**Титульна сторінка**

**Анотація**

Дослідження присвячено розробці алгоритму оптимізації витрат авіаційного палива з урахуванням впливу атмосферних умов, зокрема відхилень від міжнародної стандартної атмосфери (ISA). Метою роботи є підвищення ефективності планування польотів шляхом динамічного вибору висоти на основі аналізу температурних аномалій, маси повітряного судна та метеорологічних даних. На основі математичного моделювання взаємодії аеродинамічних параметрів (TAS, IAS), маси літака та зовнішніх факторів запропоновано ітеративний підхід, що передбачає сегментацію маршруту.

Результати демонструють зниження витрат палива для Boeing 738 за рахунок оптимізації висоти та врахування ISA-відхилень. Практична цінність роботи полягає у можливості застосування алгоритму в системах управління польотами (FMS) для автоматизації розрахунків, що особливо актуально в умовах зростання вимог до екологічної та економічної ефективності авіаперевезень.

**Abstract**

The study focuses on developing an algorithm for optimizing aircraft fuel consumption while accounting for atmospheric conditions, particularly deviations from the International Standard Atmosphere (ISA). The research aims to enhance flight planning efficiency through dynamic altitude selection based on analysis of temperature anomalies, aircraft mass, and meteorological data. Using mathematical modeling of interactions between aerodynamic parameters (TAS, IAS), aircraft mass and external factors, the study proposes an iterative approach involving route segmentation.

Results demonstrate reduction in fuel consumption for Boeing 738 through altitude optimization and ISA deviation adjustments. The practical value lies in the algorithm's potential application in Flight Management Systems (FMS) to automate calculations, which is particularly relevant given increasing demands for environmental and economic efficiency in air transport.

**Зміст**

[Вступ 4](#_Toc199578125)

[РОЗДІЛ 1. АНАЛІТИЧНИЙ ОГЛЯД ЛІТЕРАТУРИ 6](#_Toc199578126)

[1.1 Міжнародний стандарт атмосфери (ISA) та його вплив 6](#_Toc199578127)

[1.2 Моделювання льотних характеристик 8](#_Toc199578128)

[1.3 Використання інтерполяції 8](#_Toc199578129)

[1.4 Програмні засоби 9](#_Toc199578130)

[1.5 Перспективи досліджень 10](#_Toc199578131)

[РОЗДІЛ 2. МЕТОДИКА ДОСЛІДЖЕНЬ 12](#_Toc199578132)

[2.1 Формалізація моделі набору висоти та польоту 12](#_Toc199578133)

[2.2 Використання даних про ISA deviation 14](#_Toc199578134)

[2.3 Розробка алгоритму оптимізації висоти польоту 15](#_Toc199578135)

[2.4 Аналіз та верифікація моделі 20](#_Toc199578136)

[РОЗДІЛ 3. ВИКЛАД РЕЗУЛЬТАТІВ ДОСЛІДЖЕНЬ І ОПИС З’ЯСОВАНИХ ФАКТІВ 21](#_Toc199578137)

[3.1. Вступ до результатів досліджень 21](#_Toc199578138)

[3.2. Методологія тестування 21](#_Toc199578139)

[3.3. Тестові сценарії та результати 22](#_Toc199578140)

[3.4. Графічний аналіз результатів 26](#_Toc199578141)

[3.5. Підсумок аналізу 28](#_Toc199578142)

[РОЗДІЛ 4. ОБГОВОРЕННЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ 30](#_Toc199578143)

[4.1 Порівняння отриманих даних із результатами інших досліджень 30](#_Toc199578144)

[4.2 Аналіз похибок і обмежень методики 31](#_Toc199578145)

[4.3 Перспективи вдосконалення методів 32](#_Toc199578146)

[Висновки 34](#_Toc199578147)

[Список літератури 39](#_Toc199578148)

[Додатки 40](#_Toc199578149)

[Додаток А Лістинг програми 41](#_Toc199578150)

# Вступ

У сучасній авіації ефективне управління польотом, зокрема вибір оптимальної висоти, є ключовим фактором для зниження витрат палива, зменшення часу перельоту та забезпечення екологічної стійкості. Особливу роль у цих розрахунках відіграють атмосферні умови, зокрема відхилення від стандартної атмосфери (ISA deviation), які впливають на аеродинамічні характеристики літака та споживання палива. Тому розробка інструменту для автоматизованого розрахунку оптимальної висоти польоту з урахуванням ISA deviation є актуальною задачею для авіаційної галузі.

