МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

ЦЕНТР ЗАОЧНОЇ, ДИСТАНЦІЙНОЇ ТА ВЕЧІРНЬОЇ ФОРМ НАВЧАННЯ

КАФЕДРА КОМП'ЮТЕРНИХ НАУК

ВИПУСКНА РОБОТА

на тему:

«Комп'ютерне моделювання системи стабілізації вертикального стану корпусу корабля»

Завідувач
випускаючої кафедри **Довбиш А.С.**

Керівник роботи **Авраменко В. В.**

Студента групи ІНдн-71С **Потоцький Д. Л.**

СУМИ 2021

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

Центр заочної, дистанційної та вечірньої форм навчання

Кафедра комп'ютерних наук

Затверджую_____

Зав. кафедрою Довбиш А.С.

"_____" _____2021 р.

ЗАВДАННЯ

до випускної роботи

Студента четвертого курсу, групи ІНдн-71С спеціальності "Комп'ютерні науки" дистанційної форми навчання Потоцького Дениса Леонідовича.

Тема: "Комп'ютерне моделювання системи стабілізації вертикального стану корпусу корабля"

Затверджена наказом по СумДУ

№_____ від _____2021 р.

Зміст пояснювальної записки: 1) аналітичний огляд методів випадкового пошуку екстремуму; 2) постановка завдання й формування завдань дослідження; 3) опис основних положень, математичних моделей і критеріїв, що використовуються системою стабілізації вертикального стану корпусу корабля; 5) розробка інформаційного і програмного забезпечення для цієї системи; 6) аналіз результатів моделювання.

Дата видачі завдання "_____"_____2021 р.

Керівник випускної роботи _____ Авраменко В. В.

Завдання прийняв до виконання _____ Потоцький Д. Л.

РЕФЕРАТ

Записка: 39 стор., 8 рис., 4 табл., 1 додаток, 12 джерел.

Об’єкт дослідження — система стабілізації вертикального стану корпусу корабля.

Мета роботи — розробка алгоритму зменшення тривалості коливань корпусу корабля відносно вертикалі, яка проходить через центр ваги корабля, на основі інформаційних мір випадкового пошуку мінімуму цієї тривалості.

Методи дослідження — метод простого випадкового пошуку.

Результати — розроблено алгоритм та програмне забезпечення системи стабілізації вертикального стану корпусу корабля. При цьому задача мінімізації тривалості коливань корпусу корабля відносно вертикалі, яка проходить через центр ваги корабля, та знаходження оптимальних параметрів системи стабілізації розв’язана в рамках методу простого випадкового пошуку. Використано метод Рунге-Кутта для розв’язання задачі Коші. Розроблений алгоритм реалізовано у формі програмного забезпечення, створеного за допомогою інструментального програмного середовища Visual Studio 16.9.

СИСТЕМА СТАБІЛІЗАЦІЇ, ТРИВАЛІСТЬ КОЛИВАНЬ, ВЕРТИКАЛЬНИЙ СТАН, МЕТОД ПРОСТОГО ВИПАДКОВОГО ПОШУКУ, ДИФЕРЕНЦІАЛЬНІ РІВНЯННЯ, ЗАДАЧА КОШІ, ЧИСЕЛЬНИЙ МЕТОД РУНГЕ-КУТТА 4-ГО ПОРЯДКУ, ВЕРТИКАЛЬ, ЯКА ПРОХОДИТЬ ЧЕРЕЗ ЦЕНТР ВАГИ КОРАБЛЯ.

ЗМІСТ

ВСТУП

В даній роботі розв’язується задача комп’ютерного моделювання системи стабілізації вертикального стану корпусу корабля за допомогою методів випадкового пошуку екстремуму.

Задача оптимізації є дуже поширеною в науці. Кожного разу, коли потрібно щось обрати, краще зробити це оптимальним способом, ніж випадковим. Потреба щось обирати стоїть перед людством з прадавніх часів, але актуальна і нині. Наприклад, архетипова «задача комівояжера» – обійти деяку кількість місць, витративши якомога менше ресурсів – часу, грошей і т. д. І подібні проблеми оптимізації виникають майже перед кожною людиною кілька разів на день.

1 ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

1.1 Задачі оптимізації

1.1.1 Історія

Вперше математично сформулював задачу оптимізації Ж. Фур'є [1] у 1820 році. Це була задача лінійного програмування. Першим математичним методом розв'язання задач оптимізації став симплекс-метод. Назва задачі, яка включає термін «програмування», склалася тому, що спочатку ця задача мала місце в економіці, а англійське слово «programming» означає складання планів. Тому нормально, що математичний опис задачі і її економічна інтерпретація тісно пов'язані. Саму назву «лінійне програмування» представив світу Дж. Данциг у 1949 році, і означала вона оптимізацію лінійних функцій за умови, що функції обмежень також лінійні. Тому «Математичне програмування» – це клас задач, ціль яких – вибір оптимальної «програми» дій.

У 1930-х роках було виділено у чистому вигляді екстремальний клас задач лінійного програмування. Зробили це фон Нейман і Канторович, який описав багато задач лінійного програмування і метод їх вирішення, майже еквівалентний симплекс-методу. Тоді ж була сформульована «проблема вибору», метод її розв'язання був названим «венгерським методом».

У 1940-х роках Канторович і ще одна особистість [1, 2] запропонували метод вирішення транспортних задач. Після цього паралельно з лінійним розвивалося і нелінійне програмування. У цій області оптимізації серед функцій, які описують задачу, є нелінійні. 1951 рік відзначився формулюванням всеохоплюючих умов оптимальності нелінійної функції за Куном і Такером. Є навіть мови програмування, створені спеціально для задач оптимізації.

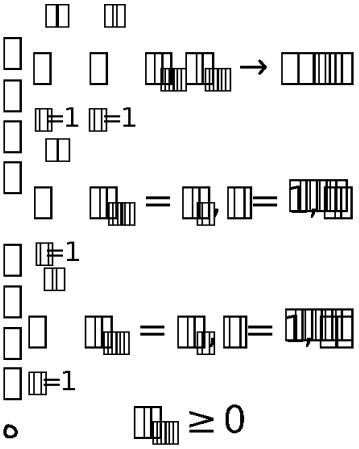
1.1.2 Класифікація задач оптимізації

Існують багато ознак [1, 3], за якими можна класифікувати задачі оптимізації. Найбільш поширена з таких ознак – за умовністю: в залежності від того, де потрібно знайти екстремум – на всьому просторі або на деякій гіперповерхні, обмеженій умовами  або . За цією ознакою виділяють задачі безумовної та умовної оптимізації.

Інший спосб класифікації задач оптимізації – за властивостями обмежуючих функцій – це гіперплощина, гіперпараболоід або інша гіперповерхня. В залежності від цього, розрізняють лінійні, квадратичні, експоненціальні та інші задачі оптимізації.

