 20 %, кратне 3, $\Delta Z_1 = 3$
13	-		Z_2		9, табл. 6-15, дисперсія 20 %, $\Delta Z_2 = 3$
14	мм	0,4	δ	1,2	$\Delta\delta = 0,1$ мм
15	-	0,68	$k_{з.п.}$	0,72	

Для урахування обмежень (табл. 5.2) використовується метод зовнішніх штрафних функцій, розглянутий раніше в розділі 4.9.

Вибір коефіцієнтів штрафів визначається умовами завдання. При виконанні проекту коефіцієнти штрафів приймаються постійними. Рекомендується прийняти величину коефіцієнта штрафу К = 200.

Завдання оптимізації вирішується методом покоординатного спуску, розглянутого в розділі 4.7. Цей метод вибраний у зв'язку з його простотою і ефективністю в завданнях оптимізації з досить обмеженою зоною пошуку. Пошук оптимальних параметрів здійснюється шляхом руху точки уздовж координатних осей n -мірного простору параметрів оптимізації.

На кожному кроці змінюється лише один параметр, а інші залишаються незмінними. Рух уздовж вибраної координати виконується до тих пір, поки цільова функція Fn зменшується, потім вибирається наступний координатний напрям. Метод простий в реалізації, але схильний до зациклення в ситуаціях "ярів". Якщо вибраний варіант не потрапляє в допустиму область, то здійснюється корекція незалежних змінних моделі за допомогою неформальних правил або на основі використання формалізованих процедур.

Програмна реалізація першого етапу оптимізації вирішується об'єднанням в одну нову функцію усіх одинадцяти підпрограм розрахунку, створених раніше на етапі неоптимізованого проектування. Нова підпрограма модифікується у відповідність з впливом варійованих параметрів на параметри двигуна. Лістинг підпрограми автоматичного розрахунку асинхронного двигуна, зважаючи на великий розмір, приведений в дод. Б.

Усі залежності магнітної напруженості від магнітної індукції реалізуються у вигляді функції сплайн-інтерполяції.

Робочі характеристики не розраховуються, а визначаються тільки втрати та ККД при номінальному ковзанні. Пускові характеристики розраховуються повністю з визначенням величин критичного моменту і критичного ковзання.

Виклик функції автоматизованого розрахунку (дод. Б) викликається в циклі реалізації методу покоординатного спуску (див. лістинг в розділі 4.7).

Приклад оформлення діалогового вікна програми показаний на рис. 5.14.

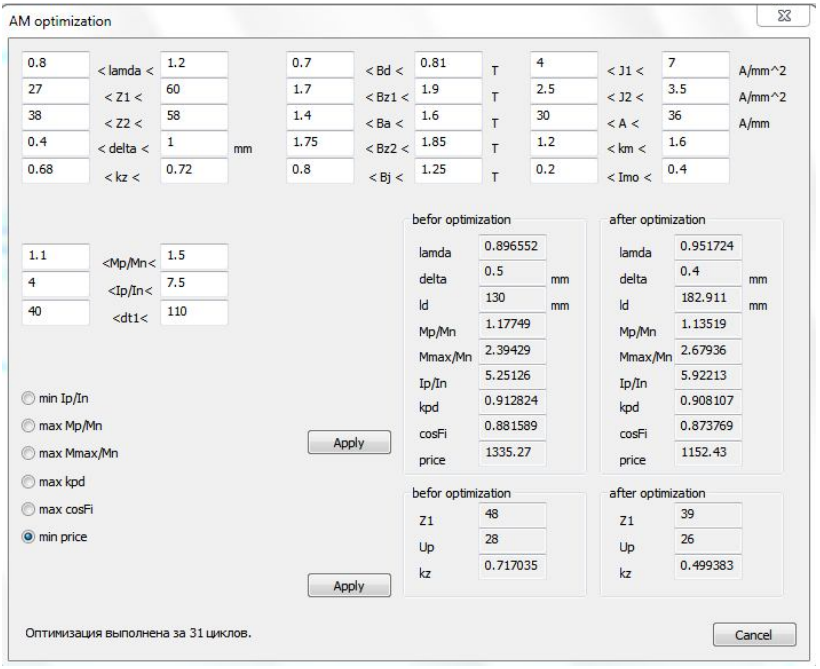


Рисунок 5.14 - Оптимальний розрахунок асинхронного двигуна

Як критерій оптимальності у вікні програми (рис. 5.14) можуть бути прийняті:

- мінімальна кратність пускового струму;
- максимальна кратність пускового струму;
- максимальна перевантажувальна здатність;

- максимальний ККД;

- максимальний коефіцієнт потужності;

- мінімальна приведена вартість.

У функції автоматичного розрахунку в змінну *opt* в секції вибору *swtch* записуватиметься поточне значення параметра, відповідного прийнятому критерію оптимальності у вікні програми. Далі це значення використовуватиметься функцією, що реалізовує метод зовнішніх штрафних функцій і у методі покоординатного спуску.

5.2.2 Автоматизований пошук оптимального числа зубців статора і числа пазів на полюс і фазу. У моделі асинхронного двигуна, що реалізовується, як варійовані параметри приймаються, в допустимому діапазоні, кількість зубців статора Z_1 і кількість пазів на полюс і фазу $U_п$. Міру впливу цих параметрів на вихідні показники двигуна можна оцінити за допомогою якісного аналізу багатопов'язаної системи нелінійних рівнянь, що лежать в основі інженерної методики проектування.

Збільшення $U_п$ і Z_1 веде до зростання індуктивного опору обмотки статору, що обумовлює зниження пускового струму, але при цьому одночасно зменшуються пусковий і максимальний моменти. Негативним результатом збільшення $U_п$ і Z_1 є також зростання втрат в обмотці статора і отже, підвищення її перегріву. Таким чином, при порушенні обмежень на кратності моментів $M_п/M_н$, $M_{макс}/M_н$ і на перегрівання обмотки статора $\Delta\theta_1$, необхідно знижувати $U_п$ і Z_1, а при порушенні обмеження на кратність пускового струму $I_п/I_н$ ці параметри слід збільшувати. Крок зміни $U_п$ для одношарових обмоток статора приймається рівним 1, а крок зміни Z_1 - рівним 3, за умови, що Z_{1min} є кратним трьом. Підпрограма розрахунку електродвигуна складається аналогіч-

но п. 5.2.1. Завдання оптимізації вирішується методом зовнішніх штрафних функцій із застосуванням методу покоординатного спуску. Лістинг функції автоматичного розрахунку при пошуку оптимального числа зубців статора і числа пазів на полюс і фазу приведений в дод. В.

5.2.3 Вибір оптимального варіанта. На ранніх стадіях проектування показник приведеної вартості $З_{сум}$ визначається з великою погрішністю. В межах цієї погрішності існує безліч варіантів двигуна, що помітно відрізняються по ряду важливих показників, яки характеризують якість проектування.

До таких показників передусім належать:

- питомі втрати: $F_1 = 1 - ККД$

- питомі витрати міді : $F_2 = \frac{\text{Маса міді}}{\text{повна потужність}}$;

- питомі витрати сталі: $F_3 = \frac{\text{Маса сталі}}{\text{повна потужність}}$.

При виборі оптимального варіанту асинхронного двигуна разом з показником приведеної вартості доцільно використовувати приватні критерії. Проте через їх суперечність не існує варіанту двигуна, який був би оптимальним відразу по усіх критеріях. Можна виділити лише варіанти, які не поступаються іншим по якому-небудь одному або декільком показникам. Такі варіанти дістали назву ефективних варіантів.

В результаті проектування можливо мінімум три ефективні варіанти, отриманих при:

1) знаходженні екстремуму цільової функції першого етапу оптимізації при варіюванні l_δ, δ;

2) знаходженні екстремуму цільової функції другого етапу оптимізації при варіюванні Z_1, $U_п$;

3) мінімізації приведених витрат $З_i$.

Для кожного з варіантів розраховуються показники F_1, F_2, F_3 і заносяться в таблицю (табл. 5.3).

Таблиця 5.3 - Розрахункові показники оптимальності

№ варіанта	**F_1**	**F_2**	**F_3**
1			
2			
3			

Алгоритм виділення ефективних варіантів заснований на правилі переваги Парето. Згідно з цим правилом, з безлічі прийнятних варіантів виділяють варіант Ко, починаючи з Ко= 1, і для усіх j критеріїв перевіряють умову

$$F_{kj} < F_{koj}, \qquad k = 1,3; \quad j = 1, 3.$$

Варіанти, що не задовольняють цій умові, відкидаються як свідомо "погані", оскільки поступаються іншим по усіх критеріях. З інших варіантів вибирається новий варіант, якому надається індекс Ко, і знову перевіряється ця умова. Процес повторюється до тих пір, поки не залишиться жодного варіанту, якому б не надавався індекс Ко.

Варіанти, що залишилися, і складатимуть сукупність ефективних варіантів.

Побудова безлічі ефективних варіантів дозволяє значно звузити зону пошуку, але проблема вибору оптимального варіанту все одно залишається. При невеликому числі ефективних варіантів вибір кращого з них здійснюється на основі ретельного аналізу кожного варіанту з урахуванням вимог технологічності, уніфікації, стандартизації і інших чинників, не врахованих в моделі.

Якщо кількість ефективних варіантів велике, то часто використовують згортання критеріїв. Розглянемо один із способів згортання.

Нехай F_j^* - рекордне значення j - критерію серед ефективних варіантів, а F_{kj} - значення j - критерію в k варіанті.

Тоді величина

$$W_{kj} = \frac{F_{kj} - F^*_j}{F^*_j}$$

визначає наскільки конкретний варіант гірше рекордного по j показнику.

Позначимо величину W_{kj} для самого гіршого варіанту через W^*_j і виконаємо нормування

$$\hat{W}_{kj} = W_{kj} / W^*_j .$$

Якщо ввести вагові коефіцієнти ξ_j для кожного критерію, то можна сформувати узагальнений критерій

$$F_\xi = \frac{1}{3}\sum_{j=1}^{3} \xi_j \cdot \hat{W}_{kj} \to \min.$$

Варіант, що має найменше значення F_ξ, є найбільш близьким до рекордних і, отже, є кращим (оптимальним) при заданій вектором ξ

відносній значущості критеріїв. Міняючи елементи вектора ξ відповідно до тих або інших переваг, можна отримувати різні кращі варіанти.

У учбовому проектуванні виконується формальне порівняння трьох варіантів за допомогою узагальненого критерію F_ξ, вважаючи однаковою значущість усіх приватних критеріїв ($\xi_1 = \xi_2 = \xi_3 = 1$). Результати розрахунків заносяться в таблицю (табл. 5.4).

По таблиці вибирається варіант, що має найбільш рекордні показники по одному або двох параметрах W_i. Вибраний варіант вважається оптимальним.

Таблиця 5.4 - Визначення оптимального варіанту

№ варіанта	**W1**	**W2**	**W3**	**F_ξ**
1				
2				
3				

5.3 Параметричне проектування електродвигуна

Після завершення останнього етапу оптимізації, виконується параметричне проектування оптимального електродвигуна, тобто автоматична генерація креслярської документації. Реалізація завдання виконується на мові AutoLisp в середовищі AutoCad.

Програма скрипт-коду на мові AutoLisp, призначена для запуску в середовищі AutoCad, повинна генеруватися автоматично при натисненні на кнопку "Drawing" головного вікна програми (див. рис. 5.1).

При цьому викликається підпрограма, яка розраховує координати для побудови креслення по наявних після останнього етапу оптимізації геометричних розмірах електродвигуна.

Результати підпрограми виводяться у файл з розширенням LSP. Такий файл, завантажений в AutoCad, приведе до побудови креслення із створенням шарів і виставленням розмірів і написів.

Звичайно ж, в учбовому проектуванні не ставиться завдання генерації повного креслення асинхронного двигуна. Проте прості завдання можна легко реалізувати тими засобами, які були розглянуті в навчальному посібнику.

Розглянемо створення програмного коду для побудови окремих вузлів електродвигуна.

5.3.1 Створення шарів. Шари є невід'ємною частиною будь-якого креслення і, як правило, відкриваються за умовчанням у відповідність із заздалегідь створеним шаблоном.

Немає необхідності програмувати створення шарів - досить перед завантаженням генерованих креслень завантажити файл шаблону формату dwt або запустити скрипт, описаний в п. 3.2.1.

5.3.2 Побудова листа ротора. Спрощене креслення листа ротору, для якого необхідно написати програму побудови, показане на рис. 5.15.

Перед написанням програми необхідно відлагодити код AutoLisp, який пізніше потрібно буде формувати в C++. Код побудови листа ротора на мові AutoLisp, приведений нижче.

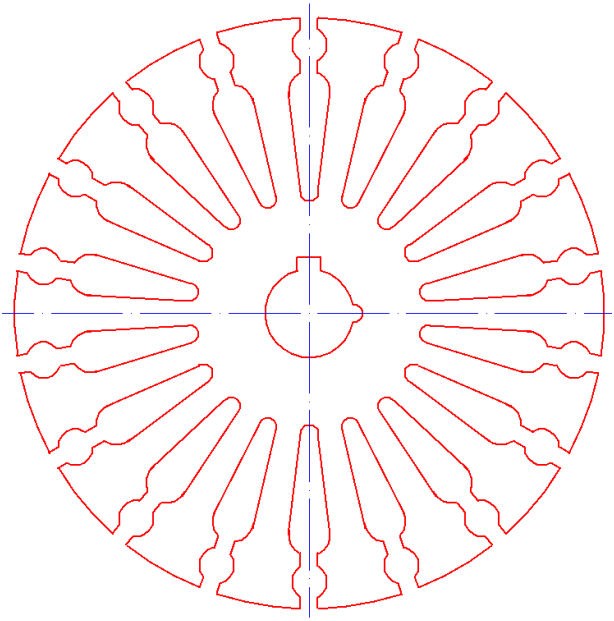


Рисунок 5.15 - Лист ротора

```
;Коло ротора
(command "_LAYER" "Make" "solid 1" "")
(command "_CIRCLE" "250, 250" "100")
;Паз
(command "_CIRCLE" "250, 341" "6")
(command "_CIRCLE" "250, 325" "7")
(command "_CIRCLE" "250, 291" "3")
;
;похилі лінії
(command "_LINE" "243, 325" "247, 291" "")
(command "_LINE" "257, 325" "253, 291" "")
;
;шліц
(command "_LINE" "247, 350" "247, 325" "")
(command "_LINE" "253, 350" "253, 325" "")
```

```
(command "_TRIM" "_CROSSING" "0, 0" "400, 400" "" "247, 350" "253,
350" "247, 340" "253, 340" "247, 330" "253, 330" "250, 347" "250, 335"
"250, 332" "250, 318" "250, 294" "250, 350" "")
(setq ang(angtof "20"))
(setq angx(sin ang))
(setq angy(cos ang))
(setq x(* 100 angx))
(setq y(* 100 angy))
(setq p(list(+ 250 y)))
(command "_TRIM" "_CROSSING" "0, 0" "360, 360" "" "350, 250" p "")
(command "_ERASE" "_CROSSING" "0, 0" p "")
(command "_ARRAY" "_WINDOW" "150, 250" "350, 360" "" "_POLAR" "250,
250" "18" "360" "Y")
;отвір під вал
(command "_CIRCLE" "250, 250" "15")
(command "_LINE" "246, 249" "246, 269" "254, 269" "254, 249" "")
(command "_TRIM" "_CROSSING" "235, 250" "265, 270" "" "246, 249" "254,
249" "250, 265" "")
(command "_CIRCLE" "265, 250" "3")
(command "_TRIM" "_CROSSING" "260, 245" "270, 260" "" "265, 250" "262,
250" "")
;осьові лінії
(command "_LAYER" "Make" "longdashdot" "")
(command "_LINE" "250, 145" "250, 355" "")
(command "_LINE" "145, 250" "355, 250" "")
;Показати усе
(command "_ZOOM" "_E")
```

Створення коду на C++ виконати так, як описано в п. 3.3.3.

5.3.3 Побудова секції вала. В учбових цілях виконуватиметься побудова не промислового вала електродвигуна, а декількох секцій, з такими елементами як фаски, округлення, виставлення розмірів, написів.

Побудову валу почнемо з розгляду проекту в Visual Studio. У редакторові ресурсів треба створити діалогове застосування, як показано на рис. 5.16. На формі діалогу уперше поєднуються текстові поля для введення даних і растрове зображення.

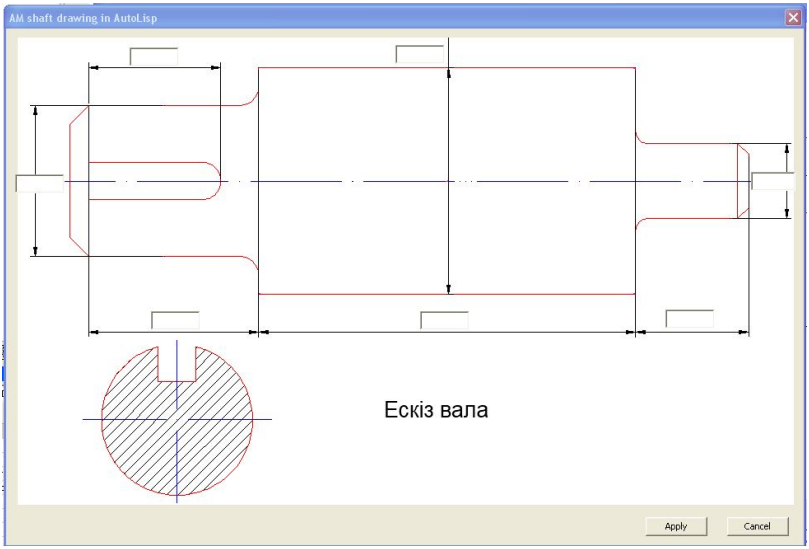


Рисунок 5.16 - Діалогове вікно програми параметричного креслення

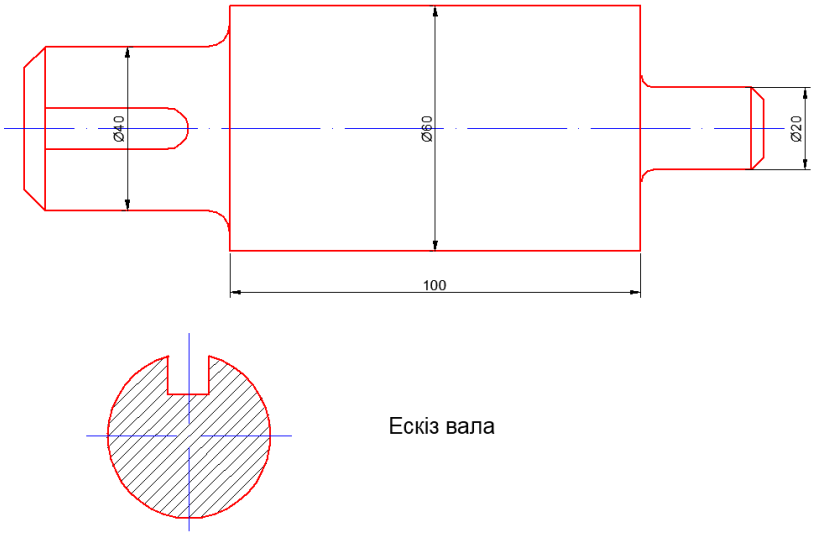
Дані мають бути типу *float*. Рисунок 24bit формату "bmp". Вікна введення розмірів валу розташувати на рівні розмірних ліній. Дані у вікнах, розташованих над розмірними лініями, можна ініціалізувати автоматично, після завершення проектування основної частини програми, або, за бажанням, змінити їх.