**Актуальність теми.** Оптимізація висоти польоту дозволяє авіакомпаніям значно знизити витрати на паливо, що становить одну з найбільших статей витрат у комерційній авіації. Крім того, правильний вибір висоти впливає на тривалість польоту, зменшує навантаження на двигуни та покращує екологічні показники. Сучасні системи планування польотів враховують багато факторів, проте автоматизований підхід до вибору висоти з урахуванням ISA deviation, маси літака та інших параметрів дозволяє отримати більш точні результати порівняно з традиційними методами.

**Мета дослідження.** Метою даної роботи є розробка програмного забезпечення для комп’ютерного розрахунку оптимальної висоти польоту літака Boeing 738 з урахуванням ISA deviation, маси літака, маршруту та інших параметрів. Програма повинна визначати висоту, на якій витрата палива буде мінімальною, а також обчислювати загальну дистанцію, час польоту, кількість витраченого палива та кінцеву масу літака.

**Об’єкт дослідження.** Об’єктом дослідження є процес набору висоти та крейсерського польоту літака Boeing 738 у різних атмосферних умовах.

**Предмет дослідження.** Предметом дослідження є математичні моделі та алгоритми розрахунку витрати палива, інтерполяції за висотою, масою та ISA deviation, а також програмна реалізація цих алгоритмів у графічному інтерфейсі.

**Методи дослідження.** У роботі використано:

* методи лінійної інтерполяції для врахування змінної маси літака, ISA deviation та висоти польоту;
* алгоритми оптимізації для визначення найефективнішої висоти;
* геометричні розрахунки для визначення пройденої дистанції на основі координат маршруту;
* інструменти програмування для створення графічного інтерфейсу та обробки даних.

**Теоретичне та практичне значення.** Теоретична цінність роботи полягає у вдосконаленні методів розрахунку оптимальної висоти польоту з урахуванням динамічної зміни маси та атмосферних умов. Практичне значення полягає в тому, що розроблена програма може бути використана диспетчерами, пілотами та авіакомпаніями для планування польотів із мінімальними витратами палива.

**Наукова новизна.** У роботі запропоновано підхід до інтеграції бази даних повітряних трас, навігаційних точок та характеристик літака в єдину систему, що дозволяє автоматично визначати оптимальну висоту польоту з урахуванням ISA deviation. Також розроблено алгоритми інтерполяції, які підвищують точність розрахунків порівняно з табличними методами.

Дана робота спрямована на розробку інструменту для автоматизованого вибору оптимальної висоти польоту з урахуванням ISA deviation. Реалізована програма дозволяє зменшити витрати палива, підвищити ефективність польотів та може бути інтегрована в сучасні системи планування маршрутів.

# РОЗДІЛ 1. АНАЛІТИЧНИЙ ОГЛЯД ЛІТЕРАТУРИ

## Міжнародний стандарт атмосфери (ISA) та його вплив

Міжнародний стандарт атмосфери (ISA) — це уніфікована модель, яка описує ідеальні умови в атмосфері на різних висотах. Вона визначає, якими мають бути температура, тиск, густина повітря та інші параметри за відсутності будь-яких аномалій. Ця модель служить еталоном для розрахунків у авіації, дозволяючи порівнювати льотні характеристики літаків у різних умовах.

За стандартом ISA, температура на рівні моря приймається за +15°C, тиск — 1013,25 гПа, а густина повітря — 1,225 кг/м³. Зі збільшенням висоти температура знижується з певним градієнтом (приблизно 2°C на 1000 футів у тропосфері), а тиск і густина повітря також зменшуються. Ці параметри критично важливі для розрахунків аеродинаміки, тяги двигунів та витрати палива.