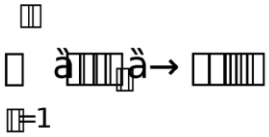
1.1.3 Транспортна задача

Транспортна задача є одним з канонічних прикладів [1, 3-5] задач оптимізації. Основа у неї така: існує деяка кількість виробників деякого товару і деяка кількість споживачів, а також матриця вартості перевезення товару від і-го виробника до j-го споживача. Потрібно за мінімальну вартість перевезти весь вироблений товар в потрібній кількості всім споживачам. Але не обов’язково використовувати всі наявні маршрути, у цьому і є сенс задачі – вибрати із всіх маршрутів декілька оптимальних. Формально це записується таким чином (*x* – об’єм товару, який буде перевезено):



1.1.4 Проблема штейнера

Первісна проблема Штейнера [1] формулювалася наступним чином: потрібно зробити «роз'їзд» в такій точці, щоб сумарна довжина доріг від нього до трьох «станцій» була мінімальна. З часом задача абстрагувалася та узагальнилася. І тепер у ній потрібно мінімізувати суму відстаней від цільовоі точки до деякоі кількості заданих точок, і не обов'язково на площині. Формально це записується так:



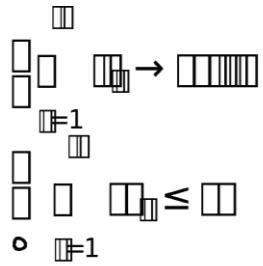
Це один з прикладів задач безумовноі оптимізації: екстремум потрібно знайти на всьому просторі .

1.1.5 Задача про розподіл ресурсів

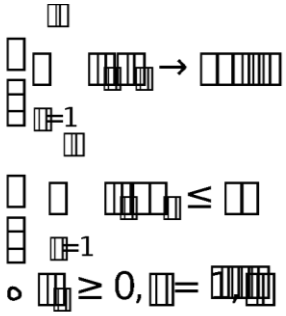
Потрібно розпланувати доходи і видатки [1, 4-5] на декількох підприємствах впродовж деякого періоду. Відома початкова сума, яку можна вкласти, функція прибутку на кожному підприємстві за кожен субперіод, і функція повернення матеріальних ресурсів, які знову можна вкласти. Потрібно у кожному субперіоді так вкласти матеріальні ресурси, щоб сума, яка буде отримана у результаті, була екстремальною. Формально ця задача записується наступним чином:

1.1.6 Задача про рюкзак

Ця задача є NP-повною [1, 3-5]. Це означає, що величезну множину задач оптимізації можливо звести до цієї. Потрібно укласти декілька «речей» заданої маси у «рюкзак», маса якого має обмеження, так, щоб вартість цих речей була екстремальною (максимальною). Формальною мовою:

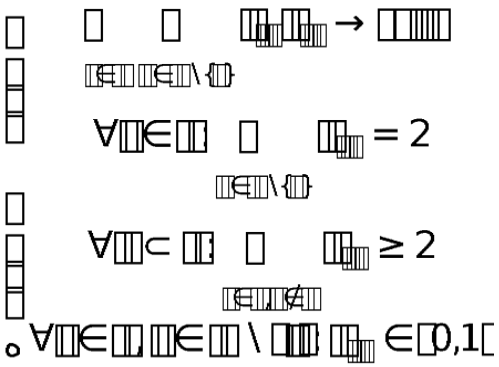


Є також варіант задачі, коли можливо укладати декілька однакових речей. Тоді формальний запис буде таким:



1.1.7 Задача комівояжера

Канонічна, архетипічна та NP-повна задача [3], яку вже було згадано. Існує багато різновидів цієї задачі, але, ймовірно, в усіх є матриця витрат на кожен маршрут. Формальний запис для цієї задачі більш складний, ніж для попередніх. Для того, щоб він був коректним, потрібно ввести позначення V – умовний граф, який охоплює всі можливі пункти та маршрути між ними.



1.2 Огляд методів оптимізації

У конкретній задачі оптимізації перше, що необхідно зробити [1, 4-5] – обрати найбільш оптимальний метод, який досягає кінцевого результату найбільш просто, або досягає результату, найбільш наближеного до кінцевого, при заданій складності. Вибір цього методу істотно залежить від способу формулювання та виду математичної моделі, використаних в задачі.

Основний перелік методів оптимізації такий:

1. Варіаційне числення;
2. Вивчення функцій математичним аналізом;
3. Геометричне програмування;
4. Динамічне програмування;
5. Лінійне програмування;
6. Метод гілок та меж;
7. Невизначені множники Лагранжа;
8. Нелінійне програмування;
9. Принцип максимуму.

При цьому один метод принципово не може вирішувати абсолютно усі задачі, виникаючі на практиці. Одні методи є більш-менш універсальними (як динамічне програмування), інші – придатні для дуже вузького класу задач (як метод еластичної сітки для невеликої підмножини задач комівояжера). Окрім того, деякі методи оптимізації доцільно застосовувати разом.

Також важливо [1, 4-5], що динамічне програмування оптимальне для оптимізації складених з декількох частин процесів, зокрема якщо у кожній окремій частині кількість варіантів невелика. Якщо це не так, застосування динамічного програмування ускладнене через дуже велику потребу в пам’яті.

Ймовірно, найкращий метод оптимізації для вирішення деякоі задачі – це той, який вже використовувався для подібних задач, та є досвід, що такий метод виявив себе краще, ніж декілька інших.

1.3 Методи випадкового пошуку екстремуму

1.3.1 Класичний метод випадкового пошуку екстремуму

Основа методу випадкового пошуку [6-8] полягає у тому, що для пошуку кожноі наступноі точки на гіперповерхні рішень задачі зсуваються у випадковому напрямку. Незважаючи на це, метод дозволяє покроково пересуватися все ближче до цільового екстремуму. Спочатку спробу роблять в деякій початковій точці, після цього роблять крок у випадковому напрямку, порівнюють значення функції, і, якщо потрібно знайти максимум, роблять крок у напрямку зростання цільовоі функції. Після цього роблять спробу в ще одній точці, і знову напрям для кроку обирають випадково. Це проілюстровано візуально на рисунку 1.1:

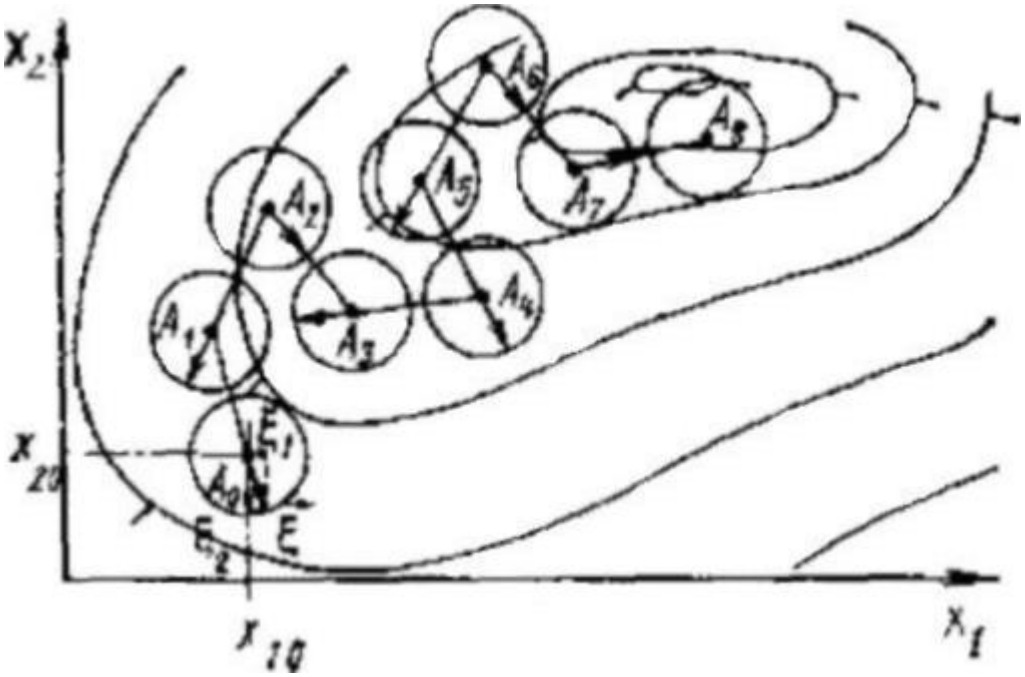


Рисунок 1.1. Ілюстрація до методу випадкового пошуку

Схема роботи методу випадкового пошуку наступна:

1. Обирають точку X_0 і розмір вектору Δ так, що довжина вектору дорвнює розміру кроку.