При натисненні на кнопку "Apply" дані з вікна мають бути передані в розрахункову програму, розміщену в зовнішньому "cpp" файлі.

Висоту фаски прийняти рівною 5 мм, кут фаски 45^{o}; радіус округлення 3 мм; глибину і ширину паза шпони прийняти 10 мм. Розміри задавати з таким розрахунком, щоб розмістити ескіз на листі формату А4 з масштабом 1:1.

У програмі виробляється розрахунок координат точок побудови валу, формування коду побудови валу на мові AutoLisp з наступним виведенням отриманого коду в текстовий файл з розширенням "LSP".

При завантаженні отриманого Лісп-файлу в AutoCad, має з'явитись креслення, як показано на рис. 5.17.



Ескіз вала

Рисунок 5.17 - Ескіз вала, отриманого методом параметричного проектування

Лістинг побудови валу, зображеного на рис. 6.15, в AutoLisp:

```
;Побудова вала
;Горизонтальні лінії
(command "_LAYER" "Make" "solid 1" "")
(command "_LINE" "30, 80" "80, 80" "")
(command "_LINE" "30, 120" "80, 120" "")
(command "_LINE" "80, 70" "180, 70" "")
(command "_LINE" "80, 130" "180, 130" "")
(command "_LINE" "180, 90" "210, 90" "")
(command "_LINE" "180, 110" "210, 110" "")
;Вертикальні лінії
(command "_LINE" "30, 80" "30, 120" "")
(command "_LINE" "80, 70" "80, 130" "")
(command "_LINE" "180, 70" "180, 130" "")
(command "_LINE" "210, 90" "210, 110" "")
;Паз шпони
(command "_LINE" "35, 95" "65, 95" "")
(command "_LINE" "35, 105" "65, 105" "")
(command "_ARC" "C" "65, 100" "65, 95" "65, 105")
;Осьова лінія
(command "_LAYER" "Make" "longdashdot" "")
(command "_LINE" "25, 100" "215, 100" "")
;Невидимі
(command "_LAYER" "Make" "invis" "")
(command "_LINE" "80, 130" "180, 70" "")
(command "_LINE" "80, 70" "180, 130" "")
;
;Фаска ліворуч
(command "_CHAMFER" "Distance" "5" "5")
(command "_CHAMFER" "40, 80" "30, 90")
(command "_CHAMFER" "40, 120" "30, 110")
(command "_LAYER" "Make" "solid 1" "")
(command "_LINE" "35, 80" "35, 120" "")
;Фаска праворуч
(command "_CHAMFER" "Distance" "3" "3")
(command "_CHAMFER" "190, 90" "210, 95")
(command "_CHAMFER" "190, 110" "210, 105")
(command "_LINE" "207, 90" "207, 110" "")
;
;Скруглення
(command "_FILLET" "Radius" "5")
(command "_FILLET" "75, 120" "80, 122")
;Відновлюємо лінію
(command "_LINE" "80, 70" "80, 130" "")
```

```
(command "_FILLET" "75, 80" "80, 78")
(command "_LINE" "80, 70" "80, 130" "")
;
(command "_FILLET" "Radius" "3")
(command "_FILLET" "180, 112" "182, 110")
(command "_LINE" "180, 70" "180, 130" "")
;
(command "_FILLET" "180, 88" "182, 90")
(command "_LINE" "180, 70" "180, 130" "")
;
;Переріз
(command "_CIRCLE" "70, 25" "20")
;паз шпони
(command "_LINE" "65, 35" "75, 35" "")
(command "_LINE" "65, 35" "65, 50" "")
(command "_LINE" "75, 35" "75, 50" "")
(command "_TRIM" "_CROSSING" "0, 40" "100, 50" "" "65, 49" "75, 49" "70,
45" "")
;Штрихування
(command "_LAYER" "Make" "solid 2" "")
(command "_BHATCH" "P" "USER" "45" "2" "N" "72, 25" "")
;
(command "_LAYER" "Make" "longdashdot" "")
(command "_LINE" "45, 25" "95, 25" "")
(command "_LINE" "70, 0" "70, 50" "")
(command "_LAYER" "Make" "solid 1" "")
;
;Простановка розмірів
(command "_LAYER" "Make" "solid 2" "")
(command "_DIMLINEAR" "80, 70" "180, 70" "130, 60")
(command "_DIMLINEAR" "130, 70" "130, 130" "Text" "%%c60" "130,
100")
(command "_DIMLINEAR" "55, 80" "55, 120" "Text" "%%c40" "55, 100")
(command "_DIMLINEAR" "205, 90" "205, 110" "Text" "%%c20" "220,
100")
(command "_LAYER" "Make" "solid 1" "")
;Показати усе
(command "_ZOOM" "_E")
```

Витяг з лістингу створення коду на мові AutoLisp приведений нижче:

```
ofstream df("Drawing.lsp");
void DrawAM()
{
void line(int*, int*, int*, int*);
int x1, x2, y1, y2;
int xc = 400;
int yc = 250;
ld = 130;
D = 185;
df << ";Побудова вала\n";
df << ";\n";

//0 - внутрішній циліндр
//нижня лінія
x1 = xc - intg(ld*1000/2);
y1 = yc - intg(D*1000/2);
x2 = x1 + intg(ld*1000);
line(&x1, &y1, &x2, &y1);

//верхня лінія
y2 = yc + intg(D*1000/2);
line(&x1, &y2, &x2, &y2);

//ліва лінія
line(&x1, &y1, &x1, &y2);

//права лінія
line(&x2, &y1, &x2, &y2);
df << ";Показати усе" << "\n";
df << "(command \"_ZOOM\" \"_E\")";

//закрити файл
df.close();
}

//Побудова лінії
void line(int* x1, int* y1, int* x2, int* y2)
{
df << "(command \"_LINE\" \"" << *x1 << "," << *y1 << "\" \"" << *x2
<< "," << *y2 << "\" \"\")\n";
}
```

Контрольні питання

1. Конструкція окремих деталей і складальних одиниць проектованого двигуна, їх призначення і чинники, що визначають розміри і конструктивне виконання.

2. Методи електромагнітного і теплового розрахунків машини, застосовані в проекті на різних етапах; основні допущення, отримані в проекті результати.

3. Обмотки статора і ротора, їх конструкція, ізоляційні матеріали, схеми обмоток статора і ротора.

4. Постановка завдання оптимізації. Етапи вибору оптимального варіанту, багатокритеріальні оптимізаційні дослідження, відбір підмножини ефективних варіантів.

5. Розкрити поняття приведеної вартості електродвигуна.

6. Яка постановка завдання оптимізації? Які змінні варіюються, їх зв'язок з параметрами електродвигуна?

7. Які чинники дозволяють виконати неавтоматизований пошук оптимальних обмотувальних даних при круговому полі в режимі номінального навантаження?

8. Як відбувається переміщення в області незалежних змінних при автоматизованому пошуку оптимального варіанту?

9. У чому полягає суть методу зовнішніх штрафних функцій з використанням методу покоординатного спуску при пошуку оптимального варіанту електродвигуна?

10. Як відіб'ється збільшення або зменшення повітряного проміжку або напруги мережі живлення на ККД асинхронного двигуна, що працює з номінальним навантаженням? На пусковому або максимальному моментах?

Практичні завдання

1. Ґрунтуючись на матеріалі, викладеному в навчальному посібнику, скласти майстер, що реалізовує перехід між сторінками проектування асинхронного двигуна. Передбачити блокування переходу на наступну сторінку, якщо не було здійснено введення початкових даних.

2. Використовуючи матеріал навчального посібника, на мові C++ скласти програму проектування асинхронного двигуна з короткозамкненим ротором. Етапи проектування розбити на 11 функцій - по числу сторінок майстра проекту.

3. Ґрунтуючись на матеріалі, викладеному в навчальному посібнику, скласти програму на мові C++, що реалізовує метод покоординатного спуску і метод зовнішніх штрафних функцій для урахування обмежень, що накладаються на проект.

4. На мові AutoLisp скласти програму для параметричної побудови листа статора асинхронного двигуна з короткозамкненим ротором.

5. На мові AutoLisp скласти програму для параметричної побудови листа ротора асинхронного двигуна з короткозамкненим ротором.

6. На мові AutoLisp скласти програму для параметричної побудови спрощеного креслення вала асинхронного електродвигуна.

7. На мові C++ скласти програму, що генерує скриптові команди на AutoLisp для побудови листа статора асинхронного двигуна з короткозамкненим ротором.

8. На мові C++ скласти програму, що генерує скриптові команди на AutoLisp для побудови листа ротора асинхронного двигуна з короткозамкненим ротором.

РОЗДІЛ 6

ОБ'ЄКТНО-ОРІЄНТОВАНЕ ПРОЕКТУВАННЯ ЕЛЕКТРОМЕХАНІЧНИХ ПРИСТРОЇВ

Однією з особливостей живої природи є її здатність породжувати потомство, що має характеристики, схожі з характеристиками попереднього покоління. Природі було потрібно мільярди років, щоб досягти сучасних форм життя, і результат цих зусиль передається кожному представникові окремого виду живих організмів. Це результат спадкоємства. Якщо спадкоємство призводить до таких результатів в природі, то чому воно не може бути корисне в проектуванні? Електричні машини можна розбити на класи, відповідні принципам формування конкретних видів машин з узагальнених, таких, що мають універсальні для породжених класів характеристики [1]. Спадкоємство дозволяє, практично без обмежень, послідовно будувати і розширювати створені класи. Кожного разу, коли з попереднього класу виробляється наступний, похідний клас наслідує якісь або усі батьківські якості, додаючи до них нові. Завершений проект може включати десятки і сотні класів, але усі вони можуть бути вироблені від ліченої кількості базових класів.

Породжуючи класи від базових, можна ефективно використовувати алгоритми базового класу для власних потреб. Концепція має паралель в живій природі: ДНК можна розглядати як базовий матеріал, з якого може бути створена будь-яка істота. Кожен організм повторно використовує ДНК для відтворення свого власного виду. У проектуванні повторне використання здійснюється через спадкоємство.

6.1 Проблеми процедурного підходу в проектуванні

Традиційне проектування супроводжується процедурним підходом до рішення завдань розрахунку електричної машини. Часто, наприклад в курсовому проектуванні, розрахунки розбиваються на пункти з набором формул, що виконуються послідовно. Проект, побудований на основі процедурного методу, розділяється на функціональні модулі, кожен з яких в ідеальному випадку виконує деяку закінчену послідовність дій і має явно виражені зв'язки з іншими функціональними модулями програми.

Можна розвинути ідею розбиття проекту на функціональні модулі, об'єднавши декілька функціональних модулів в один блок (іноді блок представляє окремий проект). При цьому зберігається процедурний принцип: проект ділиться на декілька компонентів, кожен з яких є набором інструкцій. Ділення проекту на функціональні модулі і блоки є основою традиційного структурного проектування. Проте, як би ефективно не застосовувався структурний підхід, він не дозволяє достатньою мірою спростити великі і складні проекти. Існує дві основні проблеми процедурного підходу. Перша полягає в необмеженості доступу функціональних модулів проекту до глобальних даних. Друга полягає в тому, що розподіл даних і методів їх обробки, що є основою структурного підходу, неадекватно відбиває картину реального світу [1].

Розглянемо приклад неконтрольованого доступу даних на прикладі програми проектування асинхронного двигуна з короткозамкненим ротором. У проекті виділимо наступні функціональні модулі:

1) вибір головних розмірів;

2) визначення числа зубців Z_1, числа витків W_1 і перерізу проводу обмотки статора;

3) розрахунок розмірів зубцевої зони статора і повітряного проміжку;

4) розрахунок ротора;

5) розрахунок струму намагнічування;

6) параметри робочого режиму;

7) розрахунок втрат;

8) розрахунок робочих характеристик;

9) розрахунок пускових характеристик;

10) тепловий розрахунок;

11) механічний розрахунок вала.

У процедурній програмі, написаній, приміром, на мові С++, існує два типу даних. Локальні дані знаходяться усередині функціональних модулів і призначені для використання виключно цими функціональними модулями. Наприклад, у функціональному модулі п. 6 "параметри робочого режиму" використовуються локальні дані коефіцієнтів пазового, лобового і диференціального розсіяння. Локальні дані цього функціонального модуля недоступні нікому, окрім самого модуля, і не можуть бути змінені іншими модулями.

Якщо існує необхідність спільного використання одних і тих же даних декількома функціями, то дані мають бути оголошені як глобальні. Це, як правило, торкається тих даних проекту, які є найбільш важливими. Будь-який функціональний модуль має доступ до глобальних даних. Схема, що ілюструє концепцію локальних і глобальних даних, приведена на рис. 6.1.

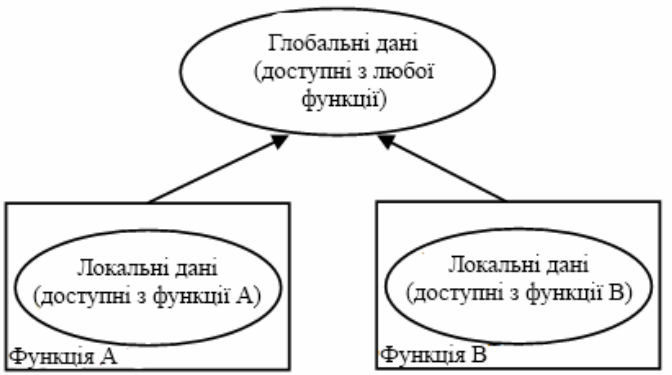


Рисунок 6.1 - Глобальні і локальні дані

Прикладом глобальних змінних можуть служити потужності, частота обертання, струми, активні і індуктивні опори обмоток статора і ротора, кількість зубців і витків обмоток і так далі.

Великі програми зазвичай містять безліч функціональних модулів і глобальних змінних. Проблема процедурного підходу полягає в тому, що кількість можливих зв'язків між глобальними змінними і функціональними модулями може бути дуже велике, як показане на рис. 6.2.

Рисунок 6.2 - Процедурний підхід

Велика кількість зв'язків між функціями і даними, у свою чергу, також породжує декілька проблем. По-перше, ускладнюється структура програми. По-друге, в програму стає важко вносити зміни. Зміна структури глобальних даних може привести до переписування усіх функціональних модулів, що працюють з цими даними. Наприклад, якщо розробник програми вирішить записувати опори в комплексній формі виду Z = R + jX, то буде необхідно внести новий тип даних, який підтримує операції з комплексними числами. Крім того, струми також треба буде записати в комплексній формі. Це означає, що в усі функціональні модулі, що оперують з опорами і струмами, мають бути внесені зміни, які дозволяють обробляти дані в комплексній формі. Можна навести аналогічний побутовий приклад, коли в супермаркеті змінюється розташування відділів, і покупцям доводиться відповідним чином міняти свій звичний шлях від одного відділу до іншого.

Коли зміни вносяться до глобальних даних великих програм, буває непросто визначити, які функціональні модулі необхідно скоректувати. Навіть у тому випадку, коли це вдається зробити, із-за численних зв'язків між функціональними модулями і даними виправлені модулі починають некоректно працювати з іншими глобальними даними. Таким чином, будь-яка зміна спричиняє за собою наслідки, що далеко йдуть.

Друга, важливіша проблема процедурного підходу полягає в тому, що відокремлення даних від розрахунку у функціональних модулях опиняється малопридатним для відображення картин реального світу. У реальному світі нам доводиться мати справу з фізичними об'єктами, такими, наприклад, як люди і електричні машини. Ці об'єкти

неможна віднести ні до даних, ні до функціональних методів, оскільки реальні речі представляють сукупність властивостей і поведінки.

Прикладами властивостей (характеристик) для людей можуть являтися колір очей або місце роботи; для електричних машин - потужність двигуна і кількість зубців статора. Таким чином, властивості об'єктів рівносильні даним в програмах: вони мають певне значення, наприклад блакитне для кольору око або 36 для числа зубців статора.

Поведінка - це деяка реакція об'єкту у відповідь на зовнішню дію. Наприклад, людина на прохання про допомогу може дати відповідь «так» або «ні». Якщо ввести активний опір в коло ротора асинхронного двигуна з фазним ротором, то станеться збільшення критичного ковзання і, як наслідок, гальмування двигуна. Відповідь і гальмування є прикладами поведінки. Поведінка схожа з функціональними модулями: функціональний модуль викликається, щоб зробити якунебудь дію (наприклад, розрахувати величину пускового струму), і функціональний модуль здійснює цю дію.

Таким чином, ні окремо взяті дані, ні окремо взяті функції не здатні адекватно відобразити об'єкти реального світу.

Основоположною ідеєю об'єктно-орієнтованого підходу є об'єднання даних і дій, вироблюваних над цими даними, в єдине ціле, яке називається об'єктом.

Функції об'єкту, або функціональні модулі (також їх називають функції, методи), призначені для доступу до даних об'єкту. Якщо необхідно отримати які-небудь дані об'єкту, треба викликати відповідний метод, який викликає дані і поверне необхідне значення. Прямий доступ до даних ззовні неможливий. Дані об'єкту приховані від зовнішньої дії, що захищає їх від випадкової зміни.

Говорячи в термінах мови програмування С++, дані і методи об'єкту інкапсульовані.

Якщо необхідно змінити дані об'єкту, то, очевидно, ця дія також буде покладена на методи об'єкту. Ніякі інші функції не можуть змінювати дані об'єкту. Такий підхід полегшує написання, редагування і використання проекту. При цьому, типовий проект складається з сукупності об'єктів, що взаємодіють між собою за допомогою виклику методів один одного. Структурна схема об'єктно-орієнтованого проекту представлена на рис. 6.3.

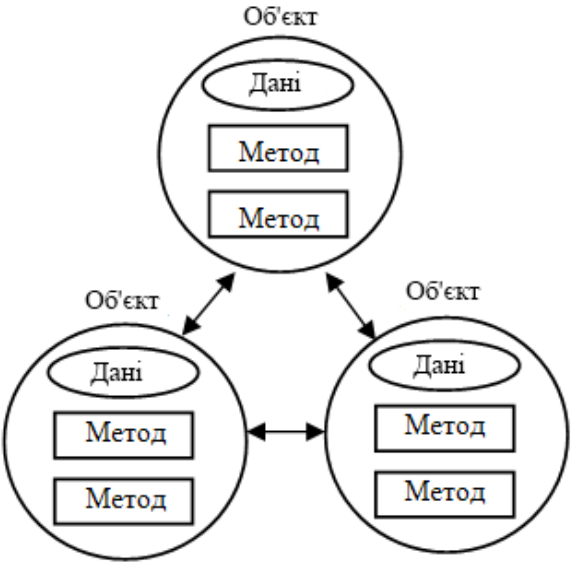


Рисунок 6.3 - Об'єктно-орієнтований підхід

При підході до рішення задачі за допомогою об'єктно-орієнтованого методу, замість проблеми розбиття завдання на функції, ставиться питання розбиття цього завдання на об'єкти.