Реальні атмосферні умови рідко точно відповідають ISA. Відхилення виникають через сезонні коливання, географічні особливості, погодні явища тощо. Наприклад:

* **ISA-plus** (температура вища за стандартну) — призводить до розрідження повітря, що знижує підйомну силу крил і ефективність двигунів. У таких умовах літаку потрібна більша дистанція для зльоту, а набір висоти відбувається повільніше через зменшення тяги.
* **ISA-minus** (температура нижча за стандартну) — сприяє підвищенню густини повітря, що покращує аеродинамічні характеристики. Однак надто низькі температури можуть ускладнити роботу систем літака, наприклад, обмерзання.
* Висота FL - В авіації та авіаційній метеорології ешелоном польоту (flight level (FL) є умовна висота за стандартного тиску, що подається цифрою у сотнях футів. Тиск визначений, виходячи з даних тиску рівня моря за стандартної атмосфери в значенні 1013,2 гПа (760 мм або 29,92 дюймів ртутного стовпа), в зв'язку з чим не обов'язково відповідатиме справжній висоті польоту як над середнім рівнем моря, так і над поверхнею Землі.
* TAS Істинна швидкість (у авіації) — це швидкість повітряного судна відносно повітря, в якому воно рухається, з урахуванням поправки на зміну густини (щільності) повітря, залежно від барометричної висоти.
* Фунт (англ. pound) — одиниця ваги та маси, розповсюджена у різних країнах у різні часи.
* Інтерполяція — в обчислювальній математиці спосіб знаходження проміжних значень величини за наявним дискретним набором відомих значень.
* Тропопа́уза — порівняно тонкий шар у атмосфері планети, де припиняється зниження температури з висотою і вище якого атмосфера стає прозорою для теплового випромінювання. Термін застосовується для опису температурної стратифікації в моделях атмосфер.
* Повітряний простір — обшир повітря, котрий знаходиться під контролем окремої країни та розташований над її теренами (в міжнародно-визнаних кордонах), в тому числі її територіальними водами, або більш загально, будь-яка певна тривимірна частина атмосфери, юридично приналежна країні.
* Перформанс таблиці - це таблиці в яких для кожного типу повітряного судна вказують істинну повітряну швидкість, швидкість набору/зниження висоти та витрату палива за умов набору висоти, крейсерського польоту та зниження на різних ешелонах польоту.

Дослідження (наприклад, *ISA Aviation: Understanding its Impact on Aircraft Performance*) підкреслюють, що відхилення ISA безпосередньо впливають на:

1. **Витрату палива** — при ISA-plus двигуни споживають більше палива через зниження ККД.
2. **Динаміку набору висоти** — у спекотному кліматі літак може потребувати зменшення корисного навантаження для компенсації втрати тяги.
3. **Максимальну висоту польоту** — у разі значних відхилень ISA практична стеля літака знижується.

## Моделювання льотних характеристик

У сучасному авіаційному проєктуванні та симуляції польотів широко використовуються математичні та обчислювальні підходи для відтворення поведінки літака в різних умовах. Ці моделі враховують не лише аеродинамічні параметри, а й змінні зовнішні фактори, такі як вітер, атмосферний тиск і температура. Значна увага приділяється визначенню точок переходу між етапами польоту — наприклад, завершення набору висоти. При цьому критичними залишаються параметри, як-от залишкова маса, профіль підйому, доступна тяга та погодні умови.

Багато класичних моделей оперують ідеальними умовами атмосфери, нехтуючи варіаціями на зразок відхилення від стандартної атмосфери (ISA deviation). Такі спрощення знижують точність у передбаченні витрати палива чи швидкості набору висоти. Актуальні алгоритми моделювання намагаються враховувати більше змінних, у тому числі збурення атмосфери на різних рівнях польоту. У роботі використовується підхід, який не лише враховує ці параметри, але й адаптує маршрут відповідно до них, підвищуючи точність розрахунків і ефективність використання палива.

## Використання інтерполяції

Інтерполяція є критично важливим інструментом при роботі з таблицями характеристик літака, які зазвичай не містять усіх можливих значень маси, висоти чи атмосферних умов. Найчастіше застосовується лінійна інтерполяція — проста, але ефективна методика, яка дозволяє швидко отримати приблизні значення між відомими точками.