2. Визначають вектор ΔX так, що його початок лежить у точці X_0, а кінець рівномірно розподілений на поверхні гіперсфери з радіусом, що дорівнює довжині вектору. Після цього за допомогою послідовності випадкових чисел поступово отримують координати кінця вектору.

3. У початковій точці та в точці кінця вектору обчислюють значення функції. Якщо у другій точці воно більше, у цю точку потрбно зробити робочий крок. Якщо менше, роблять зворотній робочий крок. При цьому розмір робочого кроку може бути більшим за довжину вектору.

4. Початкову точку замінюють отриманою і повторюють кроки 2-3.

5. Якщо раптом значення функції в попередній та наступній точках виявиться однаковим, напрям кроку – прямий чи зворотній – обирають випадково.

6. Якщо у всіх напрямках від деякоі точки значення функціі менше, ніж у цій точці, скоріш за все, вона близька до екстремуму. У цій ситуаціі слід обійти декілька невипадкових точок, щоб бути впевненим, що центр цієі гіперсфери справді є екстремумом [6-8].

Сильні сторони методу:

1. Вибір напряму кроку не залежить від випадкових перешкод.

2. Легкість автоматизаціі.

3. Можливість доповнення методу алгоритмом самонавчання, завдяки чому ефективність стає значно більшою.

4. Чим більше число вимірів, тим більше ефективність методу.

Слабкі сторони методу:

1. Напрям руху за графіком не завжди оптимальний.

2. При відсутності автоматизованоі програми із самонавчанням ефективність методу значно знижується.

3. Ефективність знижується на пологих поверхнях [6-8].

1.3.2 Метод випадкового пошуку з послідовною редукцією області дослідження

Цей метод [9], який також має назву «метод Лууса-Якола» (Luus-Jakola, LJ), може оптимізувати цільову функцію $f(x_1, x_2, \cdots, x_n)$, яка визначена в області допустимих значень *D*, і дозволяє знайти найменший з усіх мінімумів на множині *D*, тобто таку точку $x^* \in D$, що

$$f(x^*) = \min_{x \in D} f(x)$$

Знайти максимум можливо, тільки якщо змінити знак на протилежний спочатку у функціі, а після цього у отриманого екстремуму.

Цей метод використовує змінну область пошуку та виконує іі редукцію і відновлення.

Спочатку обирається початкова точка і розмір області пошуку. Якщо явно не указано інше, початкова точка ставиться приблизно у центрі області допустимих значень, або, якщо вона безлімітна, у довільній точці.

На кожному кроці обираються декілька дочірніх точок з випадковими координатами в межах області пошуку. Якщо якась точка вийшла за межі області допустимих значень, її координата зсувається у відповідному напрямі.

В множині дочірніх точок обирають ту, в якій значення цільовоі функціі екстремальне. Ця точка є початковою для наступного кроку [9].

Після кожного кроку виконується редукція області пошуку. Після деякоі кількості кроків настає кінець етапу і перевірка умов неефективності пошуку.

Перед наступним етапом область пошуку знову збільшується. Множники на редукцію і відновлення підбираються так, щоб область пошуку злегка зменшувалась на кожному етапі. Це відновлення дозволяє зменшити сходимість до точки локального екстремуму.

Процесс триває задану кількість етапів або доки пошук ефективний.

На рисунку 1.2 наведена загальна схема методу випадкового пошуку з послідовною редукцією області дослідження [9].

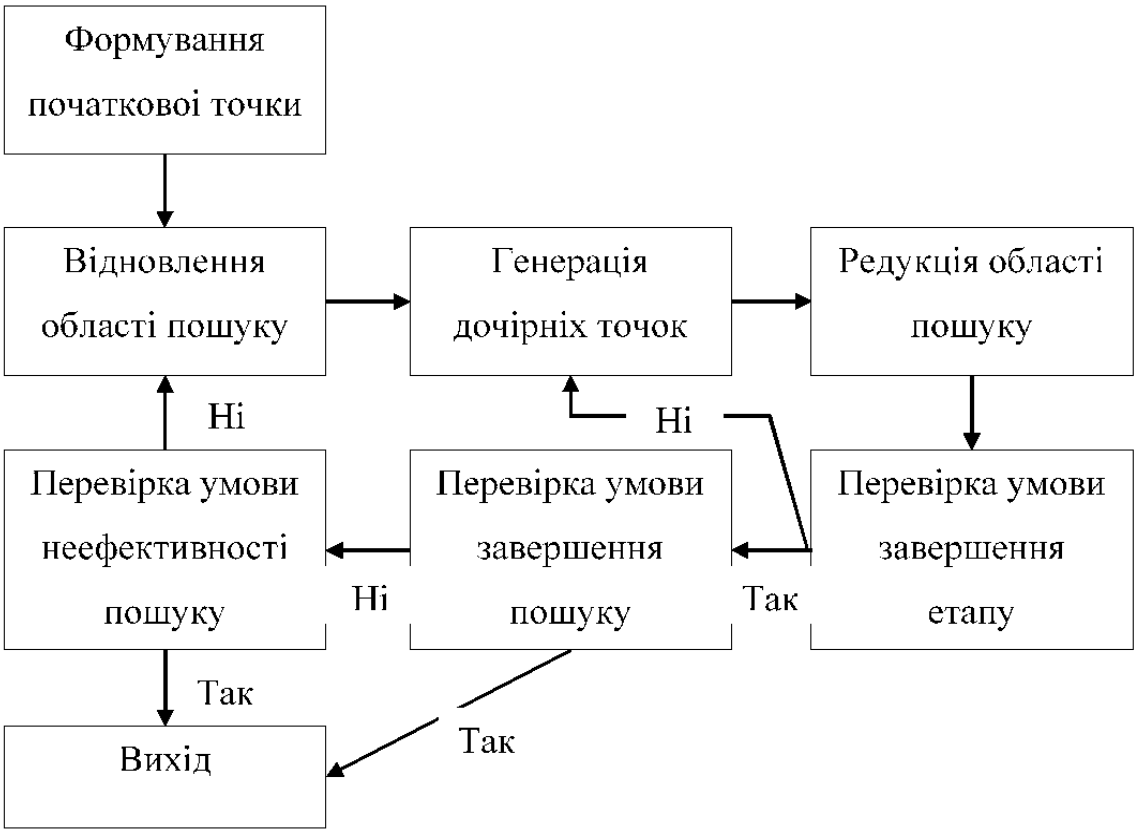


Рисунок 1.2. Схема методу випадкового пошуку з послідовною редукцією області дослідження

1.3.3 Жадібний адаптивний метод випадкового пошуку

Цей метод [7, 10] також оптимізує цільову функцію $F(x_1, x_2, \dots, x_n)$, задану на множині $x \in X$, і також дозволяє знайти лише мінімум, а задача пошуку максимуму зводиться до такої самої підстановки, як і в попередньому методі.