Мислення в термінах об'єктів виявляється простішим і наочним, чим в термінах функцій, оскільки програмні об'єкти схожі з об'єктами реального світу. Наведемо приклад таких об'єктів.

Фізичні об'єкти - статор, ротор, обмотка, електродвигун.

Параметричні об'єкти: комплексні числа, масиви, час.

Графічні об'єкти: лінії, прямокутники, круги.

Відповідність між програмними і реальними об'єктами є наслідком об'єднання даних і функцій. Жодна програмна модель, створена на основі процедурного підходу, не здатна відбити існуючі речі так точно, як це вдається зробити за допомогою об'єктів.

Коли йде мова про об'єкти, то треба мати на увазі, що вони є екземплярами класів. Це означає, що клас є свого роду формою, що визначає, які дані і функції будуть включені в об'єкт класу. При формуванні класу не створюються ніякі об'єкти цього класу, по аналогії з тим, що існування електричної машини на кресленні ще не означає наявність виготовленої по них машини. Таким чином, клас є описом сукупності схожих між собою об'єктів. Наприклад, синхронні генератори і асинхронні двигуни відносяться до класу машин змінного струму. Не існує конкретної машини, в якій замість серійного номера було б написано "машина змінного струму", проте машини зі своїми унікальними серійними номерами є об'єктами цього класу, якщо вони мають певний набір характеристик. Тому об'єкт класу і називається екземпляром класу. У проектуванні об'єкти, що мають усі ознаки своїх класів, створюються при виконанні розрахункової програми, коли під них фізично виділяється пам'ять.

Об'єктно-орієнтоване проектування ніяк не пов'язане з процесом виконання проекту і розрахунками, а є лише способом його організації.

Розроблені класи можуть бути повторно використані в інших проектах, реалізовуючи концепцію повторного використання. Проектант може узяти існуючий клас, і, нічого не змінюючи, внести до нього свої елементи. Усі похідні класи успадкують ці зміни, і в той же час кожен з похідних класів можна модифікувати окремо.

Легкість повторного використання є важливою перевагою об'єктно-орієнтованого проектування.

Сформулюємо основні проблеми процедурного підходу в проектуванні.

1) Традиційне структурне проектування з процедурним підходом супроводжується двома основними проблемами:

а) збільшення, із зростанням складності проекту, числа зв'язків між глобальними даними і розрахунковими функціональними модулями;

б) відокремлення даних від функціональних модулів непридатне для відображення об'єктів реального світу, що мають властивості і поведінку.

2) Наявність проблем процедурного підходу ускладнює зміну і супровід проекту, робить нездійсненною концепцію спадкоємства.

3) Поєднання даних і функціональних модулів в одному об'єкті адекватно відображує властивості і поведінку об'єктів реального світу.

4) Рішенням проблем процедурного проектування є перехід до об'єктно-орієнтованого проектування, яке є основою спадкоємства, поліморфізму і повторного використання.

6.2 Об'єктно-орієнтоване проектування електромеханічних перетворювачів енергії з суміщеними функціями

Застосування методів об'єктно-орієнтованого проектування актуальне для електромеханічних перетворювачів енергії (ЕМПЕ), що мають декілька технологічних функцій, поєднаних в одному пристрої. Завдяки концепції спадкоємства можна створити клас ЕМПЕ, породжений від декількох базових класів, кожен з яких виконує певну технологічну функцію. При цьому функції реальних об'єктів повинні адекватно відтворюватися функціями-членами відповідних класів. Таким чином, створений в результаті множинного спадкоємства клас-нащадок інтегруватиме в собі властивості базових батьківських класів.

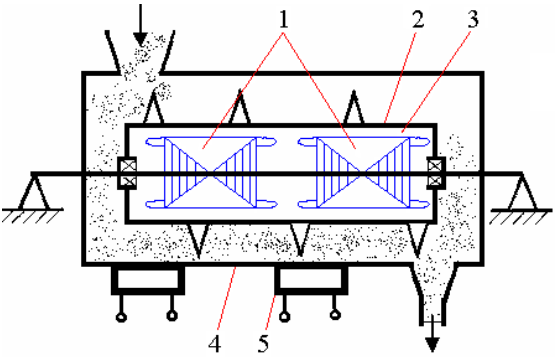
Розглянемо об'єктне представлення електротепломеханічний перетворювач (ЕТМП) шнекового типу, що поєднує в собі функції сушки, перемішування і транспортування сипких матеріалів [11].

Створення ресурсо- і енергозбережних технологій базується на двох основних напрямах. Перше з них пов'язано з підвищенням ККД окремих елементів системи перетворення енергії. Друге - засновано на інтеграції функціональних властивостей одним агрегатом і використанні дисипативної енергії. Оскільки перше з напрямів при розвиненій системі оптимізації конструктивних рішень визначається, передусім, створенням нових активних і ізоляційних матеріалів, темпи його розвитку обмежені.

Для технологічних систем, що об'єднують процеси транспортування, нагріву, перемішування матеріалів, найбільш перспективним слід вважати другий напрям. В цьому випадку стає можливим збереження ресурсів за рахунок об'єднання окремих елементів устаткування в одному корпусі і, що дуже важливо, використання дисипативної ене-

ргії вказаних елементів, яка раніше при традиційній схемі перетворення і використання енергії марно розсіювалася в довкілля. Одним з основних шляхів підвищення техніко-економічних показників електроприводу є створення поєднаних в одному корпусі двигунів насосів, двигунів-мішалок, двигунів-шнеків. Це перспективний клас електромеханічних систем, що призначених для безпосереднього здійснення технологічних процесів і відрізняються посиленою концентрацією функціональних і енергетичних властивостей.

Конструкція ЕТМП представлена на рис. 6.4.



1 - статори; 2 - масивний ротор шнека; 3 - повітряний проміжок; 4 - днище шнека; 5 - нагрівальна система днища.

Рисунок 6.4 - Конструктивна схема ЕТМП

ЕТМП складається з двох модулів, що працюють в режимі противключення. Два статори, посаджені на загальний порожнистий вал, створюють зустрічно спрямовані електромагнітні моменти, забезпечуючи необхідну швидкість обертання порожнистого циліндра загального ротора без застосування механічного редуктора. Ротор, що має

шнекову навивку, окрім функції переміщення робочого матеріалу одночасно забезпечує нагрів останнього.

Для отримання класового представлення ЕТМП, необхідно виділити ключові слова, що є абстрактним представленням електричної машини. Слід зазначити, що не усі ключові слова увійдуть до класової структури - вони допоможуть систематизувати уявлення про ЕТМП як про складний об'єкт. Наприклад, можна розглядати наступні ключові слова:

- статор; статор внутрішній; статор зовнішній; ротор; ротор зовнішній; ротор внутрішній; паз статора; паз статора внутрішнього; паз статора зовнішнього; обмотка; статор внутрішній з пазами внутрішнього статора і обмоткою; статор зовнішній з пазами зовнішнього статора і обмоткою; ЕТМП із зовнішнім ротором; ЕТМП з внутрішнім ротором.

Наведені ключові слова не є просто позначенням структурних складових електричної машини і, як може помилково здатися, необхідні лише для розуміння складу ЕТМП. Ключові слова слід розглядати як класи, що містять змінні і функції окремого елементу структури, наділені його властивостями і характерною поведінкою при заздалегідь відомих діях. Змінні класів і функції, що характеризують роботу і властивості ЕТМП, можна визначити вже на ранньому етапі класового проектування.

Функціональні характеристики ЕТМП будуть наступні:

- параметри класів;

- енергетичні показники класів;

- коефіцієнти і постійні класів.

Перераховані властивості ЕТМП, що визначають розрахунковий склад проектування, також можна виділити в окремі класи. Проте це привело б до багатократного створення вкладених класів усередині відповідних основних. Наприклад, для класу "статор" буде потрібно створення вкладених класів "магнітна система статора", "параметри статора", "енергетичні показники статора", "коефіцієнти і постійні величини статора". Розглядаючи повне класове представлення ЕТМП, можна прийти до досить складної системи класів з ще складнішими внутрішніми зв'язками. Завдання будь-якого проектування - прийти до максимально оптимальної і простої структури класів, зручної в розрахунках, розумінні, і, нарешті, в складанні розрахункової програми. Доцільно виділені функції і змінні включити у відповідні класи як функції і змінні самого класу, а не як окремі класові структури. Класове дерево електромеханічної структури ЕТМП можна сформувати як показано на рис. 6.5.

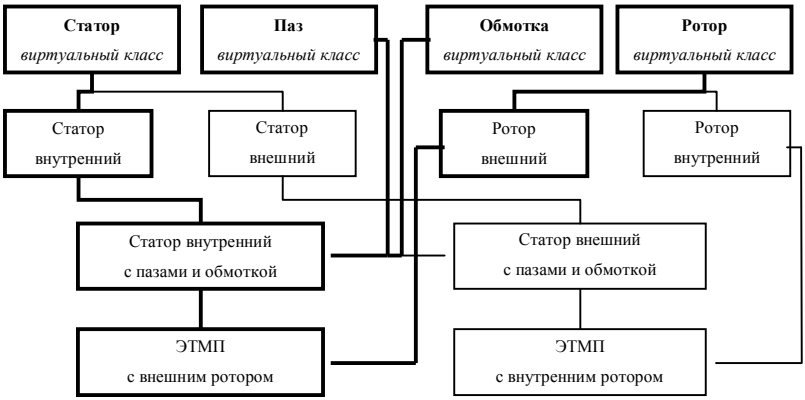


Рисунок 6.5 - Дерево спадкоємства електромеханічної структури ЕТМП

На рис. 6.5 як базові прийняті віртуальні класи "статор", "паз", "обмотка", "ротор". Віртуальний клас "статор" містить два похідних класу "статор внутрішній" і "статор зовнішній". Кінцеві класи "ЕТМП із зовнішнім ротором" і "ЕТМП з внутрішнім ротором" породжені від відповідних класів і мають усі характеристики батьківських класів, а також привносять свої власні дані, які дають повне уявлення про роботу і поведінку ЕТМП.

Програма проектування реалізується на мові програмування С++, яка підтримує об'єктно-орієнтоване програмування [1].

При розгляді методики проектування ЕТМП на основі класового представлення доцільно дотримуватися послідовності виклику функцій і змінних дерева класів, а не порядку їх перерахування в тілі відповідних класів. Таким чином, динаміка поведінки об'єкту "ЕТМП із зовнішнім масивним ротором" буде краще простежуватися.

Будь-яке проектування електричної машини починається із завдання початкових даних. Цими даними, як правило, являються: корисна потужність P_2, кількість пар полюсів p, номінальна напруга статора $U_н$, група з'єднання обмоток, ступінь захисту, режим роботи, робоча температура. Крім того, існують рекомендації по орієнтовному номінальному ковзанню машини $s_н$.

Методика проектування звичайних асинхронних машин розроблена вже більш за півстоліття назад, постійно удосконалювалася, підводилася під нові ізоляційні, конструкційні матеріали, нові технічні рішення, виробничий процес, обчислювальну базу. Теорія і багаторічний досвід виробництва електричних машин дозволили створити емпіричні залежності, графіки, таблиці, які задовільною для інженерних

розрахунків точністю дозволяють виконати проект з необхідними характеристиками машини.

ЕТМП - абсолютно новий пристрій, і практика його застосування налічує декілька років. Це, з одного боку, ускладнює завдання проектування, з іншого - дозволяє підійти до проекту більш скрупульозно.

Наприклад, розрахунок і проектування звичайного асинхронного двигуна починається із завдання розрахункової потужності, необхідної для роботи приводу, по якій вибирають висоту осі обертання, основні геометричні розміри, попередні значення ККД, cosφ.

З ЕТМП справа йде інакше. Складнощі починаються вже на етапі вибору розрахункової потужності. Проектна потужність визначається двома складовими:

- теплова потужність $P_т$, необхідна для висушування заданої маси вологого сипкого матеріалу при його проходженні по довжині ротора шнека, що обертається з постійною швидкістю;

- механічна потужність $P_{мех}$, необхідна для подолання моменту опору матеріалу обертанню ротора шнека і опору, що створюється системою примусової вентиляції.

Для рухового модуля ЕТМП проектна потужність вибирається виходячи з суми теплової $P_т$ і механічною $P_{мех}$ потужностей. Для гальмівного модуля, що працює в режимі противключення, - тільки теплової.

Для розрахунку вказаних потужностей $P_т$ і $P_{мех}$ потрібне розуміння властивостей оброблюваного матеріалу при нагріві і перемішуванні, а, отже, проведення попередніх теплових і гідродинамічних розрахунків. Ці розрахунки вже включені в тіло класу "статор" і викликаються до початку проектування власне ЕТМП.

Частота обертання ротора n_p виходить як різниця швидкостей рухового і гальмівного модуля і дозволяє визначити номінальне ковзання для кожного з модулів. Номінальне ковзання рухового модуля дорівнюватиме ковзанню ротора $s_{н1} = s_p$. Номінальне ковзання гальмівного модуля $s_{н2} = 2 - s_{н1}$.

Проектування ЕТМП доцільно розглядати для режиму максимального і тривалого струмового навантаження. При цьому обмотка має бути вибрана так, щоб при протіканні максимально можливого струму, густина струму в обмотках не перевищувала допустиме значення по тепловому навантаженню машини.

Після прийняття початкових даних відбувається формування об'єктів класу ЕТМП. Спочатку створюється базовий тип машини змінного струму. Він може містити дані, властиві усім машинам змінного струму - номінальна напруга, потужність, синхронна частота, кількість фаз, робоча температура, частота мережі. Цей тип є чисто віртуальним і служить для створення класів-нащадків. Термін "віртуальний" означає, що цей клас, наприклад, не може служити для створення об'єктів, а є лише носієм інформації. Об'єкт же, після завершення проекту, придбає унікальні характеристики конкретної машини з вибраною лінією спадкоємства в дереві класів.

Клас двигунів служить для позначення призначення машини і відповідного напряму проекту.

Рід асинхронних двигунів приймає дані батьків, ідентифікує змінні з метою їх залучення до конкретнішого призначення.

Вид асинхронних двигунів з масивним ротором остаточно підводить до створення об'єкту ЕТМП і несе інформацію про структуру

машини. Вид має такі відмітні вхідні дані, як вторинна частота, температура ротора, ковзання.

І, нарешті, створюється глобальний об'єкт ЕТМП, який має підоб'єктну структуру і містить в собі повну інформацію про машину (включаючи дані батьків).

Алгоритм формування об'єкту ЕТМП на базі підоб'єктів (ПО) має наступний вигляд. При формуванні об'єкту ЕТМП запускається конструктор, що викликає потрібні модулі, створюючи при цьому ПО глобальної структури. Порядок виклику конструкторів об'єкту наступний:

1) асинхронний двигун з масивним ротором (отримання початкових даних і втрат в роторі);

2) статор / внутрішній статор (геометричні розміри, сталь);

3) масивний ротор (геометричні розміри, сталь);

4) паз (розміри і заповнення), обмотка (обмотувальні дані) - запускається з конструктора паза;

6) магнітна система масивного ротора (розрахунок магнітної системи машини з урахуванням того, що ротор є масивним);

7) параметри (параметри статора);

8) параметри масивного ротора;

9) енергетичні показники масивного ротора (розрахунок струмів, втрат, моментів, ККД).

Оскільки на первинному етапі проектування окрім завдання початкових даних не вироблено ще ніяких розрахунків, об'єкти, що є складовими частинами машини, створюються у вигляді шаблонів з нульовими даними. Ці дані, що є геометричними розмірами, застосо-

ваними матеріалами, електромагнітні параметри і так далі будуть отримані надалі при виклику функцій-членів відповідних класів.

Спочатку викликається функція класу "статор внутрішній" для розрахунку основних геометричних розмірів і коефіцієнтів.

Робоча довжина ротора L_p, отримана в результаті гідродинамічних і теплових розрахунків робочого нагрівально-охолоджувального середовища (РНОС) з урахуванням вимог технічного завдання, розбивається на певну кількість рухових і гальмівних модулів. Принципово не існує обмежень на кількість таких модулів і не обов'язкова кількість гальмівних і рухових модулів має бути однаковою. Цей вибір визначається, передусім, трудомісткістю і технологією виготовлення.

Довжина ділянки ротора, яка доводиться на окремий модуль, може бути отримана за наступним виразом:

$$L_a = \frac{L_p - \sum \delta_{e.з}(N_{РМ} + N_{ГМ} + 1)}{N_{РМ} + N_{ГМ}}, \qquad (6.1)$$

де $\delta_{e.з}$ - величина електричного проміжку між суміжними лобовими частинами статорних модулів або лобовими частинами і щитами ротору. Вибирається відповідно до [9].

Співвідношення довжин ділянок ротора для рухівного (РМ) і гальмівного (ГМ) модулів визначається виходячи із збереження однакового рівня джоулевих втрат, що доводяться на одиницю активної довжини ротора:

$$\frac{L_{a(РМ)}}{L_{a(ГМ)}} = \frac{2 - S_{РМ}}{S_{РМ}}. \qquad (6.2)$$

Зовнішній діаметр масивного ротора визначений виходячи з числових і гідродинамічних розрахунків (РНОС)[11]. Товщина ротора вибирається виходячи з необхідної механічної міцності і, як правило,

складає дві-три глибини проникнення електромагнітної хвилі. Величина повітряного проміжку вибирається виходячи з рекомендацій [9] і складає 1,0-2,0 мм.

Обчислюються довжина пакету статора l_δ, зовнішній D і внутрішній $D_{вн}$ діаметри статора, полюсне ділення τ, питома поверхнева потужність ротора p_o.

За величиною p_o знаходяться магнітна проникність μ_2 і питомий електричний опір ρ_2 масивного ротору.

Визначається МРС ротору F_2

$$F_2 = k_\phi k_\Delta p \sqrt{\frac{\sqrt{S_н} \cdot \mathrm{Re}(\dot{A}) P_н (L_2 + \tau)}{\pi l_\delta m \cdot \sqrt{\rho_2 \mu_e f_1'}}}, \qquad (6.3)$$

де $\dot{A} = 1{,}13 + j1{,}85$ - коефіцієнт Нейману; $P_н$ - розрахункова потужність; m - кількість фаз; $\rho_2 = f(^o t_{рот})$ - питомий опір сталі ротора; μ_e - відносна магнітна проникність матеріалу ротору; L_2 - довжина активної частини ротору; k_ϕ - коефіцієнт форми поля; $k_\Delta = f(l_\delta / \tau)$ - коефіцієнт відносного подовження ротора.