У рамках реалізованого програмного комплексу інтерполяція використовується в кількох напрямках. По-перше, вона застосовується до маси літака, оскільки вона постійно змінюється під час польоту внаслідок витрати палива. Розрахунок відбувається з певним кроком відстані, що дозволяє адаптувати масу в реальному часі. По-друге, інтерполяція необхідна для роботи з ISA deviation — користувач може ввести значення, яке не передбачене у характеристичних таблицях, тому потрібно оцінити наближене значення. Нарешті, висота польоту також вимагає інтерполяції, бо дані в таблицях представлені для обмеженої кількості ешелонів, тоді як фактична траєкторія літака може проходити між ними.

Таким чином, використання інтерполяційних підходів дозволяє адаптувати модель до непередбачуваних сценаріїв польоту, підвищуючи її гнучкість і точність.

## Програмні засоби

Сучасні дослідження в галузі авіаційного моделювання активно використовують програмні засоби для аналізу льотних характеристик. У вашому випадку, програма, розроблена на Python, є потужним інструментом для симуляції процесу набору висоти з урахуванням відхилень ISA (International Standard Atmosphere). Вона дозволяє враховувати ключові параметри, такі як початкова маса літака, маршрут (готовий або заданий вручну), навігаційні точки та значення ISA deviation.

Важливою частиною програми є база даних, яка містить координати повітряних трас, навігаційних точок, аеропортів, а також характеристичні таблиці для Boeing 738 на різних етапах польоту (зліт, круїз, посадка). Це дозволяє проводити точні розрахунки, враховуючи специфіку конкретного типу літака.

Для оптимізації витрати палива програма аналізує всі можливі висоти маршруту, використовуючи методи лінійної інтерполяції. Це дозволяє знайти оптимальну висоту, на якій витрата палива буде мінімальною. Інтерполяція проводиться за трьома параметрами: масою літака (яка зменшується під час польоту), ISA deviation (як статичне значення) та висотою польоту (оскільки характеристичні дані наведені з певним кроком).

Результати розрахунків включають оптимальну висоту, пройдену дистанцію, витрату палива, кінцеву масу літака та час польоту. Ці дані є критично важливими для планування маршрутів із мінімальними витратами та максимальною ефективністю.

## Перспективи досліджень

1. Інтеграція реальних метеорологічних даних

Наразі програма використовує статичне значення ISA deviation, але в майбутньому можна підключити API метеорологічних сервісів (наприклад, OpenWeatherMap або NOAA) для отримання актуальних даних про температуру, тиск та інші параметри атмосфери. Це дозволить робити розрахунки з урахуванням реальних погодних умов, що підвищить точність симуляції.

1. Покращення алгоритмів інтерполяції

Замість лінійної інтерполяції можна використовувати більш точні методи, або машинне навчання для прогнозування витрати палива. Це особливо актуально для складних маршрутів із нелінійними змінами параметрів.

1. Розширення бази даних

Додавання характеристик інших типів літаків (наприклад, Airbus A320 або Boeing 787) дозволить використовувати програму для аналізу різних моделей. Також варто розглянути можливість інтеграції даних про повітряні потоки, що впливають на витрату палива.

1. Оптимізація інтерфейсу

Додавання візуалізації маршруту (наприклад, за допомогою бібліотек Matplotlib або Plotly) полегшить аналіз результатів. Також можна реалізувати функцію збереження та порівняння різних сценаріїв польоту.

1. Валідація результатів

Для підвищення надійності програми варто провести тестування на реальних даних польотів і порівняти результати з існуючими авіаційними системами (наприклад, Boeing Performance Tool або Airbus FlySmart).

# РОЗДІЛ 2. МЕТОДИКА ДОСЛІДЖЕНЬ

Методика дослідження, представлена в цьому розділі, зосереджена на розробці та реалізації комп’ютерної програми для розрахунку оптимального процесу набору висоти літаком Boeing 738 з урахуванням відхилень від Міжнародної стандартної атмосфери (ISA deviation). Програма створена з використанням мови програмування Python і має графічний інтерфейс користувача для зручного введення даних та відображення результатів. Основна мета дослідження — створення інструменту, який дозволяє визначити оптимальну висоту польоту для мінімізації витрат палива на заданому маршруті з урахуванням атмосферних умов, маси літака та характеристик маршруту.