Цей метод використовує двофазний багаторазовий пошук рішення:

1. Фаза конструювання.
2. Фаза локального пошуку.

Рішення задач приблизно дорівнює найкращій з знайдених точок.

На першій фазі знаходяться більш-менш нормальні рішення, які стають основою для другоі фази. Після цього з отриманих точок починається перша фаза, і все продовжується.

Спочатку на першій фазі виконується пошук паралельно до вісей координат. Формується список найперспективніших напрямів, з яких випадково обирають один. Знайдена точка стає початковою для наступного етапу, координата фіксується та виключається з списка напрямів. Процесс повторюють, доки не знайдуть вектор *x* повністю [7, 10].

Фаза локального пошуку включає пошук в $3^n - 1$ можливих напрямах, як показано на рисунку 1.3:

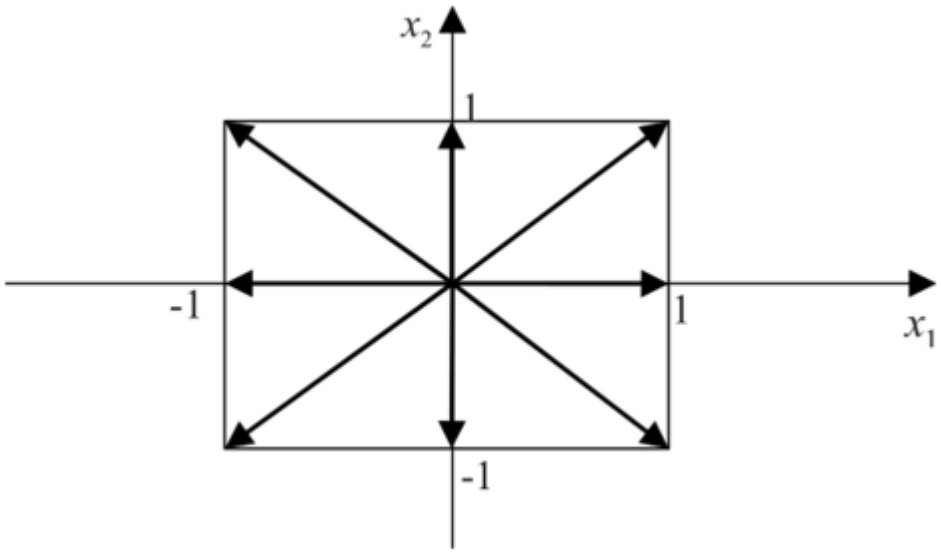


Рисунок 1.3. Фаза локального пошуку у жадібному адаптивному методі випадкового пошуку

Оскільки це число при великій кількості вісей координат може виявитися дуже великим, можна обмежити максимальну кількість напрямів до якогось заздалегідь визначеного числа [7, 10].

2 ВИБІР МЕТОДУ РІШЕННЯ

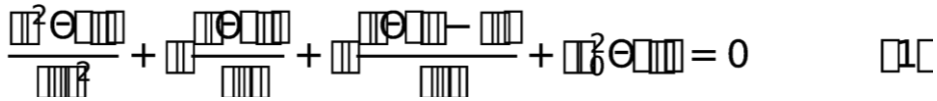
2.1 Постановка задачі

Розробити алгоритм і комп'ютерну програму для моделювання перехідних процесів, які виникають при роботі системи стабілізації вертикального положення корпусу корабля внаслідок хвилювання води. Використати комп'ютерну модель для знаходження оптимальних параметрів системи стабілізації. В якості критерію оптимальності використати тривалість затухання власних коливань корпусу корабля відносно вертикалі, яка проходить через центр ваги.

Оптимальні параметри системи повинні забезпечити найменшу тривалість затухання коливань. При моделюванні врахувати зміну сили, яка давить на борт, при відхиленні корпусу від вертикалі.

2.1.1 Математична постановка задачі

В наступній формулі [11] наведено дифференційне рівняння системи, що зменшує коливання корпусу корабля в вертикальній площині:



(1)

- де t – час;
- T – затримка у часі;
- $\theta(t)$ – кут відхилення корпусу від вертикалі;
- a, b, k_0^2 – параметри системи стабілізації.

При відхиленні змінюються як сила, що діє на корабль, так і площа, на яку давить хвиля. Тому в рівняння (1) необхідно внести зміни, щоб коефіцієнт при $\theta(t)$ змінювався у часі залежно від $\theta(t)$.

Якщо корабль стоїть вертикально, хвиля давить під ∠90°. Якщо виникає відхилення на ∠α, параметр ω_0^2 описується наступним рівнянням [11]:

$$\frac{d^2\theta(t)}{dt^2} + \frac{d\theta(t)}{dt} + \frac{d\theta(t) - d\tau}{dt} + \omega_0^2 \left(\frac{\pi}{2} - c\alpha\theta(t) + \theta(t)\right) = 0 \qquad (3)$$

Параметри a, ω_0^2 залежать від конструкції корпусу корабля, b – лише від системи стабілізації.

В результаті цього задача зводиться до моделювання процесу для заданих початкових умов $\theta_0, \theta(\theta_0), \frac{d\theta(\theta_0)}{dt}$ при різних значеннях параметрів a і b. При цьому c і ω_0^2 будемо вважати фіксованими.

Значення a і b, на яких коливання корпусу затухають найшвидше, будемо вважати оптимальними.

2.2 Вибір методу рішення задачі

В дифференційному рівнянні (3) є складова з затримкою у часі T. Це ускладнює аналітичне рішення задачі Коші. Тому використовується чисельний метод Рунге-Кутта 4-го порядку.

Оптимальні значення коефіцієнтів a і b знаходяться методом простого випадкового пошуку. Для цього завдаються інтервали, в яких можуть знаходитися їх оптимальні значення. В цих інтервалах обираються випадкові значення a і b, які підставляються в (3).

Вирішується задача Коші. Якщо коливання затухають, то визначається, протягом якого часу.

2.3 Хід рішення задачі

Від одного диференційного рівняння (3) другого порядку переходимо до системи з двох рівнянь першого порядку. Для цього позначимо:

$$y_0 = \frac{\Theta}{\omega} \qquad (4)$$

$$y_1 = \Theta \qquad (5)$$

З урахуванням (4) і (5) система рівнянь має вид:

$$\frac{dy_0}{dt} = -\omega y_0 \cos(\omega t) - \omega y_0 \sin(\omega t) - \sin(\omega t) - \frac{t}{2} - y_1 \cos(y_1 \sin(\omega t))$$

$$\frac{dy_1}{dt} = y_0$$

При заданих початкових умовах $y_0(t_0) = y_{00}$, $y_1(t_0) = y_{10}$ і заданих t_0, t_1, t_2, t_3 потрібно отримати $y_1(t_3)$.

Час t змінюється вд t_0 до t_3 з кроком h.

3 ПРОЕКТУВАННЯ І РЕАЛІЗАЦІЯ

3.1 Таблиця ідентифікаторів

В таблиці 3.1 приведені ідентифікатори, які вживаються при описанні алгоритму.