За величиною МРС (6.1) знаходимо попереднє значення комплексного магнітного опору масивного ротора.

Слід пояснити наведені вирази (6.1) - (6.3). У більшості джерел за розрахунком масивно-роторних асинхронних машин можна зіткнутися з таким парадоксальним фактом, що МРС і струми масивного ротора визначаються по його параметрах, а останні неможливо розрахувати без відомих значень струмів і МРС. Виходом є розрахунок ряду струмів при різних параметрах ротора з наступним вибором того, що найбільш задовольняє початковим умовам. Але це підходить більше для перевірочних розрахунків при відомих результатах експерименту,

коли можна орієнтуватися за величиною вторинних струмів. До проектування такий підхід непридатний. Природно, до представлення ЕТМП у вигляді диференціальних рівнянь і їх чисельному рішенню на ЕОМ це не відноситься, але в інженерних методиках вони використовуються рідко.

У випадку з ЕТМП знайдений оригінальний підхід, що заслуговує уваги. Одним з найважливіших початкових даних, які недоступні для звичайних масивно-роторних машин, є втрати в роторі.

У нашому ж випадку втрати в роторі вже відомі за попередніми тепловими і гідродинамічними розрахунками і з'являється можливість визначити шукану МРС ротора F_2 і його комплексний магнітний опір.

Отриманих даних вистачає для визначення основних величин, що характеризують статор ЕТМП : числа витків обмотки статора, попереднього значення магнітної індукції в повітряному проміжку, модуля вторинного струму, МРС повітряного проміжку, магнітного опору повітряного проміжку, МРС магнітного кола машини, повного струму, лінійного навантаження, коефіцієнта потужності.

Функція, що викликається з класу "паз", дозволяє зробити розрахунок зубцевої зони, після якого стають відомими максимальне t_{1max} і мінімальне t_{1min} зубцеві ділення, а також діапазон можливого числа зубців статора Z_{min} - Z_{max}.

Далі управління розрахунком передається в руки розробника - надається можливість вибору числа зубців статора Z_1, виду паза статора, типу обмотки (одношарова або двошарова). Після прийняття цих даних виробляється розрахунок і підбір оптимального значення числа ефективних провідників $U_п$, вибирається кількість паралельних гілок. Уточнюються: кількість витків обмотки статора, лінійне навантажен-

ня, основний магнітний потік, магнітна індукція в повітряному проміжку.

При цьому продовження розрахунків можливе, якщо отримана густина струму в обмотці статора не перевищує допустиме значення.

На наступному етапі вибирається обмотувальний провід, розраховується конфігурація і заповнення паза статора.

База даних обмотувального прямокутного і круглого проводу, закладена в програму розрахунку, дозволяє без зусиль його підібрати.

По вже відомим значенням магнітних індукцій в зубцях і ярмі, знаходяться відповідні магнітні напруженості при вибраній марці сталі листа статора.

Далі виробляється розрахунок магнітного кола машини. Як відомо, тіло масивного ротору в розрахунок кола намагнічування не входить. Основні ділянки кола - це зубцева зона, ярмо статора і повітряний проміжок. По відомих виразах визначається МРС повітряного проміжку F_{δ}, зубців F_z, ярма F_a і сумарна МРС $F_{сум}$, знаходяться коефіцієнт насичення k_{μ} і струм намагнічування I_{μ}, визначається індуктивний опір гілки намагнічування .

Розрахунок параметрів обмотки статора, а саме її активного r_1 і індуктивного x_1 опорів, виробляється по відомим виразам [9, 10] з урахуванням конструктивного виконання статора (зовнішнього або внутрішнього).

Важливий етап, що стосується розрахунку приведених параметрів масивного ротора, виробляється з урахуванням робочої температури, кінцевої довжини, насичення і, при необхідності, кривизни масивного ротора. Параметри машини для ковзань, відмінних від s = 1, зна-

ходяться перерахунком по відомих виразах при урахування величини приведеного струму ротора I'_2.

По знайдених параметрах визначаються величини, необхідні для побудови робочих і пускових характеристик ЕТМП. Характеристики електричної машини з масивним ротором розраховуються з урахуванням впливу насичення масивного ротора. Це досягається введенням в розрахункові формули поточного значення критичного ковзання $s'_{кр}$. Змінній величиною в розрахунках робочих і пускових характеристик буде поточне $s_i = s_i - 1 + \Delta s$ і критичне ковзання.

Склад функцій-членів класів дозволяє провести тепловий розрахунок ЕТМП, розрахувати пусковий режим при дії змінних навантажень. На закінчення хотілося б відмітити, що результати розрахунків за розглянутою методикою дають задовільну збіжність з результатами експериментальних досліджень, як фізичних моделей, так і виготовлених експериментальних, а також серії промислових зразків ЕТМП з погрішністю 8 - 12 %, допустимою при інженерних розрахунках.

За представленою методикою було розроблено два типи розрахункових програм : перший - з використанням низхідного структурного програмування, другий - із застосуванням теорії класів.

Об'єктно-орієнтований код структури класу ЕТМП наводиться нижче.

```
//Клас ЕТМП із зовнішнім масивним ротором
class ETMP: virtual public Stator, Paz, MassiveRotor
{
protected:
//Змінні класу
public:
//Функції-члени
void Init();
```

```
void Geom(double);// розрахунок основних геом. розмірів
void Zubc(int);// розрахунок зубцевої зони
void Magn(int);// розрахунок струмів, магнітної системи
void ParamS();// розрахунок параметрів статора
void ParamR();// розрахунок параметрів масивного ротору
void Wind(double, int);// розрахунок обмотувального провіду
void RecalcPaz(int);// перерахунок статора при зміні пазу
void RecalcWind();// перерахунок даних статора обмотки
void Char();// розрахунок робочих і пускових характеристик
void DataOutput();//виведення розрахованих даних у файл
void Graph();// побудова графіків
void Paz_PR();//попередній розрахунок прямокутного пазу
void PazPR_End(double, double, double, double, int);
//розрахунок пазу
void Paz_TR(double, double, double, int, double, double, double, double);
double SetTPaz(int, double);

ETMP();// конструктор класу
//~ETMP();// деструктор класу
};
```

У лістингу наведені тільки категорії змінних без детального опису, що, проте, не впливає на розуміння об'єктно-орієнтованого підходу. Порівняльний код при структурному підході не приводиться, оскільки для реалізації представленої в об'єктно-орієнтованому варіанті програми було б потрібно набагато більше рядків програмного коду. Крім того, об'єктний підхід дозволяє опустити опис змінних і функцій і при цьому зберегти розуміння програми. При структурному підході для пояснення логіки роботи програми спрощення і розриви в послідовному виконавчому коді неможливі, оскільки повністю обривають розуміння матеріалу, що викладається, зважаючи на втрату структурних зв'язків.

Отже, було отримано класове представлення внутрішньої структури ЕТМП і реалізація його електромеханічної функції. Розглянемо

інтеграцію теплових і гідромеханічних функцій при формуванні результуючого класу. Наведені вище класи і дерево спадкоємства ЕТМП не зачіпають процесів, пов'язаних з транспортуванням і нагрівом сипкого матеріалу, що переробляється. Тим часом, процеси гідродинаміки і теплообміну ініціює ЕТМП. Класове представлення ЕТМП має бути доповнене бракуючими базовими класами, які адекватно відбиватимуть картину реального світу. Таким чином, поліфункціональний ЕТМП з'явиться як результат множинного спадкоємства з трьома базовими класами (рис. 6.6).

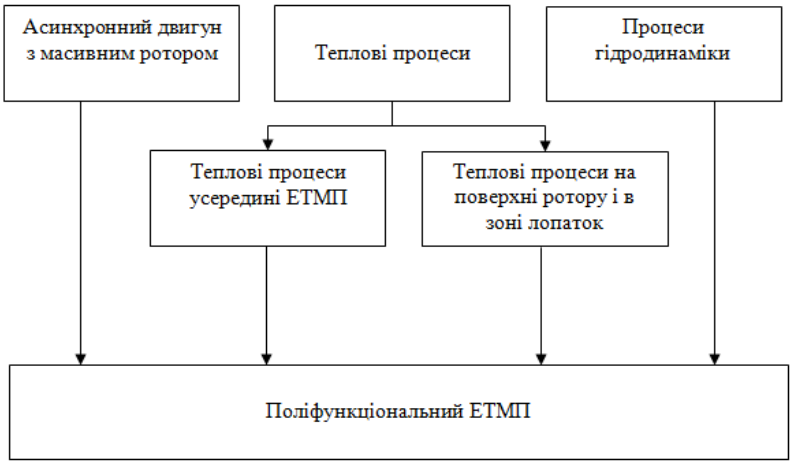


Рисунок 6.6 - Класове представлення ЕТМП з інтеграцією технологічних функцій

Клас гідродинамічних процесів привносить функціональні модулі, пов'язані з транспортуванням, перемішуванням і завихренням матеріалу в міжлопатковій області ротора.

Клас теплових процесів, представлений у вигляді двох підкласів, відбиває процеси теплообміну як усередині ЕТМП, так і на зовнішній поверхні масивного ротора.

Перевага класового представлення проявляється в тому, що базові класи зовнішніх процесів пов'язані з ЕТМП тільки за допомогою функціональних модулів і не вбудовані в його внутрішню структуру. При зміні умов роботи ЕТМП (наприклад, в середовищі в'язкого легкоплавкого матеріалу, такого як озокерит, парафін, легкоплавкі фракції бітуму тощо) будуть повністю заміщені базові класи гідродинаміки і теплообміну при збереженні міжкласових зв'язків. Класи, що представляють "чорний ящик" з множиною входів і виходів, для кінцевого об'єкту виглядають як потік даних: температура, теплопровідність, в'язкість і так далі. Реалізація ж складних процесів гідродинаміки, тепло-масообміну і теплопередачі прихована усередині класів. Кінцевий користувач при цьому зосередить свою увагу на проектуванні електромеханічної системи, беручи до уваги вплив гідродинаміки і теплообміну у вигляді зручних для розрахунків коефіцієнтів і змінних, не зупиняючись на їх розрахунку. Усі розрахунки вже будуть зроблені при формуванні класу-нащадка.

Лістинг класового представлення поліфункціонального ЕТМП, у відповідність з деревом спадкоємства, представленим на рис. 6.6, приведений нижче.

```
class PETMP: virtual public ETMP, Hydro, ThermoETMP, ThermoRotor
{
//Змінні класу
public:
//Функції-члени
ETMP();// конструктор класу
};
```

У лістингу вказані тільки основні породжувальні структури, без детального опису змінних. Потрібно відмітити, що описи класів (змінних, констант, функцій) розділені від реалізації класів (власне, розрахункової методики). Не зачіпаючи структуру класів, що є ідеологією проекту електричної машини, досить внести зміни до розрахункової методики. При цьому для користувача-проектувальника ніяких змін в розрахунку не виникне. Він отримує "чорний ящик" з набором вхідних і вихідних даних. Методика розрахунку, закладена в реалізацію класів, прихована ("інкапсульована"). Оскільки ідеологія класів не міняється, то залишаються без зміни усі алгоритми, порядок виклику функцій, звернення до них. Усі зміни залишаються усередині функцій і не порушують злагоджену міжкласову взаємодію.

При структурному підході дотримується послідовне представлення розрахункової програми. Поняття "функція-член" не існує - є тільки підпрограми вищого і нижчого рівнів. Будь-яка зміна підпрограм, самого коду у верхніх рівнях призводить до непрацездатності усього коду програми, розташованого нижче внесених змін, оскільки зв'язок в програмі при структурному підході строго послідовний і жорстко пов'язаний як по самих підпрограмах, так і по їх аргументах. Послідовність не міняють навіть розгалужені цикли - ідеологія структурного підходу разюче відрізняється від об'єктно-орієнтованого.

Як результат - збільшення більшою чи меншою мірою, залежно від виду змін, що вносяться, часу розробки проекту, від лагодження програми, якості виконуваних робіт. Неминуче страждає і точність розрахунку, оскільки вона безпосередньо пов'язана з логікою побудови не лише основного коду програми, але і усіх структурних складових, що до неї входять.

Розрахунки, проведені по двох методиках, виявилися ідентичними і на перший погляд переваги розрахунку на основі класів не відчувається. Адже були отримані одні і ті ж результати, графічні залежності. Але вони представляють лише кінцевий результат, і виробляти оцінку за результатами розрахунку - логічно невірно і необґрунтовано. Різниця стає помітною в перспективі розвитку, вдосконаленні методики розрахунку, внесенні змін і поправок до вже існуючого програмного коду. Так, для будь-яких змін при структурному підході доводиться кожного разу не лише складати нові підпрограми, але і піклуватися про порядок їх виклику, узгодженні із вже існуючими змінними і так далі. При об'єктному підході досить лише створити новий об'єкт і додати в нього тільки ті дані, які зазнали зміни. Причому, на відміну від структурного підходу, не виникає складнощів в узгодженні змінних. Змінні в класах можуть впливати на змінні, що стоять як вище, так і нижче в дереві класів.

Доля структурного програмування - тільки низхідний напрям. Наприклад, конструктивно потрібно було застосувати в конструкції ЕТМП електромагнітний екран, розташований в повітряному проміжку. Наші дії наступні: створюємо похідний клас на базі вже існуючих з додаванням властивостей екранованої машини. От і все! Немає необхідності впроваджуватися в лістинги програмного коду, шукати те місце, куди необхідно внести нові дані і підпрограми. І ще не факт, що нові змінні не увійдуть до конфлікту із вже існуючими, і чи стануть вони доступні для функцій, розміщених в дереві вище або нижче. Як бачимо, проектування при об'єктно-орієнтованому підході стає дуже гнучким і таким, що динамічно розвивається. Якщо ж виникає необхідність внести істотніші зміни, змінити методику розрахунку цілком,

то переваги об'єктно-орієнтованого підходу і спадкоємства стають ще очевиднішими.

При структурному програмуванні необхідно наново переписувати усю програму. При класовому представленні справа йде інакше. Класи представляють не просто розрахунок, але і конструкцію машини. Навіть якщо поміняється методика, тіло машини: сталь, конструкція, обмотка, пази залишаються колишніми. Адже за допомогою такого "набору" можна створювати не лише різновиди вже існуючих машин (асинхронні, синхронні, постійного струму, трансформатори), але і нові, використовуючи методи генетичних комбінацій. Отже, наші дії зводяться до наступного: шляхом спадкоємства породжуємо новий клас, наділяємо його тільки тими змінами, які необхідно внести, а інші функціональні характеристики наслідуємо. Ось так проявляється уся сила спадкоємства!

6.3 Клас асинхронного двигуна з короткозамкненим ротором

На прикладі асинхронного двигуна з короткозамкненим ротором покажемо формування його класової організації і підготуємо створений клас до рішення завдань проектування за допомогою створення функціональних модулів.

Опис класів має бути виконаний в заголовному файлі, наприклад, "motor.h", а реалізація функцій класу - у файлі "motor.cpp".

Клас статора. У клас статора включаються змінні і константи, пов'язані з розрахунком геометрії і магнітної системи статора. Як дружні оголошені класи "Паз" і "Ротор".

Теплова система представлена функціональним модулем "Thermo". Динаміка пускових характеристик реалізується у функціональному модулі "Pusk". Також зарезервований функціональний модуль (прихований під коментарем), що реалізовує процеси гідродинаміки і механічного опору обертанню "Hydro".

```
//Клас статора
class Stator
{
friend class Paz;
friend class Rotor;
protected:
//Енергетичні змінні
double U1, f1, P1, P2, Pe, Pe1, Pem, Psum, Pst, Ppul2, Pdob, Pmeh;
double I1, Im, A, n1, cosFi1, kpd, p1_50, ppov2, cosFis, P1s, I0;
double Mn, Mem, Mkr, Mp, s, sn, s1;
double W1, W1_old, A_old, J1, J1_old;
int s1i;
//Магнітне коло
double Bd, Bd_old, Ba, Bzmin, Bzmax, Bzmid, Bz, Hz, Hzmid, Hzmin, Hzmax;
double Ha, Fd, Fa, Fz, Fm, Fs, Fi, Fim, Bzmax_old, Ba_old;
//Геометричні розміри
double delta, ld, ldmin, ldmax, tau, ha, La, bkt, Llob, Lv;
//Параметри
double r1, x1, Rd, Rm, Xm, xm, Zm, zbaz, x1oe;
//Коефіцієнти
double mue, mu0, ke, kdd, kc, kd, gamma, kdelta, p, dp, pi;
double Z1, Z1min, Z1max, betta, km, betta02, omega;
double lamdaP, lamdaL, lamdaD, kbt, kbt1, kbt11, k1, k11, c, d, ksi, cd, cdi;
double d_d, ma, mz, sigma;
double rad;
int ZZ, ZR, nmin, nci, Z1index;
//Постійні
int proj_type;  //0  тип проги : 0 - перевірка; 1 - проектування
int rotor_type; //0  тип МР : 0 - звичайний, 1 - обернений
int slot_type;  //- 1 тип паза : 0 - трапецеїдальний
int wind_type;  //0  тип обмотки : 0 - одношарова
int wire_type;  //0  тип дроту : 0 - круглий
int steel;      //0  марка стали: 0 - 2013; 1 - 2211 і 2312;2 – 2411
```

```
int steel_box;//0
int wire_box;//13
int lob_iz; // 0 - лобові частини не ізольовані стрічкою
bool he_flag;//0
int schem; //схема соед. обмоток: зірка = 0/трикутник=1
double gamma115, ro1, kB, m, gammaSt, kda, kdz;
double gammaCu, kf, km_old;
int ismax;
//Масиви
int SteelBox[5];
double cosFir[50], P1r[50], Pe1r[50], Pemr[50], Pmehr[50], Pdobr[50],
SumPr[50], P2r[50];
double kpdr[50], Per[50], Mpp[50];
double I1_mod[50], I0_mod[50];
double dimZ1[20];
//Функції-члени
// розрахунок напруженостей для прямокутних пазів
void GetH(double Bzmax, double Bzmin, double Bzmid, double Ba, int steel);
 // розрахунок напруженостей для трапецієвидних пазів
void GetH(double Bzmax, double Ba, int steel);
void Thermo();// тепловий розрахунок
//void Hydro();// гідродинамічний розрахунок
void Pusk();// розрахунок пуску ЕТМП
public:
//Допоміжні функціональні модулі
int GetZindex();
double GetZ1(int);
int GetInt();
double GetW1old();
void Init(double, double, double, double, int);

Stator();  // конструктор
~Stator(); // деструктор
};
```