## 2.1 Формалізація моделі набору висоти та польоту

Для виконання поставлених завдань була розроблена математична модель, яка описує процес набору висоти та крейсерського польоту літака з урахуванням впливу відхилень від стандартної атмосфери (ISA deviation). Модель базується на фізичних принципах аеродинаміки, кінематики та залежності витрат палива від параметрів польоту. Основні аспекти моделі включають:

1. **Основні параметри моделі**:
   * **Маса літака**: Вхідний параметр, який задається користувачем на початку маршруту. Маса динамічно змінюється в процесі польоту через витрату палива.
   * **Маршрут польоту**: Складається з набору повітряних трас, навігаційних точок та аеропортів, які користувач може вибрати з бази даних програми або ввести вручну.
   * **ISA deviation**: Відхилення температури від стандартної атмосфери, яке впливає на аеродинамічні характеристики та витрати палива.

Формула для щільності повітря з урахуванням ISA deviation:

Зображення, що містить Шрифт, символ, типографія, дизайн

Вміст, створений ШІ, може бути неправильним.

ρ₀ — стандартна щільність повітря на рівні моря, яка становить 1,225 кг/м³

L — температурний градієнт, що дорівнює 0,0065 К/м

h — висота (в метрах)

T₀ — стандартна температура на рівні моря, 288,15 К

g — прискорення вільного падіння, 9,81 м/с²

M — молярна маса повітря, 0,028964 кг/моль

R — універсальна газова стала, 8,314 Дж/(моль·К)

Корекція температури для відхилення від стандартної атмосфери (ISA deviation):

**T = T\_ISA + ΔT**

де  
ΔT — це відхилення від стандартної температури (тобто наскільки фактична температура відрізняється від температури за стандартною атмосферою на певній висоті).

* + **Висота польоту**: Змінна, яка варіюється в межах доступних рівнів польоту (flight levels) для знаходження оптимального значення.
  + **Витрата палива**: Залежить від маси літака, висоти, швидкості та атмосферних умов, розраховується на основі характеристичних таблиць літака Boeing 738.

1. **Математична основа**: Модель використовує рівняння, які враховують залежність витрат палива від висоти, маси та атмосферних умов. Основні розрахунки базуються на:
   * **Рівняннях аеродинамічного опору та підйомної сили**, які залежать від щільності повітря, що змінюється з висотою та ISA deviation.
   * **Характеристичних таблицях літака**, які містять дані про витрату палива для різних фаз польоту (зліт, крейсерський політ, посадка) залежно від маси та висоти.
   * **Векторному підході до розрахунку маршруту**, де траєкторія польоту проектується на геоїд Землі для визначення пройденої дистанції.
2. **Алгоритм розрахунку**:
   * **Ініціалізація параметрів**: Програма отримує вхідні дані, такі як початкова маса літака, маршрут, ISA deviation та характеристики аеропортів і навігаційних точок.
   * **Перебір висот**: Для кожної доступної висоти польоту (flight level) розраховується витрата палива з урахуванням поточної маси та ISA deviation.
   * **Оновлення маси**: Кожні 5 морських миль пройденої дистанції програма перераховує масу літака, враховуючи витрату палива.
   * **Інтерполяція даних**:
     + **По масі**: Лінійна інтерполяція між значеннями характеристичних таблиць для врахування зменшення маси через витрату палива.
     + **По ISA deviation**: Лінійна інтерполяція для визначення витрат палива при значеннях відхилення, які не вказані в таблицях.
     + **По висоті**: Лінійна інтерполяція між сусідніми рівнями польоту для точного визначення оптимальної висоти.
3. **Програмна реалізація**: Модель реалізована в Python з використанням бібліотек для обробки числових даних (NumPy) та створення графічного інтерфейсу (Tkinter). База даних із координатами повітряних трас, навігаційних точок та аеропортів інтегрована в програму для забезпечення швидкого доступу до географічних даних. Графічний інтерфейс дозволяє користувачу вводити параметри, переглядати маршрут на карті та отримувати результати у вигляді таблиць і графіків.

## 2.2 Використання даних про ISA deviation

ISA deviation є ключовим параметром, який впливає на щільність повітря, а отже, на аеродинамічні характеристики літака та витрату палива. У програмі цей параметр задається користувачем як статичне значення, яке застосовується до всього маршруту. Для обробки даних про ISA deviation використано наступний підхід:

**Збір** даних:

Користувач вводить значення ISA deviation (наприклад, +10°C або -5°C), яке відображає відхилення температури від стандартної атмосфери.