Таблиця 3.1. Ідентифікатори програми

Позначення у формулах	**Ідентифікатор**	**Пояснення**
a, b, R	a, b, R	Параметри системи
[illegible]	t, t0, h, tk	Час, його початкове значення, крок зміни і кінцеве значення
[illegible]	y1, y1_mas	Кут відхилення і масив значень
—	amin, amax, bmin, bmax	Інтервали коефіцієнтів a і b
[illegible]	y0, y0_mas	Похідні кута відхилення
[illegible]	y0_tau, y0_tau_mas	Похідні з зсувом в часі T
—	N	Кількість замірів
[illegible]	y00, y10	Початкові умови
—	k1, k2, k3, k4, g1, g2, g3, g4	Коефіцієнти в рівняннях
—	a_opt, b_opt	Оптимальні значення a і b
—	trivalist	Тривалість перехідного процесу
—	trivalist_opt	Тривалість при оптимальних a і b

—	umova	Якщо амплітуда коливань по модулю менше умови, то вважати, що перехідний процес закінчився
—	pr_pereh	Признак, що корпус почав відхилятися в протилежний бік
—	y1_poper, y1_potoch	Попереднй і поточний кути відхилень
—	kil_sprob	Кількість спроб з розігруванням a і b
—	kil_opt	Номер спроби, при якому було отримано найменший термін коливань
—	kil	Номер поточноі спроби

3.2 Опис процедур і функцій

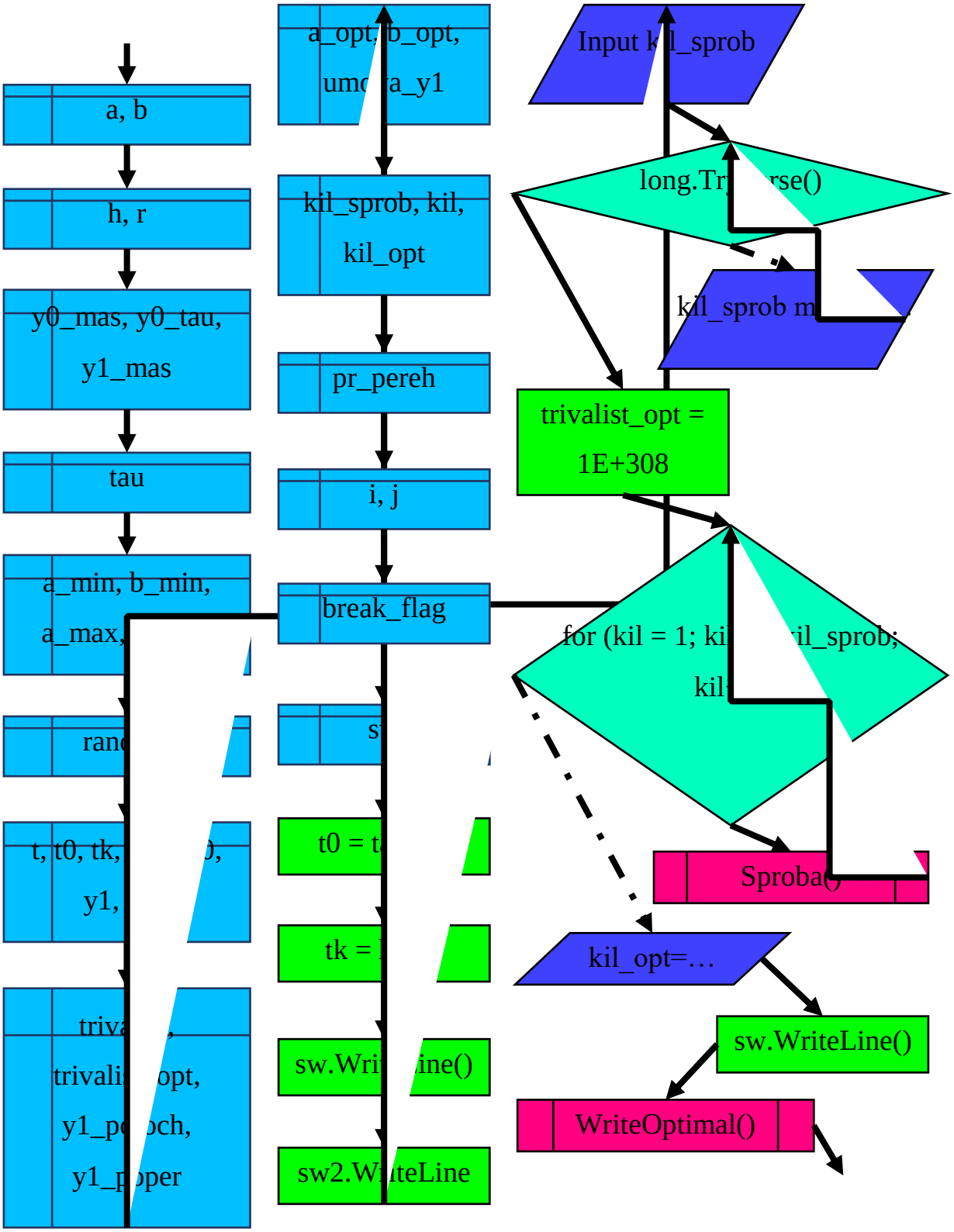
В таблиці 3.2 приведено опис використаних в програмі процедур і функцій.

Таблиця 3.2. Продедури і функціі програми

Процедура	**Опис**
Main()	Функція Main. В цьому місці починається і закінчується виконання програми.
Sproba()	Кожна з спроб знайти оптимальну тривалість
Iteration()	Кожна з ітерацій методу випадкового пошуку
KoshiProblem()	Чисельний метод вирішення задачі Коші

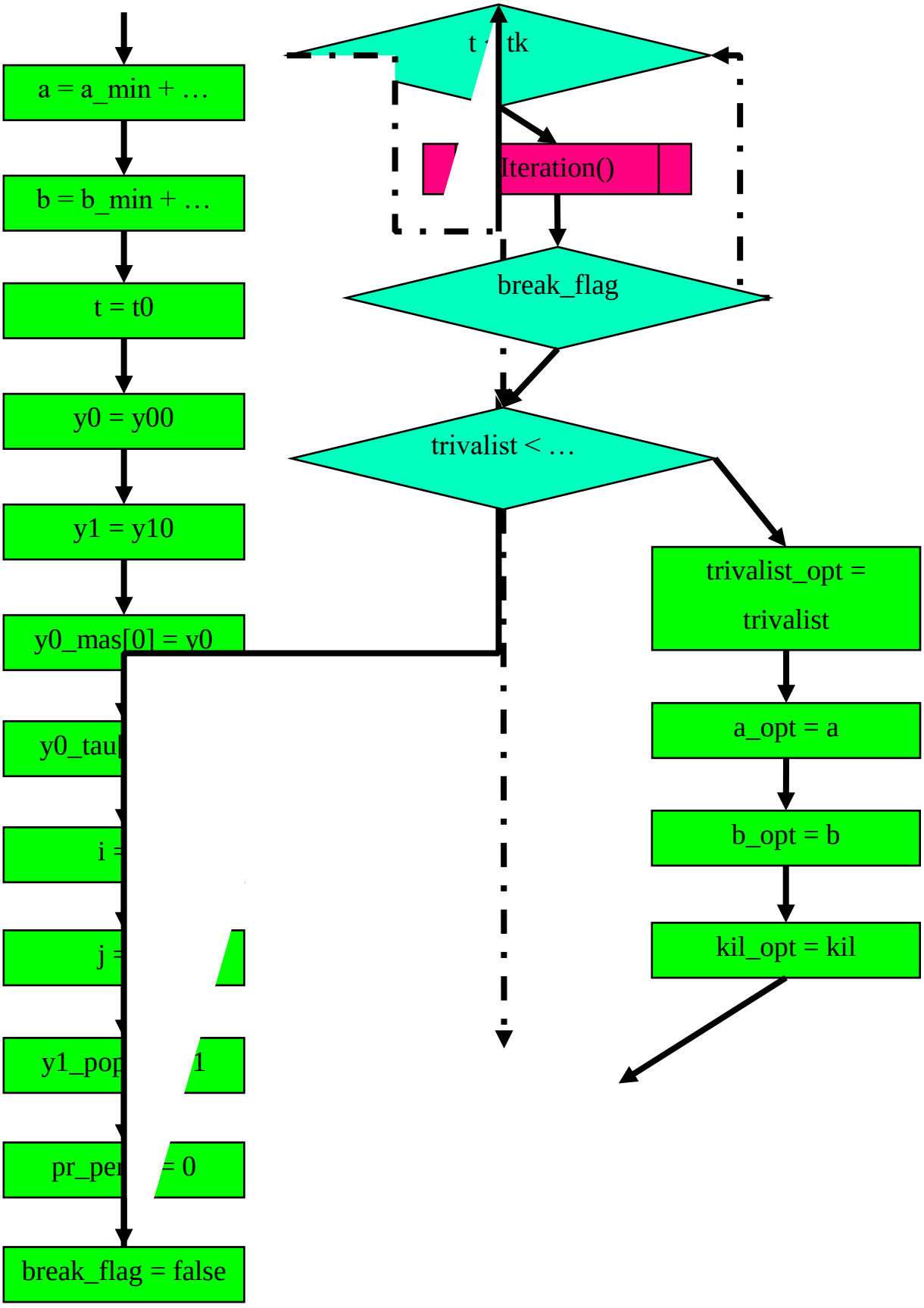
F(double t, double y0, double y1)	Функція обчислення дифференційного рівняння
WriteOptimal()	Функція запису в файл процесу оптимального рішення