Клас ротора. Ротор представлений у вигляді класу "Ротор" з дружнім класом "Паз". Структура класу очевидна по коментарях.

```
class Rotor
{
protected:
//Енергетичні змінні
double Perot, I21, f2, n2, Ppov2, s, skr, s2, s_pr, nr, I21_old;
//Магнітне коло
double Br, Hr, B02, Brot, Bpul2, F2;
//Геометричні розміри
double D2, L2;
//Параметри
double r2, x2, z2x, r20;
//Коефіцієнти
double kv, kdv, mrot, ds;
//Постійні
double gammaSt_r;
//Масиви
double sr[50], I21p[50], z2_re[50], z2_im[50], I2_mod[50], Fi2x[50];
public:
void Init(double, double);
Rotor();  // конструктор класу
};
```

Клас паза. Реалізація класу паза, що є дружнім для класів статору і ротору, включає шість видів трапецієвидних пазів і три види прямокутних пазів. У реалізації класу "Паз" потрібні вхідні дані, що містяться в класах "Статор" і "Ротор". Для цього в тіло класу "Паз" включені покажчики на об'єкти класів "Статор" і "Ротор".

```
//Клас паза
class Paz
{
protected:
//Геометричні розміри
double bzmin, bzmax, bz, bzmid, bsh, hsh, bp, dbp, dhp, bz1, b1, b2, h0;
double h1, h2, h3, hz1, b11, b21, h31, h21, hp, hp1, hk, t1min, t1max, t1;
double t2, bp_old, be_old, ha_old, hp_old, ddh, ddb, h1z;
```

```
//Обмотувальні дані
double biz, Sp, Siz, Sp1, diz, del, def, apr, bpr, apriz, bpriz, bpiz, qef, qel;
double qpiz, Spr, apiz, diz1, mwire, lmid, B, dizp;
double del_old, qel_old, qef_old, nel_old;
//Коефіцієнти
double Up, Up1, a, da, q, kob, ku, kr, kzmin, kzmax, kz, nel, kV, kL, amin,
ncf, ddz;
//Постійні
double S, mm, KL;
int ST0;//0-гільзова ізоляція; 1-термореактивна ізоляція
int aindex, i, j;
//Масиви
CString WireMark[20];
double dima[10], dimUp[10], dimUpm[10];
double dimkr[10], dimde[60], dimdiz[60], dimqt[60], dimapr1[60], di-
mapr2[60], dimbpr1[60], dimbpr2[60];
double dimLa[50], dimLb[50], dimSa[50], dimSb[50], p1A[10], p1B[10],
p23[10];
double dimqSP2[40][40], dimqSL[40][40], dimqSS[40][40];
double dimqSP1[40][40];
//Функції-члени
void paz_trap1(double, double, double);// розрахунок трапец. пазу 1
void paz_trap2(double, double, double);// розрахунок трапец. пазу 2
void paz_trap3(double, double, double);// розрахунок трапец. пазу 3
void paz_trap4(double, double, double);// розрахунок трапец. пазу 4
// розрахунок трапец. пазу 5
void paz_trap5(double, double, double, double, double);
// розрахунок трапец. пазу 6
void paz_trap6(double, double, double, double, double);
void pazTZ();// розрахунок заповнення трапецевидного паза
// розрахунок паза прямокутного відкритого з гільзовою ізоляцією
void paz_1A(double, double, int, double);
//розрахунок пазу прямокутного відкритого з термореактивною
ізоляцією
void paz_1B(double, double, int, double);
// розрахунок пазу прямокутного напівзакритого
void paz2_3(double, double, int, double);
// розрахунок пазу прямокутного відкритого з гільзовою ізоляцією 2
void paz_1AZ();
// розрахунок пазу прямокутного відкритого з термореакт. изоляц. 2
void paz_1BZ();
void paz2_3Z();// розрахунок паза прямокутного напівзакритого 2
void pazPZKR();// розрахунок заповнення прямоуг. паза круглим дротом
```

```
void paz_T(double, int);// попередній розрахунок паза прямокутного
public:
Stator* stat; //покажчик на об'єкт класу "Статор"
Rotor* rot; //покажчик на об'єкт класу "Ротор"

Paz();// конструктор класу
~Paz();// деструктор класу
};
```

Клас асинхронного двигуна з короткозамкненим ротором. Фінальний клас проекту є спадкоємцем створених раніше класів "Статор", "Паз", "Ротор" і формує базовий інтерфейс класу. Тіло класу містить функціональні модулі, які раніше, при розгляді процедурного підходу проектування в розділі 5, були представлені у вигляді окремих функцій. Тепер вони є засобом доступу до даних класу і реалізацією розрахункових методів. На додаток до приведених в даному розділі переваг об'єктно-орієнтованого проектування потрібно віднести також і той факт, що тепер усі розрахункові функції не треба оголошувати "зовнішніми" по відношенню до розрахункової програми, що підвищує ефективність використання пам'яті ЕОМ, захист даних і виключає помилки доступу у виклику розрахункових функцій. Оскільки звернення до класу виконується за допомогою функціональних модулів, а не по окремих змінних, як в процедурній програмі, усі поліпшення класу не торкнуться інтерфейсу користувача і використовуваних їм даних.

```
class AMotor: virtual public Stator, Paz, Rotor
{
protected:
double D, Dv, L0, B0;
public:
//Функції-члени
int GetSteel(); //
double GetData();
```

```
double GetLdmin();
double GetLdmax();
void SetJ1(double);
int GetWireType();
int AnalizeQ(double);
void Init(double, double, double, double, int, double, double, double, dou-
ble, double, double);
void Geom(double);// розрахунок основних геом. розмірів
void Zubc(int);// розрахунок зубцевої зони
void Magn(int);// розрахунок струмів, магнітної системи
void ParamS();// розрахунок параметрів статора
void ParamR();// розрахунок параметрів масивного ротора
void Wind(double, int);// розрахунок обмотувального дроту
void RecalcPaz(int);// пересчет статора при зміні паза
void RecalcWind();// перерахунок даних статора обмотки
void Char();// розрахунок робочих і пускових характеристик
void DataOutput();//выод расчитанных даних у файл
void Graph();// побудова графіків
void Paz_PR();//попередній розрахунок прямокутного пазу
void PazPR_End(double, double, double, double, int);
void Paz_TR(double, double, double, int, double, double, double, double);
//попередній розрахунок трапец. пазу
double SetTPaz(int, double);

AMotor();// конструктор класу
~AMotor();// деструкція класу
};
```

Реалізація класу в cpp-файлі наведена, зважаючи на великий розмір, в додатку Е.

СПИСОК РЕКОМЕНДОВАНОЇ ЛІТЕРАТУРИ

1. Р. Лафоре Объектно-ориентированное программирование в С++. Санкт-Петербург: Питер, 2004. – 923 с.

2. Арчер Т., Уайтчепел Э. Visual Studio.NET. Библия пользователя. – М.: Издательский дом «Вильямс», 2005. – 1216 с.

3. Полещук Н.Н., Лоскутов П.В. AutoLISP и Visual LISP в среде AutoCAD. – СПб.: БХВ-Петербург, 2006. – 960 с.

4. Зуев С.А., Полещук Н.Н. САПР на базе AutoCad – как это делается. – СПб.: БХВ-Петербург, 2004. – 1168 с.

5. Полещук Н.Н. Самоучитель AutoCad 2007/ Н.Н. Полещук, В.А. Савельева. – СПб.: БХВ-Петербург, 2006. – 624 с.

6. Плюгин В.Е. Курс лекций «САПР электромеханических устройств». Алчевск, 2009. – 120 с.

7. Бахвалов Н.С., Жидков Н.П., Кобельков Г.М. Численные методы. М.: Бином. Лаборатория знаний, 2003. – 632 с.

8. Реклейтис Г., Рейвиндан А., Рэгсдел К. Оптимизация в технике: В 2-х кн. Кн. 1. Пер. с англ. – М.: Мир, 1986. – 351 с.

9. Проектирование электрических машин: Учебник для студентов электромехан. и электроэнерг.спец.вузов/ И.П.Копылов, Б.К. Клоков, В.П. Морозкин, Б.Ф. Токарев. Под ред. И.П. Копылова. – 3-е изд., перераб. и доп. – М.: Высшая школа, 2002.- 757 с.

10. Вольдек А.И. Электрические машины. Л.: Энергия, 1978. - 832 с.

11. Заблодський М.М. Теплові розрахунки електротехнічних устаткувань. Навчальний посібник. – Алчевськ: ДГМІ, ВПЦ «Ладо», 2003. – 290 с.

ДОДАТОК А

ЛІСТИНГ ФУНКЦІЇ ЗБЕРЕЖЕННЯ ПРОЕКТУ У ФАЙЛ

```
void DataOutput(int flag, char* f)
{
ofstream df(f);
switch (flag)
{
case 0:
        df << "первинний неоптимізований розрахунок".<< "\n";
        break;
case 1:
        df << "розрахунок з оптимізацією геометрії і параметрів" << "\n";
        break;
case 2:
        df << "розрахунок з оптимізацією Up і Z1" << "\n";
        break;
case 3:
        df << "варіант оптимізації за найменшою вартістю" << "\n";
        break;
}
df << "Варіант №: --" << "\n";
df << "Рн = " << Pn/1000 << " кВт" << "\n";
df << "Uн = " << U << " В" << "\n";
df << "f1н = " << f1 << " Гц" << "\n";
df << "nн = " << n << " про/мін" << "\n";
df << "1. Розміри статора" << "\n";
df << "2. Обмотувальні дані статора" << "\n";
df << "3. Розміри ротора" << "\n";
df << "4. Параметри обмотки статора" << "\n";
df << "5. Параметри обмотки ротора" << "\n";
df << "6. Магнітне коло" << "\n";
df << "7. Енергетичні показники" << "\n";
df << "8. Втрати і робочі характеристики" << "\n";
df << "9. Пускові характеристики" << "\n";
df << "10. Нагрів і вентиляція" << "\n";
df << "11. Механічний розрахунок" << "\n";
df << "12. Вартість" << "\n";
} //Кінець функції DataOutput()
```

ДОДАТОК Б

ФУНКЦІЯ АВТОМАТИЗОВАНОГО РОЗРАХУНКУ 1

```
//Автоматичний розрахунок в циклі оптимізації
//Up = const, Z1 = const, Z2 = const
//Da, D = const
// ld, delta, Bd = var
void auto_func1()
{
//Обмеження виходу за допустимі межі
if (ld > ld_max)
{
        //ld = 1.1*ld_max;
        KP1 = 0;
}
else if (ld < ld_min)
{
        //ld = 0.9*ld_min;
        KP1 = 0;
}
else
        KP1 = 1;
if (delta > delta_max)
{
        //delta = 1.1*delta_max;
        KP2 = 0;
}
if (delta < delta_min)
{
        //delta = 0.9*delta_min;
        KP2 = 0;
}
else
        KP2 = 1;
lamda = ld/tau;
//Розрахунок зубцевої зони
Bd = p*Fi/(D*ld);
//Розрахунок ротора
D2 = D - delta;
```

```
l2 = ld;
t2 = pi*D2/Z2;
Dk = D2 - bk;
//Розрахунок струму намагнічування
Bz1 = Bd*t1/(bz1*kc);
if (Bz1 < 0.4)
        Bz1 = 0.4;
if (Bz1 > 2.39)
        Bz1 = 2.39;
Bz2 = Bd*t2/(bz2*kc);
if (Bz2 < 0.4)
        Bz2 = 0.4;
if (Bz2 > 2.39)
        Bz2 = 2.39;
Ba = Fi/(2*ha*ld*kc);
if (Ba < 0.4)
        Ba = 0.4;
if (Ba > 2.09)
        Ba = 2.09;
hj1(3.2*p);
Bj = Fi/(2*hj1*ld*kc);
if (Bj < 0.4)
        Bj = 0.4;
if (Bj > 2.09)
        Bj = 2.09;
gamma(5 + bsh/delta);
kd = t1/(t1 - gamma*delta);
Fd = 1.59e06*Bd*kd*delta;
Hz1 = spline(Bz1, "St_Bz.dat", "St 2013Z.dat");
Fz1 = 2*hz1*Hz1;
Hz2 = spline(Bz2, "St_Bz.dat", "St 2013Z.dat");
Fz2 = 2*hz2*Hz2;
kz = 1 + (Fz1 + Fz2)/Fd;
hj = (D2 - Dj)/2 - hp2;
Lj = pi*(Dj + hj)/dp;
Ha = spline(Ba, "St_Ba.dat", "St 2013A.dat");
Hj = spline(Bj, "St_Ba.dat", "St 2013A.dat");
Fa = La*Ha;
Fj = Lj*Hj;
Fc = Fd + Fz1 + Fz2 + Fa + Fj;
km = Fc/Fd;
Im = p*Fc/(0.9*m*W1*kob);
Imo = Im/I1nc;
```

```
//Параметри робочого режиму
lp1 = ld;
bkt = pi*(D + hp)*betta1/dp;
ll1 = Kl*bkt + 2*B;
lmid1 = 2*(lp1 + ll1);
L1 = lmid1*W1;
lv = Kv*bkt + B;
r1 = p115*L1/(qef*a);
r1o = r1*I1n/U;
rc = p115a*l2/qc2;
rk = p115a*pi*Dk/(Z2*qk);
r2 = rc + 2*rk/(ddr*ddr);
r21 = r2*4*m*W1*W1*kob*kob/Z2;
r21o = r21*I1n/U;
//////////////////////
Lh1 = (b2 - bsh)/2;
lamda_p = h1*kbetta/(3*b2) + kbetta1*(3*Lh1/(b2 + 2*bsh) + hsh/bsh);
lamda_l = 0.34*q*(ll1 - 0.64*tau)/ld;
ksi = 2*ksk*kbetta - kob*kob*t2*t2(t1*t1);
lamda_d = t1*ksi/(12*delta*kd);
x1 = 15.8*f1*W1*W1*ld(100*100*100*p*q);
x1o = x1*I1n/U;
//////////////////////
Lh1r = hp2 - hsh2 - hshr1 - 0.1*b2r;
lamda_p2 = Lh1r(3*b1r) + 0.66 - bsh2/(2*b1r) + hsh2/bsh2 +
1.12*hshr1*1e06/I2;
lamda_l2 = 2.3*Dk*log(Z2*ld*ddr*ddr);
ksi2 = 1 + 0.2(pi*p/Z2) - ddz/(1 -(p/Z2));
lamda_d2 = t2*ksi2/(12*delta*kd);
x2 = 7.9*f1*ld*(lamda_p2 + lamda_l2 + lamda_d2)*1e-06;
x21 = x2*4*m*W1*W1*kob*kob/Z2;
x21o = x21*I1n/U;
//Розрахунок втрат
ha = 0.5*(Da - D) - hp;
ma = pi*(Da - ha)*ha*ld*kc*gammaSt;
mz1 = hz1*bz1*Z1*ld*kc*gammaSt;
Pst_main = 2.6*(1.6*Ba*Ba*ma + 1.8*Bz1*Bz1*mz1);
B02 = betta02*kd*Bd;
ppov2 = 0.5*k02*pow((Z1*n/10000)1.5)*B02*B02*t1*t1*1e06;
Ppov2 = ppov2*(t2 - bsh2)*Z2*ld;
Bpul2 = gamma*delta*Bz2/(2*t2);
mz2 = Z2*hz2*bz2*ld*kc*gammaSt;
Ppul2 = 0.11*pow((Z1*n*Bpul2/1000)2)*mz2;
```

```
Pst_dob = Ppov2 + Ppul2;
Pst = Pst_main + Pst_dob;
Pmeh = Kt*n*n*Da*Da*Da*Da/100;
Pdob = 0.005*Pn/kpd;
Pe10 = 3*Im*Im*r1;
Ixxa(m*U);
I0 = sqrt(Ixxa*Ixxa + Im*Im);
cosFi0 = Ixxa/Im;
//Розрахунок робочих характеристик
r12 = Pst_main/(m*Im*Im);
x12 = U/Im - x1;
c1 = 1 + x1/x12;
I0a(3*U);
aa1 = c1*c1;
bb1 = 0;
aa = c1*r1;
bb = c1*(x1 + c1*x21);
Pst_meh = Pst + Pmeh;

sn = r21o;
s = sn;
R = aa + aa1*r21/s;
X = bb + bb1*r21/s;
Z = sqrt(R*R + X*X);
I211 = U/Z;
I1a = I0a + I211*R/Z;
I1r = Im + I211*X/Z;
I1 = sqrt(I1a*I1a + I1r*I1r);
I21 = c1*I211;
P1r = 3*U*I1a;
Pe1 = 3*I1*I1*r21;
Pe2 = 3*I21*I21*r21;
Pd = Pdob*(I1*I1/(I1n*I1n));
Psum = Pst_meh + Pe1 + Pe2 + Pd;
P2 = P1r - Psum;
kpdr = 1 - Psum/P1r;
cosFir = I1a/I1;

kpdn = kpdr;
cosFin = cosFir;
I1nc = I1;
P2n = P2;
I21n = I21;
```

```
Mn = p*m*I21n*I21n*r21*r21/(2*pi*f1*sn);

//Розрахунок пускових характеристик
ds = 0.01;
s = sn*0.2;
imaxp((1 - s)/ds);
for (int i = 0; i <= imaxp+1; i++)
{
        ksip = 63.61*hc*sqrt(s);
        if (ksip < 1)
                fip = ksip/10;
        else if (ksip > 3)
                fip = ksip - 1;
        else
                fip = spline(ksip, "ksip.dat", "fip.dat");
        hr = hc/(1 + fip);
        br = b1r -(hr - 0.5*b1r)/h1r;
        qr = pi*b1r*b1r/8 + 0.5(hr - 0.5*b1r);
        kr = qc2/qr;
        Kr = 1 + rc*(kr - 1)/r2;
        r2ksi = Kr*r21;
        if (ksip > 4)
                kdp = 3*ksip/2;
        else
                kdp = spline(ksip, "ksipf.dat", "kdp.dat");
lamda_p2ksi = (Lh1r*pow(3*b1r) + 0.66 - 0.5*bsh2/b1r)*kdp + hsh2/bsh2
+ 1.12*hshr1*1e06/(6.5*I2);
Kx(lamda_p2 +lamda_l2 + lamda_d2);
x21ksi = x21*Kx;
I21p = U/sqrt(pow((r1 + r2ksi/s)2) + pow((x1 + x21ksi)2));
CN = 0.64 + 2.5*sqrt(delta/(t1 + t2));
I1nac = knac*I21p;
Fpmid = 0.7*I1nac*Up*(kbetta1 + ku*kob*Z1/Z2)/a;
Bfd = Fpmid*1e-06/(1.6*delta*CN);
xd = spline(Bfd, "Bfd.dat", "xd.dat");
c1p(1 - xd);
h11p = hp - hsh - h1;
dlamda_p1 = (hsh + 0.58*h11p)*c1p/(bsh*(c1p + 1.5*bsh));
lamda_p1nac = lamda_p - dlamda_p1;
lamda_d1nac = lamda_d*xd;
x1nac = x1(lamda_p + lamda_l + lamda_d);
c2 = (t2 - bsh2)*c2(1 - xd);
dlamda_p2 = hsh2*c2/(bsh2*(bsh2 + c2));
```

```
lamda_p2nac = lamda_p2ksi - dlamda_p2;
lamda_d2nac = lamda_d2*xd;
x21nac = x21(lamda_p2 +lamda_l2 + lamda_d2);
x12p = x12*Fc/Fd;
c1pnac = 1 + x1nac/x12p;
app = r1 + c1pnac*r2ksi/s;
bpp = x1nac + c1pnac*x21nac;

I21p2 = U/sqrt(app*app + bpp*bpp);
I1p = I21p2*sqrt(c1pnac*x12p);
Ipo = I1p/I1nc;
Mpo = I21p2*I21p2*Kr*sn/(I21n*I21n*s);
dimMp[i] = Mpo;
dimI1p[i] = Ipo;
dimsp[i] = s;

s += ds;
}//кінець циклу пускових характеристик

I1nac_s1 = I1nac;
I1ksi_s1 = I1p;
Ipo_s1 = Ipo;
Mpo_s1 = Mpo;
//skr = r21/(x1nac/c1pnac + x21ksi);
int ikr = fmax(dimMp, imaxp+1);
Mmaxo = dimMp[ikr];
skr = dimsp[ikr];

//Тепловий розрахунок
Pe1n = m*I1n*I1n*r1;
Pe2n = m*I21n*I21n*r21;
Psumn = Pst_meh + Pe1n + Pe2n + Pdob;
Pep1 = kiz*Pe1n*2*ld/lmid1;
dtpov1 = K(pi*D*ld*alfa1);
Pp1 = 2*hp + b1 + b2;
  dtizp = Pep1(Z1*Pp1*ld);
Pelob1 = kiz*Pe1n*2*ll1/lmid1;
PPlob = Pp1;
bizlob = 0;
dtizlob = Pelob1(2*Z1*PPlob*ll1);
dtpovlob = K*Pelob1/(2*pi*D*lv*alfa1);
dt11 = (dtpov1 + dtizp)*2*ld/lmid1 + (dtizlob + dtpovlob)*2*ll1/lmid1;
Psum1 = Psumn(Pe1n + Pe2n);
```

```
Psum1v = Psum1 -(Pep1 + Pst_main) - 0.9*Pmeh;
Sk(ld + 2*lv);
dtv = Psum1v/(Sk*alfa_v);
dt1 = dt11 + dtv;
kmv = mm*sqrt(n*Da/100);
Qv = kmv*Psum1v/(1100*dtv);
Qv1 = 0.6*Da*Da*Da*n/100;

price(); //розрахунок приведеної вартості

//Вибір критерію оптимуму
opt = 0;
switch (krit)
{
case 0: //Ip/In
        opt = Ipo;
        break;
case 1: //Mp/Mn
        opt = 1/Mpo;
        break;
case 2: // Mmax/Mn
        opt = 1/Mmaxo;
        break;
case 3: // kpd
        opt = 1/kpdn;
        break;
case 4: // cosFi
        opt = 1/cosFin;
        break;
case 5:// price
        opt = SZ;
        break;
}

}//кінець функції auto_func()
```