3.3 Блок-схеми



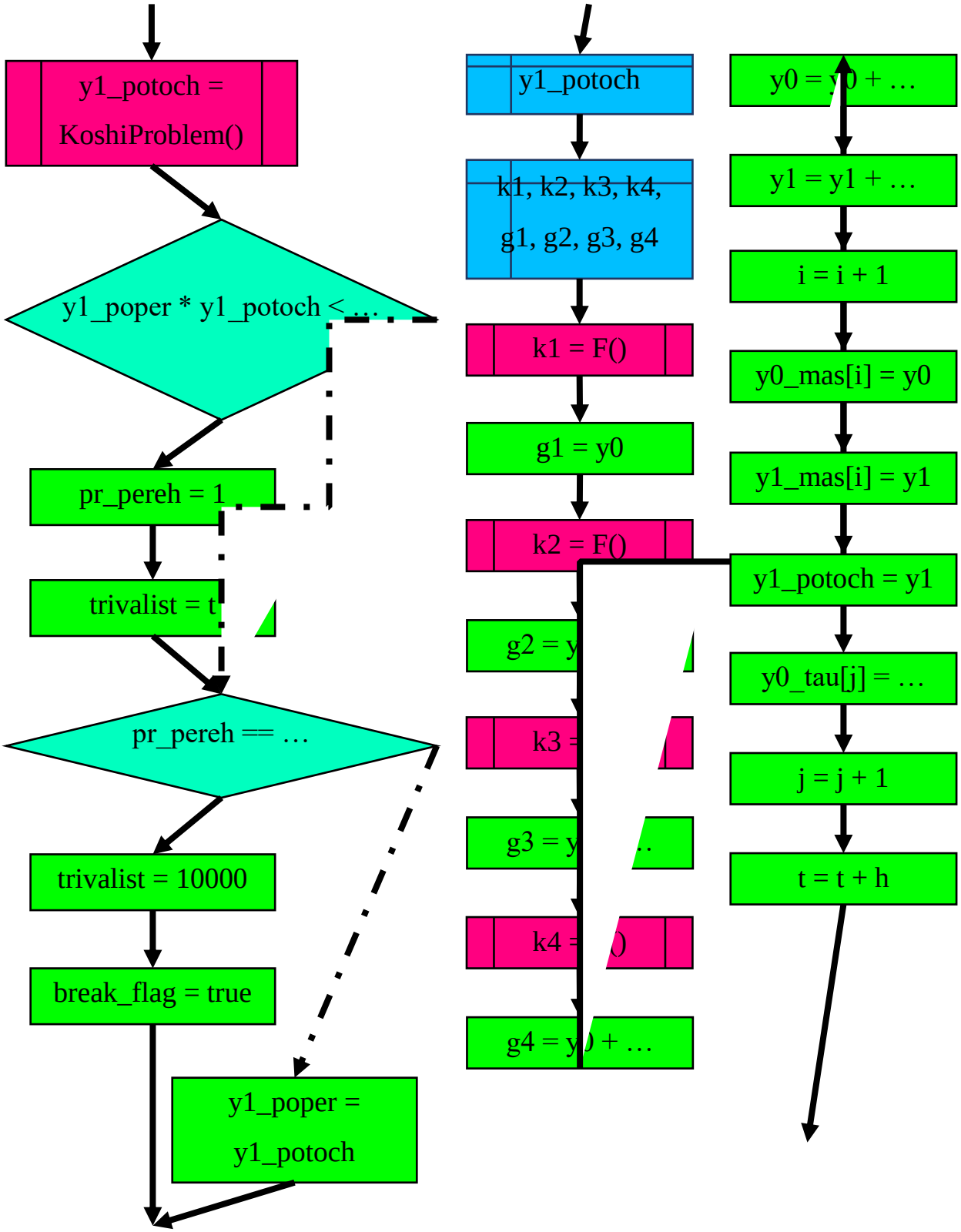
Початок

Кінець


a = a_min + …
b = b_min + …
t = t0
y0 = y00
y1 = y10
y0_mas[0] = y0
y0_tau
i =
j =
y1_pop
1
pr_per
= 0
break_flag = false
t
tk
Iteration()
break_flag
trivalist < …
trivalist_opt =
trivalist
a_opt = a
b_opt = b
kil_opt = kil

Sproba()

return

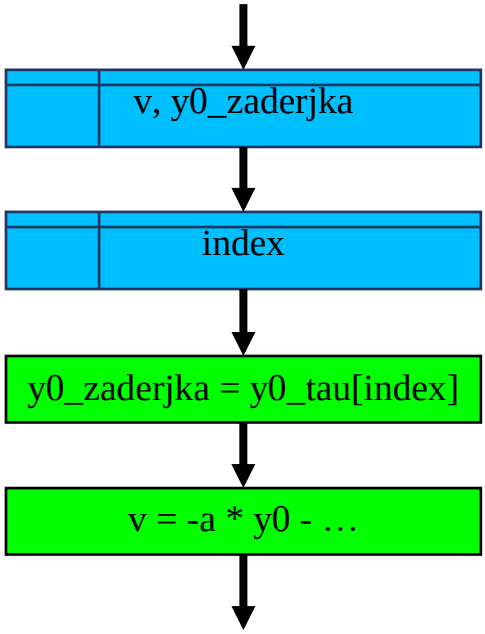

y1_potoch = KoshiProblem()
y1_poper * y1_potoch < …
pr_pereh = 1
trivalist = t
pr_pereh == …
trivalist = 10000
break_flag = true
y1_poper = y1_potoch
y1_potoch
k1, k2, k3, k4, g1, g2, g3, g4
k1 = F()
g1 = y0
k2 = F()
g2 = y
k3 =
g3 = y
k4 =
g4 = y
y0 = y0 + …
y1 = y1 + …
i = i + 1
y0_mas[i] = y0
y1_mas[i] = y1
y1_potoch = y1
y0_tau[j] = …
j = j + 1
t = t + h