ДОДАТОК В

ФУНКЦІЯ АВТОМАТИЗОВАНОГО РОЗРАХУНКУ 2

```
//Автоматичний розрахунок в циклі оптимізації
//Виконується після функції auto_func()
//Up = var, Z1 = var, Z2 = var
//Da, D, delta, Bd = const
void auto_func()
{
//Розрахунок зубцевої зони
q = Z1/(dp*m);
t1 = pi*D/(dp*m*q);
A = I1nc*Z1*Up/(pi*D);
W1 = Up*Z1/(2*a*m);
Fi = ke*U/(4*kB*kob*f1*W1);
ld = p*Fi/(Bd*D);
bz1 = Bd*t1/(Bz1n*kc);
ha = Fi/(2*Ban*ld*kc);
hp = (Da - D)/2 - ha;
b1 = pi*(D + 2*hp)/Z1 - bz1;
b2(Z1 - pi);
h1 = hp - (hsh + (b2 - bsh)/2);
b11 = b1 - dbp;
b21 = b2 - dbp;
h11 = h1 - dhp;
Siz = biz*(2*hp + b1 + b2);
Sp1 = (b11 + b21)*h11/2 - Siz - Spr;
kzp = diz*diz*Up*nel/Sp1;
J1 = I1n/(a*qef);
//Розрахунок ротора
l2 = ld;
t2 = pi*D2/Z2;
nui = 2*m*W1*kob/Z2;
I2 = ki*I1n*nui;
qc2 = I2/J2n;
bz2 = Bd*t2*ld/(Bz2*l2*kc);
b1r(pi + Z2);
b2r = sqrt((b1r*b1r*(Z2/pi + pi/2) - 4*qc2)/(Z2/pi - pi/2));
h1r = (b1r - b2r)*Z2/(2*pi);
hp2 = hshr1 + hsh2 + b1r/2 + h1r + b2r/2;
```

```
qc2 = pi*(b1r*b1r + b2r*b2r)/8 + 0.5*h1r*(b1r + b2r);
J2 = I2/qc2;
Jk = 0.85*J2;
ddr = 2*sin(pi*p/Z2);
Ik = I2/ddr;
qk = Ik/Jk;
bk = 1.25*hp2;
ak = qk/bk;
Dk = D2 - bk;

//Розрахунок струму намагнічування
Bz1 = Bd*t1/(bz1*kc);
if (Bz1 < 0.4)
        Bz1 = 0.4;
if (Bz1 > 2.39)
        Bz1 = 2.39;
Bz2 = Bd*t2/(bz2*kc);
if (Bz2 < 0.4)
        Bz2 = 0.4;
if (Bz2 > 2.39)
        Bz2 = 2.39;
Ba = Fi/(2*ha*ld*kc);
if (Ba < 0.4)
        Ba = 0.4;
if (Ba > 2.09)
        Ba = 2.09;
hj1(3.2*p);
Bj = Fi/(2*hj1*ld*kc);
if (Bj < 0.4)
        Bj = 0.4;
if (Bj > 2.09)
        Bj = 2.09;

gamma(5 + bsh/delta);
kd = t1/(t1 - gamma*delta);
Fd = 1.59e06*Bd*kd*delta;
Hz1 = spline(Bz1, "St_Bz.dat", "St 2013Z.dat");
if (Bz1 < 0.4)
        Hz1 = 100;
hz1 = hp;
Fz1 = 2*hz1*Hz1;
Hz2 = spline(Bz2, "St_Bz.dat", "St 2013Z.dat");
if (Bz2 < 0.4)
```

```
        Hz2 = 100;
hz2 = hp2 - 0.1*b2r;
Fz2 = 2*hz2*Hz2;
kz = 1 + (Fz1 + Fz2)/Fd;
La = pi*(Da - ha)/dp;
hj = (D2 - Dj)/2 - hp2;
Lj = pi*(Dj + hj)/dp;
Ha = spline(Ba, "St_Ba.dat", "St 2013A.dat");
Hj = spline(Bj, "St_Ba.dat", "St 2013A.dat");
if (Ba < 0.4)
        Ha = 45;
if (Bj < 0.4)
        Hj = 45;
Fa = La*Ha;
Fj = Lj*Hj;
Fc = Fd + Fz1 + Fz2 + Fa + Fj;
km = Fc/Fd;
Im = p*Fc/(0.9*m*W1*kob);
Imo = Im/I1nc;

//Параметри робочого режиму
lp1 = ld;
bkt = pi*(D + hp)*betta1/dp;
ll1 = Kl*bkt + 2*B;
lmid1 = 2*(lp1 + ll1);
L1 = lmid1*W1;
lv = Kv*bkt + B;
r1 = p115*L1/(qef*a);
r1o = r1*I1n/U;
rc = p115a*l2/qc2;
rk = p115a*pi*Dk/(Z2*qk);
r2 = rc + 2*rk/(ddr*ddr);
r21 = r2*4*m*W1*W1*kob*kob/Z2;
r21o = r21*I1n/U;
//////////////////////
Lh1 = (b2 - bsh)/2;
lamda_p = h1*kbetta/(3*b2) + kbetta1*(3*Lh1/(b2 + 2*bsh) + hsh/bsh);
lamda_l = 0.34*q*(ll1 - 0.64*tau)/ld;
ksi = 2*ksk*kbetta - kob*kob*t2*t2(t1*t1);
lamda_d = t1*ksi/(12*delta*kd);
x1 = 15.8*f1*W1*W1*ld(100*100*100*p*q);
x1o = x1*I1n/U;
```

```
Lh1r = hp2 - hsh2 - hshr1 - 0.1*b2r;
lamda_p2 = Lh1r(3*b1r) + 0.66 - bsh2/(2*b1r) + hsh2/bsh2 +
1.12*hshr1*1e06/I2;
lamda_l2 = 2.3*Dk*log(Z2*ld*ddr*ddr);
ksi2 = 1 + 0.2(pi*p/Z2) - ddz/(1 -(p/Z2));
lamda_d2 = t2*ksi2/(12*delta*kd);
x2 = 7.9*f1*ld*(lamda_p2 + lamda_l2 + lamda_d2)*1e-06;
x21 = x2*4*m*W1*W1*kob*kob/Z2;
x21o = x21*I1n/U;
//Розрахунок втрат
ha = 0.5*(Da - D) - hp;
ma = pi*(Da - ha)*ha*ld*kc*gammaSt;
mz1 = hz1*bz1*Z1*ld*kc*gammaSt;
Pst_main = 2.6*(1.6*Ba*Ba*ma + 1.8*Bz1*Bz1*mz1);
B02 = betta02*kd*Bd;
ppov2 = 0.5*k02*pow((Z1*n/10000)1.5)*B02*B02*t1*t1*1e06;
Ppov2 = ppov2*(t2 - bsh2)*Z2*ld;
Bpul2 = gamma*delta*Bz2/(2*t2);
mz2 = Z2*hz2*bz2*ld*kc*gammaSt;
Ppul2 = 0.11*pow((Z1*n*Bpul2/1000)2)*mz2;
Pst_dob = Ppov2 + Ppul2;
Pst = Pst_main + Pst_dob;
Pmeh = Kt*n*n*Da*Da*Da*Da/100;
Pdob = 0.005*Pn/kpd;
Pe10 = 3*Im*Im*r1;
Ixxa(m*U);
I0 = sqrt(Ixxa*Ixxa + Im*Im);
cosFi0 = Ixxa/Im;

//Розрахунок робочих характеристик
r12 = Pst_main/(m*Im*Im);
x12 = U/Im - x1;
c1 = 1 + x1/x12;
I0a(3*U);
aa1 = c1*c1;
bb1 = 0;
aa = c1*r1;
bb = c1*(x1 + c1*x21);
Pst_meh = Pst + Pmeh;

sn = r21o;
s = sn;
R = aa + aa1*r21/s;
```

```
X = bb + bb1*r21/s;
Z = sqrt(R*R + X*X);
I211 = U/Z;
I1a = I0a + I211*R/Z;
I1r = Im + I211*X/Z;
I1 = sqrt(I1a*I1a + I1r*I1r);
I21 = c1*I211;
P1r = 3*U*I1a;
Pe1 = 3*I1*I1*r21;
Pe2 = 3*I21*I21*r21;
Pd = Pdob*(I1*I1/(I1n*I1n));
Psum = Pst_meh + Pe1 + Pe2 + Pd;
P2 = P1r - Psum;
kpdr = 1 - Psum/P1r;
cosFir = I1a/I1;

kpdn = kpdr;
cosFin = cosFir;
I1nc = I1;
P2n = P2;
I21n = I21;
Mn = p*m*I21n*I21n*r21*r21/(2*pi*f1*sn);
//Розрахунок пускових характеристик
hc = hp2 - hsh2 - hshr1;
ds = 0.01;
s = sn*0.2;
imaxp((1 - s)/ds);
for (int i = 0; i <= imaxp+1; i++)
{
        ksip = 63.61*hc*sqrt(s);
        if (ksip < 1)
                fip = ksip/10;
        else if (ksip > 3)
                fip = ksip - 1;
        else
                fip = spline(ksip, "ksip.dat", "fip.dat");
        hr = hc/(1 + fip);
        br = b1r -(hr - 0.5*b1r)/h1r;
        qr = pi*b1r*b1r/8 + 0.5(hr - 0.5*b1r);
        kr = qc2/qr;
        Kr = 1 + rc*(kr - 1)/r2;
        r2ksi = Kr*r21;
        if (ksip > 4)
```

```
            kdp = 3*ksip/2;
        else
            kdp = spline(ksip, "ksipf.dat", "kdp.dat");
        lamda_p2ksi = (Lh1r*pow(3*b1r) + 0.66 - 0.5*bsh2/b1r)*kdp +
        hsh2/bsh2 + 1.12*hshr1*1e06/(6.5*I2);
        Kx(lamda_p2 +lamda_l2 + lamda_d2);
        x21ksi = x21*Kx;
        I21p = U/sqrt(pow((r1 + r2ksi/s)2) + pow((x1 + x21ksi)2));
        CN = 0.64 + 2.5*sqrt(delta/(t1 + t2));
        I1nac = knac*I21p;
        Fpmid = 0.7*I1nac*Up*(kbetta1 + ku*kob*Z1/Z2)/a;
        Bfd = Fpmid*1e-06/(1.6*delta*CN);
        xd = spline(Bfd, "Bfd.dat", "xd.dat");
        c1p(1 - xd);
        h11p = hp - hsh - h1;
        dlamda_p1 = (hsh + 0.58*h11p)*c1p/(bsh*(c1p + 1.5*bsh));
        lamda_p1nac = lamda_p - dlamda_p1;
        lamda_d1nac = lamda_d*xd;
        x1nac = x1(lamda_p + lamda_l + lamda_d);
        c2 = (t2 - bsh2)*c2(1 - xd);
        dlamda_p2 = hsh2*c2/(bsh2*(bsh2 + c2));
        lamda_p2nac = lamda_p2ksi - dlamda_p2;
        lamda_d2nac = lamda_d2*xd;
        x21nac = x21(lamda_p2 +lamda_l2 + lamda_d2);
        x12p = x12*Fc/Fd;
        c1pnac = 1 + x1nac/x12p;
        app = r1 + c1pnac*r2ksi/s;
        bpp = x1nac + c1pnac*x21nac;
        I21p2 = U/sqrt(app*app + bpp*bpp);
        I1p = I21p2*sqrt(c1pnac*x12p);
        Ipo = I1p/I1nc;
        Mpo = I21p2*I21p2*Kr*sn/(I21nc*I21nc*s);

        dimMp[i] = Mpo;
        dimI1p[i] = Ipo;
        dimsp[i] = s;

        s += ds;
}//кінець циклу пускових характеристик

I1nac_s1 = I1nac;
I1ksi_s1 = I1p;
Ipo_s1 = Ipo;
```

```
Mpo_s1 = Mpo;
//skr = r21/(x1nac/c1pnac + x21ksi);
int ikr = fmax(dimMp, imaxp+1);
Mmaxo = dimMp[ikr];
skr = dimsp[ikr];

//Тепловий розрахунок
Pe1n = m*I1n*I1n*r1;
Pe2n = m*I21n*I21n*r21;
Psumn = Pst_meh + Pe1n + Pe2n + Pdob;

Pep1 = kiz*Pe1n*2*ld/lmid1;
dtpov1 = K(pi*D*ld*alfa1);
Pp1 = 2*hp + b1 + b2;
dtizp = Pep1(Z1*Pp1*ld);
Pelob1 = kiz*Pe1n*2*ll1/lmid1;
PPlob = Pp1;
bizlob = 0;
dtizlob = Pelob1(2*Z1*PPlob*ll1);
dtpovlob = K*Pelob1/(2*pi*D*lv*alfa1);
dt11 = (dtpov1 + dtizp)*2*ld/lmid1 + (dtizlob + dtpovlob)*2*ll1/lmid1;
Psum1 = Psumn(Pe1n + Pe2n);
Psum1v = Psum1 -(Pep1 + Pst_main) - 0.9*Pmeh;
Sk(ld + 2*lv);
dtv = Psum1v/(Sk*alfa_v);
dt1 = dt11 + dtv;
kmv = mm*sqrt(n*Da/100);
Qv = kmv*Psum1v/(1100*dtv);
Qv1 = 0.6*Da*Da*Da*n/100;
price(); //розрахунок приведеної вартості
//Критерій оптимуму
opt = 0;

switch (krit)
{
case 0: //Ip/In
        opt = Ipo;
        break;
case 1: //Mp/Mn
        opt = 1/Mpo;
        break;
case 2: // Mmax/Mn
        opt = 1/Mmaxo;
```

```
        break;
case 3: // kpd
        opt = 1/kpdn;
        break;
case 4: // cosFi
        opt = 1/cosFin;
        break;
case 5:// value
        opt = SZ;
        break;
}
}//кінець функції auto_func1()
```

ДОДАТОК Г

ПРИКЛАДИ НАЙБІЛЬШ ЗАСТОСОВНИХ АЛГОРИТМІВ

У додатку розглядаються прості алгоритми, найбільш часто використовувані в автоматизованих розрахунках. Це алгоритми, що дозволяють приймати рішення про значущість якій-небудь величині і її місця в множині, а також вирішувати прості завдання проектування.

Г.1 Генерація випадкового цілого числа

```
//Генерація випадкового числа в діапазоні від 0 до MAX
int MAX = 52;
int k = rand()%MAX;
```

Г.2 Пошук мінімального/максимального значення масиву

```
int max = dim[0];
int min = dim[0];
for (i = 1; i <= imax; i++) // imax - кількість циклів
{
if (dim[i] < min)
        min = dim[i];
    if(dim[i] > max)
        max = dim[i];
} //кінець циклу for
```

Г.3 Сортування масиву методом "бульбашки"

У даному методі сортування перший елемент масиву порівнюється по черзі з іншими елементами (починаючи з другого). Якщо він більший, ніж який-небудь з елементів масиву, то ці елементи міняються місцями. Коли це буде виконано, то нам буде відомий перший елемент послідовності, найменший. Потім ми порівнюємо другий елемент по черзі з усіма іншими, починаючи з третього, і знову міняємо місцями елементи, якщо знайдеться елемент в масиві менший, ніж другий.