Iteration()

return

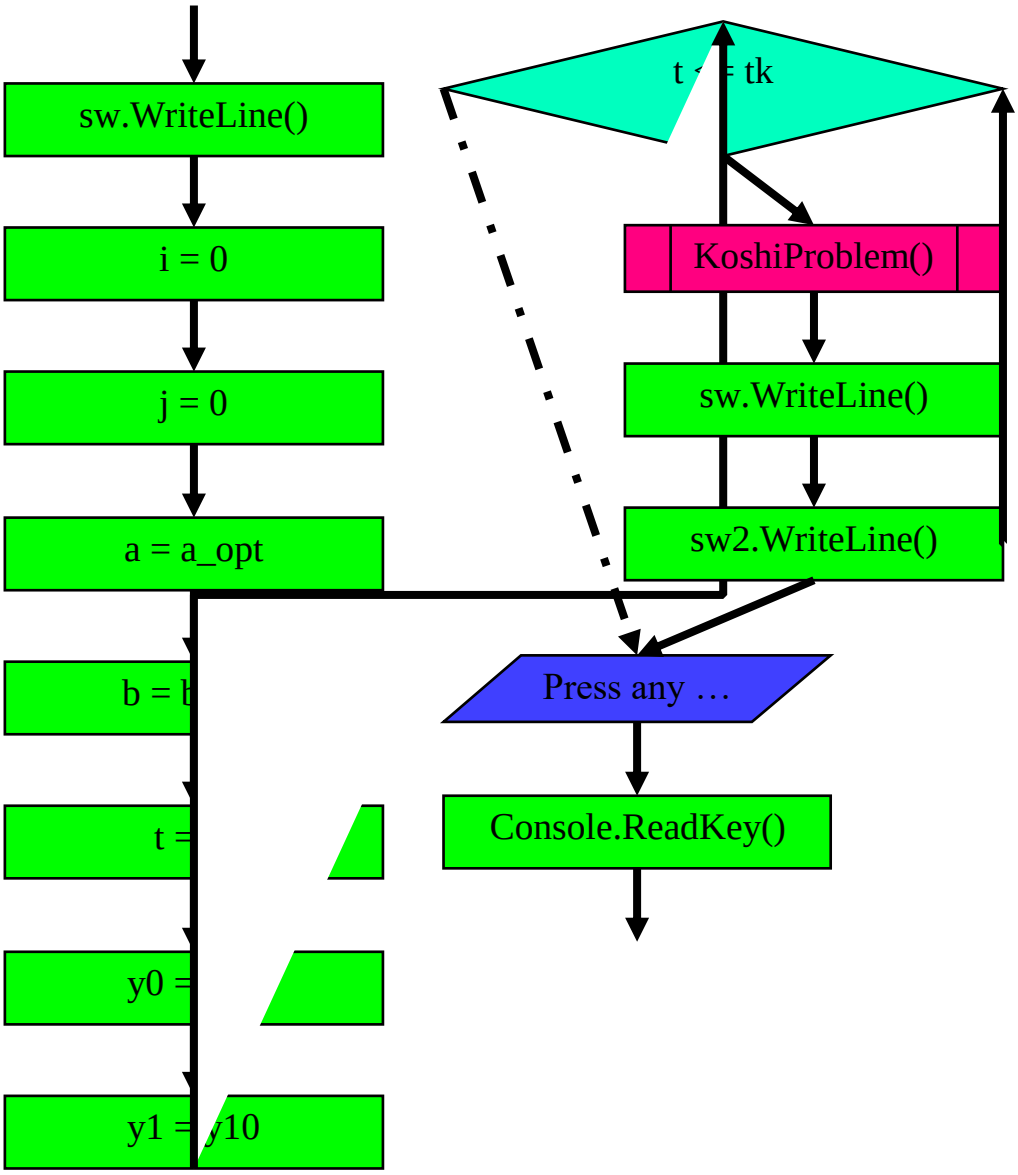
KoshiProblem()

return y1_potoch



F()

return v



WriteOptimal()

return

Наведені блок-схеми ілюструють алгоритм роботи програми.

3.4 Інструкція для користувача

Програма розроблена в середовищі Visual Studio 16.9 і запускається в операційній системі Windows 7, Windows 8, Windows 8.1 або Windows 10.

Виконуваний файл UndergraduatePractice.exe займає 10 мегабайт оперативноі пам’яті. Дані вводяться із клавіатури в режимі діалога. Передбачено встановлення даних по замовчуванні для тестового режиму роботи.

Результати обчислення виводяться на екран.

Склад програми у вихідному коді:

1) UndergraduatePractice.sln – рішення Visual Studio 16.9 (1.11 КБ).

2) UndergraduatePractice.csproj – проект Visual Studio на C# (171 байт).

3) Program.cs – вихідний файл C# (4.82 КБ).

3.5 Контрольний приклад

Вхідні дані для тестового режиму роботи програми приведені в таблиці 3.3.

Таблиця 3.3. Приклад вхідних даних

Параметр	**Значення**
N	500
h	0.05
R	0.5
a_min	0.1
b_min	0.01
a_max	10
b_max	3
y00	0.1
y10	0.5
kil_sprob	10 000
kil_opt	149
trivalist_opt	5.85
a_opt	1.359593

b_opt	0.206553

3.6 Результати

В таблиці 3.4 приведені результати пошуку оптимальних параметрів системи стабілізації в залежності від кількості спроб при випадковому пошуку.

Таблиця 3.4. Результати

kil_sprob	**kil_opt**	**trivalist_opt**	**a_opt**	**b_opt**
100	52	13.55	1.381354	0.298514
1000	149	5.85	1.359593	0.206553
10 000	149	5.85	1.359593	0.206553
20 000	16 699	5.55	1.257882	0.333665
200 000	118 392	5.45	1.180731	0.495452
1000 000	953 165	5.25	0.411802	1.989402

Аналіз отриманих результатів свідчить, що алгоритм і програма працюють правильно. Із зростанням кількості спроб знаходяться кращі значення параметрів системи, що призводить до зменшення тривалості перехідного процесу. При максимальній кількості спроб досягнута тривалість 5.25, яка зменшилася порівняно з початковою в два з половиною рази і практично не змінюється із збільшенням кількості спроб.

Таким чином можна вважати, що оптимальними значеннями параметрів системи стабілізації вертикального стану корабля є a=0.411802, b=1.989402. При цих параметрах перехідний процес триває 5.25 с.

ВИСНОВКИ

Розроблено алгоритм і комп’ютерну програму для знаходження оптимальних параметрів системи автоматичної стабілізації корпусу корабля при хвилюванні. Робота програми перевірена на контрольному прикладі. Вона може знайти практичне застосування при проектуванні систем стабілізації корпусу корабля.

СПИСОК ЛІТЕРАТУРИ

1. Основы теории оптимизации. Книга 2: Безусловная оптимизация. Ч. 1: http://ecat.diit.edu.ua/ft/Optimization2_1.pdf

2. История становления и развития теории оптимизации: https://studme.org/183569/matematika_himiya_fizik/istoriya_stanovleniya_razvitiya_teorii_optimizatsii

3. Википедия – свободная энциклопедия: https://ru.wikipedia.org/wiki

4. Без назви: http://tc.nsu.ru/uploads/met-opt-pr-zad.pdf

5. Постановка задачи оптимизации и численные методы ее решения: https://hub.exponenta.ru/post/postanovka-zadachi-optimizatsii-i-chislennye-metody-ee-resheniya356

6. Метод случайного поиска: https://studref.com/515892/tehnika/metod_sluchaynogo_poiska.

7. Анализ методов случайного поиска глобальных экстремумов многомерных функций: https://www.fundamental-research.ru/ru/article/view?id=4706.

8. Без назви: https://www.math.spbu.ru/user/gran/sb3/sushkov_kusher.pdf.

9. Метод случайного поиска с последовательной редукцией области исследования: https://studref.com/544108/matematika_himiya_fizik/primenenie_metoda_sluchaynogo_poiska_posledovatelnoy_reduktsiey_oblasti_issledovaniya#744.

10. Жадный адаптивный метод случайного поиска: https://studref.com/544113/matematika_himiya_fizik/primenenie_zhadnogo_adaptivnogo_metoda_sluchaynogo_poiska#211

11. В. Каннингхэм. Введение в теорию нелинейных систем: https://www.twirpx.com/file/1918654

12. Б. Т. Поляк. Введение в оптимизацию (384 стор.) (URL невідомий).

ДОДАТОК. КОД ПРОГРАМИ

```
using System;
using System.IO;

namespace UndergraduatePractice
{
class Program
{
        private static readonly Random random = new(1234567890);

        private static int Main()
        {
                Console.WriteLine("Input N or press Enter to set by default (500):");
                long N = long.TryParse(Console.ReadLine(), out long n) ? n : 500;
                double a = 0, b = 0;
                Console.WriteLine("Input h or press Enter to set by default (0.05):");
                double h = double.TryParse(Console.ReadLine(), out double n2) ? n2 : 0.05;
                Console.WriteLine("Input R or press Enter to set by default (0.5):");
                double R = double.TryParse(Console.ReadLine(), out double n3) ? n3 : 0.5;
                double[] y0_mas = new double[N], y0_tau = new double[2 * N], y1_mas = new double[N];
```

```
            int tau = 10;
            Console.WriteLine("Input a_min or press Enter to set by default
(0.1):");
            double a_min = double.TryParse(Console.ReadLine(), out
double n4) ? n4 : 0.1;
            Console.WriteLine("Input b_min or press Enter to set by default
(0.01):");
            double b_min = double.TryParse(Console.ReadLine(), out
double n5) ? n5 : 0.01;
            Console.WriteLine("Input a_max or press Enter to set by default
(10):");
            double a_max = double.TryParse(Console.ReadLine(), out
double n6) ? n6 : 10;
            Console.WriteLine("Input b_max or press Enter to set by default
(3):");
            double b_max = double.TryParse(Console.ReadLine(), out
double n7) ? n7 : 3;
            Console.WriteLine("Input y00 or press Enter to set by default
(0.1):");
            double t, t0, tk, y0, y00 = double.TryParse(Console.ReadLine(),
out double n8) ? n8 : 0.1;
            Console.WriteLine("Input y10 or press Enter to set by default
(0.5):");
            double y1, y10 = double.TryParse(Console.ReadLine(), out
double n9) ? n9 : 0.5;
            double trivalist = 0, trivalist_opt, y1_potoch, y1_poper;
            double a_opt = 0, b_opt = 0, umova_y1 = 0.005;
            long kil_sprob, kil, kil_opt = 0;
            int pr_pereh;
```

```
int i, j;
bool break_flag;
StreamWriter sw = new("optsudno.txt"), sw2 = new("sudnoexl.txt");

t0 = tau * h;
tk = h * N;
sw.WriteLine("tau={0} a={1} b={2} R={3}\n", tau, a, b, R);
sw2.WriteLine("{0} {1} {2}", t0, y00, y10);
Console.WriteLine("Input kil_sprob");
while (!long.TryParse(Console.ReadLine(), out kil_sprob))
{
    Console.WriteLine("kil_sprob must be an integer number!");
}
trivalist_opt = 1E+308;

for (kil = 1; kil <= kil_sprob; kil++) Sproba();

Console.WriteLine("kil_opt={0} trivalist_opt={1} a_opt={2} b_opt={3}", kil_opt, trivalist_opt, a_opt, b_opt);
sw.WriteLine("kil_opt={0} trivalist_opt={1} a_opt={2} b_opt={3}", kil_opt, trivalist_opt, a_opt, b_opt);

WriteOptimal();
return 0;

void Sproba()
{
```

```
        a = a_min + (a_max - a_min) * random.NextDouble();
        b = b_min + (b_max - b_min) * random.NextDouble();
        t = t0;
        y0 = y00;
        y1 = y10;
        y0_mas[0] = y0;
        y0_tau[0] = y0;
        i = 0;
        j = 1;
        y1_poper = y1;
        pr_pereh = 0;
        break_flag = false;
        while (t < tk)
        {
            Iteration();
            if (break_flag) break;
        }

        if (trivalist < trivalist_opt && pr_pereh == 1)
        {
            trivalist_opt = trivalist;
            a_opt = a;
            b_opt = b;
            kil_opt = kil;
        }
}

void Iteration()
{
```

```
            y1_potoch = KoshiProblem();
            if (y1_poper * y1_potoch < 0 && pr_pereh == 0)
            {
                pr_pereh = 1;
                trivalist = t;
            }
            if (pr_pereh == 1 && Math.Abs(y1_potoch) > umova_y1)
            {
                trivalist = 10000;
                break_flag = true;
                return;
            }
            y1_poper = y1_potoch;
        }

        double KoshiProblem()
        {
            double y1_potoch;
            double k1, k2, k3, k4, g1, g2, g3, g4;
            k1 = F(t, y0, y1);
            g1 = y0;
            k2 = F(t + h / 2, y0 + h * k1 / 2, y1 + h * g1 / 2);
            g2 = y0 + h * k1 / 2;
            k3 = F(t + h / 2, y0 + h * k2 / 2, y1 + h * g2 / 2);
            g3 = y0 + h * k2 / 2;
            k4 = F(t + h, y0 + h * k3, y1 + h * g3);
            g4 = y0 + h * k3;
            y0 += h * (k1 + 2 * k2 + 2 * k3 + k4) / 6;
            y1 += h * (g1 + 2 * g2 + 2 * g3 + g4) / 6;
```

```
                i++;
                y0_mas[i] = y0;
                y1_mas[i] = y1;
                y1_potoch = y1;
                y0_tau[j] = y0_mas[i - 1] + (double)(y0_mas[i] - y0_mas[i
- 1]) / 2;
                j++;
                t += h;
                return y1_potoch;
        }

        double F(double t, double y0, double y1)
        {
                double v, y0_zaderjka;
                int index = (int)(2 * (t - tau * h) / h);
                y0_zaderjka = y0_tau[index];
                v = -a * y0 - b * y0_zaderjka - R * (1.570795 - 0.2 * y1) *
y1;
                return v;
        }

        void WriteOptimal()
        {
                sw.WriteLine("For opt a_opt and b_opt");
                i = 0;
                j = 0;
                a = a_opt;
                b = b_opt;
                t = t0;
```

```
                y0 = y00;
                y1 = y10;
                while (t <= tk)
                {
                    KoshiProblem();
                    sw.WriteLine("t={0} y0={1}  y1={2}", t, y0, y1);
                    sw2.WriteLine("{0} {1} {2}", t, y0, y1);
                }
                Console.WriteLine("Press any key to close...");
                Console.ReadKey(true);
            }
        }
    }
}
```