Цей процес триває далі для усіх інших елементів до передостаннього, потім масив вважатиметься впорядкованим.

```
int temp;
for (int i = 0; i < imax; i++)
{
        for (int j = imax; j > i; j--)
        {
                if (dim[j] < dim[j - 1]
                {
                        // поміняти місцями dim[j] і dim[j - 1]
                        temp = dim[j];
                        dim[j] = dim[j - 1];
                        dim[j - 1] = temp;
                }
        }
}
```

Г.4 Округлення дійсного числа

```
int intg(float nb)
{
float f1, f2;
int i1;
i1 = (int) nb;
f1 = (float) i1;
f2 = nb - f1;
if(f2 >= 0.5)
        i1+=1;
return i1;
}
```

Г.5 Передача аргументу у функцію по посиланню

```
void function()
{
void f(int, int&); //прототип функції
int n, m;
```

```
//виклик функції
if(n, m); //аргумент записується в звичайному виді
}

void f(int n1, int& m1) // змінна, що передається по посиланню, має сим-
вол &
{
        n1 += m1;
}
```

Г.6 Повернення значення функції по посиланню

```
int a1 = 10;
//виклик функції
x = function(a1);

//Визначення функції
int& function (int& a)
{
        return a*10;
}
```

ДОДАТОК Д

ТАБЛИЦЯ КРИВОЇ НАМАГНІЧУВАННЯ ЗУБЦІВ АСИНХРОННИХ ДВИГУНІВ, СТАЛЬ 2013

B, Тл	**0**	**0,01**	**0,02**	**0,03**	**0,04**	**0,05**	**0,06**	**0,07**	**0,08**	**0,09**
	H, А/м									
0,4	124	127	130	133	136	138	141	144	147	150
0,5	154	157	160	164	167	171	174	177	180	184
0,6	188	191	194	198	201	205	208	212	216	220
0,7	223	226	229	233	236	240	243	247	250	253
0,8	256	259	262	265	268	271	274	277	280	283
0,9	286	290	293	297	301	304	308	312	316	320
1,0	324	329	333	338	342	346	350	355	360	365
1,1	370	375	380	385	391	396	401	406	411	417
1,2	424	430	436	442	448	455	461	467	473	479
1,3	486	495	504	514	524	533	563	574	584	585
1,4	586	598	610	622	634	646	658	670	683	696
1,5	709	722	735	749	763	777	791	805	820	835
1,6	850	878	906	934	962	990	1020	1050	1080	1110
1,7	1150	1180	1220	1250	1290	1330	1360	1400	1440	1480
1,8	1520	1570	1620	1670	1720	1770	1830	1890	1950	2010
1,9	2070	2160	2250	2340	2430	2520	2640	2760	2890	3020
2,0	3150	3320	3500	3680	3860	4040	4260	4480	4700	4920
2,1	5140	5440	5740	6050	6360	6670	7120	7570	8020	8470
2,2	8920	9430	9940	10460	10980	11500	12000	12600	13200	13800
2,3	14400	15100	15800	16500	17200	18000	18800	19600	20500	21400

ДОДАТОК Е

РЕАЛІЗАЦІЯ ОБ'ЄКТНО-ОРІЄНТОВАНОГО ПРОЕКТУВАННЯ АСИНХРОННОГО ДВИГУНА З КОРОТКОЗАМКНЕНИМ РОТОРОМ

```
//Реалізація функцій класу AMotor
Stator::Stator()
{
pi = 3.14159265;
f1 = 50;
mu0 = 4*pi*1e-07;
rad = pi/180.;
gamma115 = 1e-06/41;
omega = 2.*pi*f1;
m = 3;
ro1 = 2.1e-08;
kB = 1.11;
kc = 0.97;
gammaSt = 7800;
kda = 1.6;
kdz = 1.8;
sn = 0.6;
gammaCu = 8100;
kf = 4/pi;
km_old = 1.4;
s1 = 0;
wire_type = 0;
}

void Stator::Init(double mU, double mP, double mn1, double mdelta, int
mschem)
{
U1 = mU;
P2 = mP;
n1 = mn1;
delta = mdelta;
schem = mschem;
```

```
p = 60*f1/n1;
dp = p*2;
fill_array(SteelBox, "steel.dat");
}

void Stator::GetH(double Bzmax, double Bzmin, double Bzmid, double Ba,
int steel)
{
switch (steel)
{
case 0: // Steel 2013
        Hzmax = spline(Bzmax, "St_Bz.dat", "St 2013Z.dat");
        Hzmin = spline(Bzmin, "St_Bz.dat", "St 2013Z.dat");
        Hzmid = spline(Bzmid, "St_Bz.dat", "St 2013Z.dat");
        Ha    = spline(Ba, "St_Ba.dat", "St 2013A.dat");
        break;
case 1: // Steel 2211 & 2312
        Hzmax = spline(Bzmax, "St_Bz.dat", "St 2211Z.dat");
        Hzmin = spline(Bzmin, "St_Bz.dat", "St 2211Z.dat");
        Hzmid = spline(Bzmid, "St_Bz.dat", "St 2211Z.dat");
        Ha    = spline(Ba, "St_Ba.dat", "St 2211A.dat");
        break;
case 2:
        Hzmax = spline(Bzmax, "St_Bz.dat", "St 2411Z.dat");
        Hzmin = spline(Bzmin, "St_Bz.dat", "St 2411Z.dat");
        Hzmid = spline(Bzmid, "St_Bz.dat", "St 2411Z.dat");
        Ha    = spline(Ba, "St_Ba.dat", "St 2411A.dat");
        break;
}
Hz = (Hzmax + Hzmin + 4*Hzmid)/6;
if (Ba < 0.4)
        Ha = 45;
if (Bzmax || Bzmin || Bzmid < 0.4)
        Hzmax = Hzmin = Hzmid = Hz = 100;
}

void Stator::GetH(double Bzmax, double Ba, int steel)
{
switch (steel)
{
case 0: // Steel 2013
        Hz = spline(Bzmax, "St_Bz.dat", "St 2013Z.dat");
        Ha = spline(Ba, "St_Ba.dat", "St 2013A.dat");
```

```
        break;
case 1: // Steel 2211 & 2312
        Hz = spline(Bzmax, "St_Bz.dat", "St 2211Z.dat");
        Ha = spline(Ba, "St_Ba.dat", "St 2211A.dat");
        break;
case 2:
        Hz = spline(Bzmax, "St_Bz.dat", "St 2411Z.dat");
        Ha = spline(Ba, "St_Ba.dat", "St 2411A.dat");
        break;
}
if (Ba < 0.4)
        Ha = 45;
if (Bz < 0.4)
        Hz = 100;
}

//POTOP
Rotor::Rotor()
{
gammaSt_r = 7950;
krot = 0.85;
k02 = 1.8;
Kt = 7;
gammaSt_r = 7950;
}

void Rotor::Init(double mD2, double mL2)
{
D2 = mD2;
L2 = mL2;
n2 = mn2;
hr = mhr;
hl = mhv;
Tr = mT;
Pr = mP;
}

// ПАЗ
Paz::Paz()
{
kL = 0.8;
kob = 0.95;
S = 0.0035;
```

```
nel = 1;
nel_old = 1;
}

void Paz::paz_T(double D, int wind_type)
{
// Паз трапецієвидний
switch (wind_type)
{
case 0: // обмотка одношарова
        if (D <= 0.1)
        {
                biz = 0.00025;
                Spr = 0;
                h2 = 0.0003;
                h0 = 0;
        }
        else if (D > 0.1 && D <= 0.17)
        {
                biz = 0.0003;
                Spr = 0;
                h2 = 0.00035;
                h0 = 0;
        }
        else
        {
                biz = 0.00045;
                Spr = 0;
                h2 = 0.00045;
                h0 = 0;
                }
        break;
case 1: // обмотка двошарова
        if (D <= 0.33)
        {
                biz  = 0.00045;
                Spr = (0.4*b1 + 0.9*b2)/1000;
                h2 = 0.0005;
                h0 = 0.0004;
        }
        else
        {
                biz = 0.00028;
```

```
                Spr = 0.6*(b1 + b2)/1000;
                h2 = 0.00054;
                h0 = 0.00055;
        }
        break;
}// end switch
}

void Paz::paz_1AZ()
{
// Паз прямокутний відкритий, гільзова ізоляція
apiz = p1A[8]/1000;
bpiz = p1A[7]/1000;
hp1 = hp - hsh - hk;
h0 = (p1A[5] + (p1A[1] + p1A[2])/2)/1000;
h1 = hk;
h21 = (p1A[6] + p1A[3]/2 + (p1A[1] + p1A[2])/4)/1000;
h2 = hp1 - h21;
}

void Paz::paz2_3Z()
{
// Паз прямокутний напівзакритий
bpiz = p23[6]/1000;
apiz = p23[7]/1000;
hp1 = hp - hsh - hk;
h0 = (p23[3] + (p23[0] + p23[1])/2)/1000;
h1 = hk;
h2 = (p23[5] + p23[2]/2 + (p23[0] + p23[1])/4)/1000;
h3 = hp1 - 2*h2;
}

AMotor::AMotor() : Stator(), Paz(), Rotor(){}
void AMotor::Init(double mU, double mP, double mn1, double mdelta, int
mschem
double mD2, double mL2,    double mT, double mn2, double mhr, double
mhv)
{
Stator::Init(mU, mP, mn1, mdelta, mschem);
Rotor::Init(mD2, mL2);
Rotor::Init(mP, mT, mn2, mhr, mhv);
D = D2 - 2*(hr + delta);
tau = pi*D/dp;
```

```
p = 60*f1/n1;
dp = p*2;
}
void AMotor::Geom(double mld)
{
ld = mld;

if (D <= 0.17)
        dbp = dhp = 0.0001;
else if (D > 0.17 && D <= 0.33)
        dbp = dhp = 0.0002;
else if (D > 0.33 && D <0.5)
        dbp = dhp = 0.0003;
else
{
        dbp = 0.0004;
        dhp = 0.0003;
}

if (ld/tau >= 2)
        kdd = 0.09;
else
        kdd = spline(ld/tau, "kdd_arg.dat", "kdd.dat");

W1_old = ke*U1/(4.44*Fim*kob*f1);
W1_old = (double) intg(W1_old);
I21_old = pi*p*F2/(6*sqrt(2)*W1_old*kob);
Bd_old = p*Fim/(D*ld);

}// End of Geom()

void AMotor::Wind(double mZ1, int mwind_type)
{
wind_type = mwind_type;
Z1 = mZ1;
for (int i = 0; i<= 10; i++)
        dima[i] = dimUp[i] = dimUpm[i] = 0;
nmin = 0;
amin = 0;
aindex = 0;
I1 = I1c_old.mod;
q = Z1/(dp*m);
//Aold = 2*m*I1*W1old/(pi*D);
```

```
Up1 = pi*D*A_old/(I1*Z1);
t1 = pi*D/Z1;
if (!wind_type) // 1x - sloy
        ku = betta = 1.;
else // 2x - sloy
{
        betta = 2.*(3.*q);
        ku = sin(pi/2.)*betta;
}
if (q <= 6)
        kr = dimkr[intg(q)];
else
        kr = 0.955;
kob = kr*ku;
// find  a
dima[0] = 1.;
dimUp[0] = Up1;
int dpp, pp;
dpp = int(dp);
pp  = int(p);
int k = 0;
int n = 0;
switch (wind_type)
{
case 0: // обмотка одношарова
{
        for (int i = 0; i <= pp; i++)
        {
                k = pp/(i+1);
                if (k*(i+1) == pp)
                {
                        dima[n]   = (double)(i+1);
                        dimUp[n]  = (double)intg((i+1)*Up1);
                        double uc(i+1)*Up1;
                        dimUpm[n] = fabs(uc - dimUp[n]);
                        n++;
                        aindex++;
                }
        }// end for
        break;
}
case 1: // обмотка двошарова
{
```

```
        for (int i = 0; i <= dpp; i++)
        {
                k = dpp/(i+1);
                if (k*(i+1) == dpp)
                {
                        dima[n]   = (double)(i+1);
                        dimUp[n]  = (double)intg((i+1)*Up1);
                        double uc(i+1)*Up1;
                        dimUpm[n] = fabs(uc - dimUp[n]);
                        n++;
                        aindex++;
                }
        }// end for
        break;
}
} // end switch

nmin = fmin(dimUpm, n);
Up = dimUp[nmin];
a  = dima[nmin];
W1 = Up*Z1/(2.*a*m);
A = 2.*I1*W1*m/(pi*D);
Fi = ke*U1/(4.*kB*W1*kob*f1);
Bd = p*Fi/(D*ld);
}

void AMotor::Magn(int msteel)
{
// Розрахунок магнітного кола
steel = msteel;
if (slot_type <= 2 )//прямокутний паз
        Stator::GetH(Bzmax, Bzmin, Bzmid, Ba, steel);
else // трапецієвидний паз
        Stator::GetH(Bzmax, Ba, steel);
gamma = pow(5 + bsh/delta);
kd = t1/(t1 - gamma*delta);
Fd = 1.59e06*Bd*delta*kd;
Fz = hp*Hz;
La = pi*(D - hp)/dp;
bkt = pi*(D - hp)*betta/dp;
Fa = La*Ha;
Fs = Fd + Fz + Fa;
km = Fs/Fd;
```

```
Im = p*Fs/(0.9*3*W1*kob);
xm = 2*m*f1*mu0*D*ld*pow(km*kd*delta);
}

void AMotor::ParamS()
{
// Розрахунок параметрів
// Лістинг аналогічний приведеному для процедурного підходу
}

void AMotor::Char()
{
// РОЗРАХУНОК ВТРАТ
// Лістинг аналогічний приведеному для процедурного підходу
}

int AMotor::AnalizeQ(double mZ)
{
int flag = 0;
int iq = 0;
double bb = 0;

Z1 = mZ;
q = Z1/(dp*m);
iq = intg(q)/2;
if (!(q - (double) intg(q)))
        flag = 0;//q - ціле (знаменник дорівнює 0)
else
{
        cd = floor(q);
        bb = q - cd;
        d_d = Get_znam(Z1, dp);
        c = d_d*cd;
        int id = intg(d_d)/2;
if (id*2 == intg(d_d))
        flag = 2;//знаменник парний
else
        flag = 1;//знаменник непарний
}
return flag;
}
```

ТЛУМАЧНИЙ СЛОВНИК

Абстрактний клас - клас, що знаходиться на вершині ієрархії класів, об'єкти якого ніколи не будуть реалізовані. Абстрактний клас існує з єдиною метою - бути батьківським по відношенню до похідних. У тілі абстрактного класу повинна знаходитися чиста віртуальна функція.

Апроксимація (від лат. approximo - наближаюся) - заміна одних математичних об'єктів (наприклад, чисел або функцій) іншими, простішими і в тому або іншому сенсі близькими до початкових (наприклад, кривих ліній близькими до них ламаними).

Аргумент - вхідні дані для функції.

Атом - в Ліспі - це простий (на відміну від списку) тип даних: число, символьний рядок, функція.

Базовий клас - клас, що знаходиться на вершині дерева ієрархії і породжує інші класи.

Булева операція - система алгебри великої кількості, що складається з двох елементів: Брехня і Істина. Як правило, в математичних виразах Брехня ототожнюється з логічним нулем, а Істина - з логічною одиницею, а операції заперечення (НІ), кон'юнкції (І) і диз'юнкції (АБО) визначаються в звичному розумінні. Своїм існуванням наука "алгебра логіки" зобов'язана англійському математикові Джорджу Булю, який досліджував логіку висловлювань.

Визначення змінної - виділення пам'яті під змінну.

Вираз - будь-яка комбінація змінних, констант і операцій, що призводить до обчислення деякого значення.

Віртуальний базовий клас - спеціальне оголошення класу, що є базовим по відношенню до класу-нащадка при множинному спадкоємстві.

Віртуальна функція - оголошення функції, яка в процесі виконання програми вирішує, яку саме функцію необхідно викликати. Віртуальні функції реалізують ідею поліморфізму.

Глобальні дані - дані, призначені для спільного використання декількома функціями, оголошеними як в одному, так і в різних файлах програми.

Декремент (від лат. decrementum - зменшення) - операція зменшення значення змінної.

Деструктор - метод класу, що автоматично викликається при знищенні об'єкту і звільняє пам'ять, виділену конструктором при створенні об'єкту.

Діалог - вікно програми, в якому здійснюється взаємодія користувача і програми.

Діалогове застосування - те ж, що і діалог.

Директива препроцесору - рядок, який починається з символу "грати" #. Наприклад, директива #include вказує препроцесору включити в компільований файл вміст іншого файлу.

Заголовний файл - файл, що включається за допомогою директиви препроцесору *#include*.

Змінна - програмна одиниця мови програмування, що має символьне ім'я і набуває різних значень. Змінні зберігаються в певних ділянках пам'яті комп'ютера.

Зона видимості - частина програмного коду, в межах якого забезпечується доступ до значення певної змінної. Зона видимості визначає, з яких частин програми можливий доступ до змінної.

Ідентифікатор - ім'я, що дається змінним і іншим елементам програми.

Ієрархія (від грец. hieros - священний і arch; - влада) - розташування частин або елементів цілого в порядку від вищого до нижчого. У програмуванні цей термін вживається для характеристики організації спадкоємства класів.

Ініціалізація - надання певного значення змінним.

Інкапсуляція - програмне приховання даних від зовнішньої дії, що захищає їх від випадкової зміни. При інкапсуляції забороняється доступ функціям, що не є методами відповідного класу, до прихованих або захищених даних об'єкту.

Інкремент (від лат. incrementum - зростання, збільшення) - операція збільшення значення змінної.

Інтерполяція (від лат. interpolatio - зміна, переробка) - в математиці і статистиці, відшукування проміжних значень величини по деяких відомих її значеннях.

Інтерфейс (англ. interface) - система зв'язків з уніфікованими сигналами і апаратурою, призначена для обміну інформацією між пристроями обчислювальної системи (наприклад, між пристроєм введення даних і пристроєм, що запам'ятовує).

Клас - конструкція мови програмування, що описує об'єднання даних і методів. Клас є описом сукупності схожих між собою об'єктів.

Код - текст програми.

Конструктор - метод класу, що виконується автоматично при створенні об'єкту і ініціалізує його поля.

Лістинг - витяг або повне представлення коду програми в друкарському виді (найчастіше з коментарями програміста).

Локальні дані - дані, які знаходяться усередині функції і призначені для використання саме цією функцією.

Майстер - набір вкладок, що допускає введення інформації користувачем у вікні властивостей в строго певному порядку. Майстер призначений для організації проходження користувачем застосування послідовних етапів рішення комплексної задачі. Майстер містить кнопки "Назад", "Далі", "Відміна" і "Готово".

Масив (франц. massif, букв. - потужний, суцільний), сукупність великої кількості однорідних за якими-небудь ознаками об'єктів, предметів, даних. У програмуванні - програмна конструкція, що об'єднує однотипні дані.

Метод класу - функція, що входить до складу класу.

Множинне спадкоємство - спадкоємство, при якому клас-нащадок породжений від декількох базових класів.

Об'єкт - унікальний екземпляр класу, що об'єднує дані і дії, яки виробляється над цими даними.

Об'єктно-орієнтоване програмування - це методологія програмування, яка заснована на представленні програми у вигляді сукупності об'єктів, кожен з яких є реалізацією певного класу, а класи утворюють ієрархію на принципах спадкоємства.

Об'єктно-орієнтоване проектування - це методологія проектування, що сполучає в собі процес об'єктної декомпозиції і прийоми

представлення як логічної і фізичної, так статичної і динамічної моделей проектованої системи.

Оголошення змінної - вказівка імені і типу змінної.

Операнд - величина, що є об'єктом операції, яку реалізує ЕОМ в ході виконання програми обчислень.

Оператор - припис в мові програмування, призначений для завдання деякої завершеної дії в процесі переробки інформації на ЕОМ.

Оптимізація - процес вибору найкращого варіанту з можливих.

Перевантаження - окремий випадок поліморфізму, наділ існуючій операції (наприклад, складанню) можливості здійснювати дії над операндами нового типу.

Перерахування - спосіб створення призначеного для користувача типу даних, в якому змінні створюваного типу можуть приймати заздалегідь відому кінцеву безліч значень.

Повідомлення - в програмуванні - виклик методів об'єктів. У Windows повідомлення є тим засобом, за допомогою якого операційна система може дати знати застосуванню, що сталася яка-небудь подія (наприклад, користувач натиснув клавішу на клавіатурі).

Покажчик - змінна, така, що містить в собі значення адреси. При оголошенні змінної-покажчика після оголошення типу даних змінної записується символ зірочка "*". Зірочка читається як "покажчик на…", тобто вказує на певний тип даних.

Поле класу, структури - змінні, такі, що входять до складу класу, структури.

Поліморфізм - використання операцій і функцій по-різному залежно від того, з якими типами величин вони працюють.

Посилання - псевдонім, або альтернативне ім'я змінної. Одним із застосувань посилань є передача аргументів у функцію. При цьому у функцію передається не значення, а адреса змінної-аргументу в пам'яті. При використанні посилань безпосередньо після вказівки типу змінній записується символ амперсанд "&".

Постфіксна форма - запис знаку операції інкременту (декременту) після свого операнда, інкремент буде виконаний останнім.

Потік - програмна абстракція, що відбиває переміщення даних від джерела до приймача.

Похідний клас - клас, що наслідує властивості одного або декількох базових класів.

Префіксна форма - запис знаку операції інкременту (декременту) перед своїм операндом, інкремент буде виконаний першим.

Процедурне програмування - розподіл програми на декілька компонентів, кожен з яких є набором інструкцій.

Симплекс (від лат. simplex простий) - в математиці простий опуклий багатогранник даної кількості вимірів, наприклад трикутник на площини, тетраедр в просторі. У лінійному програмуванні симплексом називають непорожню множину.

Спадкоємство - поняття, що реалізовує принцип передачі даних і методів від базового класу до похідних.

Список - в AutoLisp перелік атомів або списків, відокремлених один від одного пропусками і ув'язнених в дужки. Списки можуть бути вкладеними.

Сплайн (від англ. spline - рейка, лінійка) - це визначена в деякій області кусочно-поліноміальна функція. Таким чином, існує розбиття

області на підобласті таке, що усередині кожної підобласті сплайн є поліномом деякого ступеня.

Сплайн-інтерполяція - інтерполяція за допомогою сплайнів.

Структура - одна з конструкцій мови програмування, що реалізовує об'єднання різнорідних даних.

Форма діалогу - заготівля вікна програми в редакторові ресурсів середовища проектування.

Функціональний модуль - те ж, що і функція.

Функція - програмний засіб, що виконує закінчену послідовність дій.

Цільова функція - залежна величина (функція), яка визначається проектними параметрами, за допомогою яких робиться вибір оптимального рішення або порівняння двох і більше альтернативних рішень.

Чиста віртуальна функція - віртуальна функція, після оголошення якої додано вираз порівняння з нулем " =0 ".

Шар - в AutoCad іменований примітив, що містить які-небудь графічні примітиви. Будь-який графічний об'єкт має шар і тільки один.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

№ п/п	Найменування	Познач. в тексті	Познач. у вікнах діалогу	Познач. у лістингах програми
	Головна сторінка			
1	Кількість пар полюсів	p	-	p
2	Кількість полюсів	2p	-	dp
	Сторінка 1			
3	Полюсний крок	τ	tau	tau
4	Довжина повітряного проміжку	l_δ	ld	ld
5	Індукція в повітряному проміжку	B_δ	Bd	Bd
6	ККД	η	kkd	kkd
7	Коефіцієнт потужності	cosφ	cos	cosFi
8	Величина $\lambda = l\delta / \tau$	λ	Lamda	lamda
	Сторінка 2			
9	Мінімальний зубцевий крок	t_{1min}	t1min	t1min
10	Максимальний зубцевий крок	t_{1max}	t1max	t1max
11	Зубцевий крок	t_1	t1	t1
12	Мінімальна кількість зубців статора	Z_{1min}	Z1min	Z1min
13	Максимальна кількість зубців статора	Z_{1max}	Z1max	Z1max
14	Кількість зубців статора	Z_1	Z1	Z1
15	Кількість елементарних провідників	$n_{ел}$	nel	nel
16	Переріз елементарного проводу	$q_{ел}$	qel	qel

№ п/п	Найменування	Познач. в тексті	Познач. у вікнах діалогу	Познач. у лістингах програми
17	Переріз ефективного проводу	$q_{еф}$	qef	qef
18	Діаметр елементарного проводу	$d_{ел}$	del	del
19	Діаметр ізольованого проводу	$d_{із}$	diz	diz
20	Кількість пазів на полюс і фазу	$U_{п}$	Up	Up
21	Коефіцієнт е.р.с.	$k_{е}$	kE	kE
22	Номінальний струм	$I_{1н}$	I1n	I1n
23	Магнітний потік	Ф	Fi	Fi
	Сторінка 3			
24	Індукція у зубцях статора	B_{z1}	Bz1	Bz1
25	Коефіцієнт заповнення паза	$k_{з}$	kz	kzp
26	Ширина шлицю	$h_{ш}$	hsh	hsh1
27	Висота шлицю	$b_{ш}$	bsh	bsh1
28	Площа пазової прокладки	$S_{пр}$	Spr	Spr
29	Припуск на висоту паза	Δh	dhp	dhp
30	Припуск на ширину паза	Δb	dbp	dbp
31	Ширина ізоляції	$b_{із}$	biz	biz
32	Висота паза статора	$h_{п}$	hp	hp1
33	Ширина зубця статора	b_{z1}	bz1	bz1
34	Проміжні значення ширини паза статора	b_1 b_2	b1 b2	b1 b2
35	Проміжне значення висоти паза статора	h_1	h1	h1
36	Ширина паза з урахуванням припуску	$b_1^{/}$ $b_2^{/}$	-	b11 b21
37	Висота паза з урахуванням при-	$h_1^{/}$	-	h11

№ п/п	Найменування	Познач. в тексті	Познач. у вікнах діалогу	Познач. у лістингах програми
	пуску			
38	Площа ізоляції	$S_{із}$	Siz	Siz
39	Площа паза	$S_п$	-	Sp
	Сторінка 4			
40	Повітряний проміжок	δ	delta	delta
41	Кількість зубців ротора	Z_2	Z2	Z2
42	Висота паза ротора	$h_{п2}$	hp2	hp2
43	Ширина зубця ротора	b_{z2}	bz2	bz2
44	Проміжні значення ширини паза ротора	b_1 b_2	b1 b2	b1r b2r
45	Проміжне значення висоти паза ротора	h_1	h1	h1r
46	Висота шлицю ротора	$h_ш$	hsh	hsh2
47	Ширина шлицю ротора	$b_ш$	bsh	bsh2
48	Висота вусика паза ротора	$h_ш'$	hsh1	hshr1
49	Ширина к.з. кільця	$a_к$	ak	ak
50	Висота к.з. кільця	$b_к$	bk	bk
51	Середній діаметр кільця	$D_{кл,ср}$	-	Dk
52	Індукція в зубцях ротора	B_{z2}	Bz2	Bz2
53	Переріз стержня ротора	q_c	-	qc2
54	Струм в стержні ротора	I_2	-	I2
55	Зубцевий крок ротора	t_2	-	t2
	Сторінка 5			
56	Висота спинки ротора	h_j	hj	hj
57	Коефіцієнт Картера	k_δ	-	kd

№ п/п	Найменування	Познач. в тексті	Познач. у вікнах діалогу	Познач. у лістингах програми
58	Відносне значення струму намагнічування	$I_{\mu*}$	Imo	Imo
59	МРС повітряного проміжку	F_{δ}	-	Fd
60	МРС на пару полюсів	$F_{ц}$	-	Fc
61	Коефіцієнт насичення магнітного кола	k_{μ}	km	km
	Сторінка 6			
62	Коефіцієнт вильоту лобової частини	$K_{вил}$	Kv	Kv
63	Ширина котушки	$b_{кт}$	-	bkt
64	Середня довжина витка	$l_{ср}$	-	lmid
65	Відносне скорочення кроку обмотки	β	betta	betta
66	Коефіцієнт	k_{β}	kbetta	kbetta
67	Коефіцієнт	$k_{\beta}^{/}$	kbetta1	kbetta1
68	Коефіцієнт	Δ_z	ddz	ddz
69	Коефіцієнт	$\beta_{ск}$	betta sk	betta_sk
70	Коефіцієнт скосу	$k_{ск}^{/}$	ksk	ksk
71	Коефіцієнт (статор)	ξ	-	ksi
72	Коефіцієнт пазового розсіювання (статор)	$\lambda_{п}$	-	lamda_p
73	Коефіцієнт лобового розсіювання (статор)	$\lambda_{л1}$	-	lamda_l
74	Коефіцієнт диференційного розсіювання (статор)	$\lambda_{д1}$	-	lamda_d

№ п/п	Найменування	Познач. в тексті	Познач. у вікнах діалогу	Познач. у лістингах програми
75	Коефіцієнт пазового розсіювання (ротор)	$\lambda_{п2}$	-	lamda_p2
76	Коефіцієнт лобового розсіювання (ротор)	$\lambda_{л2}$	-	lamda_l2
77	Коефіцієнт диференційного розсіювання (ротор)	$\lambda_{д2}$	-	lamda_d2
78	Коефіцієнт (ротор)	ξ	-	ksi2
79	Приведені опори обмотки ротора	$r_2^{/}$ $x_2^{/}$	r21 x21	r21 x21
80	Відносні опори обмотки ротора	$r_{2*}^{/}$ $x_{2*}^{/}$	-	r21o x21o
	Сторінка 7			
81	Густина сталі	γ_c	-	gammaSt
82	Коефіцієнт	β_{02}	betta02	betta02
83	Втрати в сталі основні	$P_{ст,осн}$	-	Pst_main
84	Втрати в сталі	$P_{ст}$	Pst	Pst
85	Втрати поверхневі	$P_{пов}$ $p_{пов}$	-	Ppov ppov
86	Втрати, індукція пульсаційні	$P_{пул}$ $B_{пул}$	-	Ppul Bpul
87	Втрати в сталі додаткові	$P_{ст,дод}$	-	Pst_dob
88	Втрати механічні	$P_{мех}$	-	Pmeh
89	Втрати електричні статора в режимі холостого ходу	$P_{e1\ x,x}$	-	Pe10
90	Струм холостого ходу активний	$I_{x,x,a}$	-	Ixxa

№ п/п	Найменування	Познач. в тексті	Познач. у вікнах діалогу	Познач. у лістингах програми
91	Струм холостого ходу	$I_{x,x}$	Ixx	I0
92	Коефіцієнт потужності холостого ходу	$\cos\varphi_{x,x}$	cosFixx	cosFi0
	Сторінка 8			
93	Приведений струм ротора (Т-подібна схема)	$I_2^{/}$	-	I21
94	Приведений струм ротора (Г-подібна схема)	$I_2^{//}$	-	I211
95	Сумарні втрати в двигуні	ΣP	-	Psum
	Сторінка 9			
96	Коефіцієнт насичення	$k_{нас}$	knac	knac
97	Струм з урахуванням насичення	$I_{1нас}$	I1nac	I1nac
98	Струм з урахуванням витіснення	I_1	I1ksi	I1p
99	Коефіцієнт	ξ	-	ksip
100	Коефіцієнт	φ	-	fip
101	Приведений активний опір обмотки ротора з урахуванням ефекту витіснення струму	$r_{2\xi}^{/}$	-	r2ksi
102	Приведений індуктивний опір обмотки ротора з урахуванням ефекту витіснення струму	$x_{2\xi}^{/}$	-	x21ksi
103	Приведений індуктивний опір обмотки статора з урахуванням насичення	$x_{1нас}$	-	x1nac
104	Приведений індуктивний опір обмотки ротора з урахуванням	$x_{2\xi нас}^{/}$	-	x21nac

№ п/п	Найменування	Познач. в тексті	Познач. у вікнах діалогу	Познач. у лістингах програми
	ефекту витіснення струму і насичення			
105	МРС віднесена до одного паза обмотки статора	$F_{п,ср}$	-	Fpmid
106	Фіктивна індукція потоку розсіювання у повітряному проміжку	$B_{ф\,\delta}$	-	Bfd
107	Коефіцієнт	χ_{δ}	-	xd
108	Коефіцієнт зменшення магнітної провідності паза статора при насиченні	$\Delta\lambda_{п1нас}$	-	dlamda_p1
109	Коефіцієнт магнітної провідності пазового розсіювання обмотки статора з урахуванням насичення	$\lambda_{п1нас}$	-	lamda_p1nac
110	Коефіцієнт магнітної провідності диференційного розсіювання обмотки статора з урахуванням насичення	$\lambda_{д1нас}$	-	lamda_d1nac
111	Коефіцієнт магнітної провідності пазового розсіювання обмотки ротора з урахуванням ефекту витіснення струму	$\lambda_{п2\xi}$	-	lamda_p2ksi
112	Коефіцієнт зменшення магнітної провідності паза ротора при насиченні	$\Delta\lambda_{п2нас}$	-	dlamda_p2

№ п/п	Найменування	Познач. в тексті	Познач. у вікнах діалогу	Познач. у лістингах програми
113	Коефіцієнт магнітної провідності пазового розсіювання обмотки ротора з урахуванням насичення і ефекту витіснення струму	$\lambda_{п2\xi нас}$	-	lamda_p2nac
114	Коефіцієнт магнітної провідності диференційного розсіювання обмотки ротора з урахуванням насичення	$\lambda_{д2нас}$	-	lamda_d2nac
	Сторінка 10			
115	Коефіцієнт тепловіддачі з поверхні	α_1	alfa1	alfa1
116	Середня еквівалента теплопровідність пазової ізоляції	$\lambda_{екв}$	-	lamda_ekv
	Середнє значення теплопровідності внутрішньої ізоляції котушок	$\lambda'_{екв}$	lamda_ekv1	lamda_ekv1
117	Одностороння товщина ізоляції лобової частини	$b_{із,лоб}$	-	bizlob
118	Периметр умовної поверхні охолодження поперечного перерізу паза	$П_{п1}$	Pp	Pp1
119	Периметр умовної поверхні охолодження лобової частини	$П_{л1}$	-	PPlob
120	Температурний коефіцієнт підігріву повітря	$\alpha_в$	alfa_v	alfa_v

№ п/п	Найменування	Познач. в тексті	Познач. у вікнах діалогу	Познач. у лістингах програми
121	Еквівалентна поверхня охолодження корпуса	$S_{кор}$	-	Sk
122	Сума втрат, що відводяться у повітря з двигуна	$\Sigma P'_{в}$	-	Psum1
123	Електричні втрати в пазовій частині	$P'_{е,п1}$	-	Pep1
124	Електричні втрати в лобовій частині	$P'_{е,л1}$	-	Pelob1
125	Перевищення температури внутрішньої поверхні осердя всередині двигуна	$\Delta\theta_{пов1}$	-	dtpov1
126	Перепад температури в ізоляції пазової частини обмотки статора	$\Delta\theta_{із,п1}$	-	dtizp
127	Перепад температури по товщині ізоляції лобової частини	$\Delta\theta_{із,л1}$	-	dtizlob
128	Перевищення температури зовнішньої поверхні лобових частин над температурою повітря в середині двигуна	$\Delta\theta_{пов,л1}$	-	dtpovlob
129	Середнє перевищення температури обмотки статора над температурою повітря в середині двигуна	$\Delta\theta'_{1}$	-	dt11
130	Перевищення температури повітря в середині двигуна над температурою навколишнього сере-	$\Delta\theta_{в}$	-	dtv

№ п/п	Найменування	Познач. в тексті	Познач. у вікнах діалогу	Познач. у лістингах програми
	довища			
131	Середнє перевищення температури обмотки статора над температурою навколишнього середовища	$\Delta\theta_1$	dt1	dt1
132	Необхідні витрати повітря	$Q_в$	Qv	Qv
133	Витрати повітря, що забезпечуються зовнішнім вентилятором	$Q'_в$	Qv1	Qv1
	Сторінка 11			
134	Маса ротора	$m_р$	-	mr
135	Мех. напруження перерізу А	$\sigma_{прА}$	-	sigma_a
136	Мех. напруження перерізу В	$\sigma_{прВ}$	-	sigma_b
137	Мех. напруження перерізу С	$\sigma_{прС}$	-	sigma_c
138	Мех. напруження перерізу Д	$\sigma_{прД}$	-	sigma_d
139	Мех. напруження перерізу Г	$\sigma_{прГ}$	-	sigma_g
140	Мех. напруження перерізу Ж	$\sigma_{прЖ}$	-	sigma_z
141	Найбільша механічна напруженість	$\sigma_{пр}$	-	sigma_max

ЗМІСТ

РОЗДІЛ 2

СЕРЕДОВИЩЕ ПРОГРАМНОГО ПРОЕКТУВАННЯ VISUAL STUDIO

РОЗДІЛ 3

AUTOCAD І СЕРЕДОВИЩЕ ПРОЕКТУВАННЯ VISUAL LISP

РОЗДІЛ 4

МЕТОДИ І ЗАСОБИ ОПТИМІЗАЦІЇ

РОЗДІЛ 5
ПОВНИЙ ПРИКЛАД ОПТИМАЛЬНОГО ПРОЕКТУВАННЯ АСИНХРОННОГО ДВИГУНА З КОРОТКОЗАМКНЕНИМ РОТОРОМ

НАВЧАЛЬНЕ ВИДАННЯ

М. М. Заблодський,

В. Є. Плюгін

СИСТЕМИ АВТОМАТИЗОВАНОГО ПРОЕКТУВАННЯ ЕЛЕКТРОМЕХАНІЧНИХ ПРИСТРОЇВ

Навчальний посібник

В авторській редакції

Підп. до друку. Формат 60x84 $^{1}/_{16}$. Папір офс.
Друк RISO. Ум. друк. арк. 18,13 Зам. № Наклад пр.
Видавництво не несе відповідальності за зміст матеріалу, наданого автором до друку.
Видавець та виготівник:
Донбаський державний технічний університет
94204, Луганська обл., м. Алчевськ, пр. Леніна, 16.
(ТВО "Ладо", ауд. 2113, т/факс 2-02-59)
E-mail: info@dmmi.edu.ua Web-site: http//www.dmmi.edu.ua
Свідоцтво Держкомтелерадіо серія ДК, № 2010 від 12.11.2004

318