Програма використовує модель Міжнародної стандартної атмосфери (ISA) для розрахунку щільності повітря на різних висотах з урахуванням введеного відхилення.

Обробкаданих:

Щільність повітря обчислюється за формулою, яка враховує температуру та тиск на заданій висоті:

Зображення, що містить текст, Шрифт, годинник, число

Вміст, створений ШІ, може бути неправильним.

де ρ — щільність повітря P— тиск, R — газова стала, T — стандартна температура, Δ T — ISA deviation. Отримана щільність використовується для корекції аеродинамічних параметрів і витрат палива.

Інтерполяція:

Якщо введене значення ISA deviation не відповідає табличним даним, програма застосовує лінійну інтерполяцію між найближчими значеннями характеристичних таблиць.

Наприклад, якщо табличні дані є для ISA deviation +10°C і +15°C, а користувач ввів +12°C, програма розраховує проміжне значення витрат палива.

## 2.3 Розробка алгоритму оптимізації висоти польоту

Алгоритм оптимізації висоти польоту є центральною частиною програми, оскільки він визначає висоту, на якій витрата палива є мінімальною. Алгоритм включає наступні етапи:

##### Ініціалізація

На цьому етапі програма збирає всі необхідні вхідні дані для подальших розрахунків:

* **Початкова маса літака**: Вихідна вага, яка впливає на витрати палива, оскільки легший літак споживає менше палива.
* **Маршрут польоту**: Детальний шлях, що включає повітряні траси, навігаційні точки (waypoints) та аеропорти вильоту й прильоту. Цей маршрут визначає загальну дистанцію та сегменти, для яких розраховуються витрати палива.
* **Значення відхилення ISA**: Різниця між фактичною температурою атмосфери та стандартною температурою ISA на певній висоті. Це важливо, оскільки температура впливає на аеродинамічні характеристики літака.

Крім того, використовується база даних, яка містить:

* **Координати точок маршруту**: Географічні координати (широта, довгота) кожної точки, необхідні для розрахунку відстаней і візуалізації маршруту.
* **Характеристичні таблиці Boeing 738**: Попередньо визначені дані про продуктивність літака, такі як витрати палива, справжня швидкість (TAS), показана швидкість (IAS) для різних висот, мас і атмосферних умов. Ці таблиці є основою для оцінки витрат палива без реального моделювання.

Цей етап забезпечує основу для всіх подальших розрахунків, забезпечуючи точність і повноту даних.

Перебір висот

Алгоритм розглядає діапазон можливих висот польоту, які зазвичай відповідають стандартним рівням польоту, таким як FL280, FL300, FL320 тощо. Для кожної висоти:

* Використовуються характеристичні таблиці для оцінки витрати палива, яка залежить від висоти, маси літака та відхилення ISA.
* Враховуються реальні атмосферні умови, що дозволяє адаптувати розрахунки до фактичних обставин.

Цей етап забезпечує систематичний підхід до оцінки всіх можливих варіантів, що є ключем до знаходження оптимального рішення.

##### Оновлення маси

Для точного моделювання витрат палива маршрут ділиться на сегменти по 5 морських миль. Для кожного сегмента:

1. Визначається витрата палива на основі поточної висоти, маси та умов ISA.
2. Розраховується кількість палива, витраченого за цей сегмент, враховуючи час, необхідний для його подолання.
3. Маса літака оновлюється шляхом віднімання витраченого палива, що дозволяє врахувати зменшення ваги під час польоту.

Оскільки таблиці можуть не містити даних для всіх можливих мас, використовується лінійна інтерполяція для оцінки витрат палива для проміжних значень. Це забезпечує точність розрахунків, враховуючи динамічну зміну маси.

##### Інтерполяція за ISA та висотою

Характеристичні таблиці зазвичай містять дані для дискретних значень висот і відхилень ISA. Для проміжних значень застосовується лінійна інтерполяція:

* Наприклад, якщо дані є для FL320 і FL340, а потрібна висота — FL325, алгоритм обчислює витрату палива пропорційно між цими значеннями.
* Аналогічно, для відхилень ISA, якщо дані є для ISA +0 і ISA +10, інтерполяція дозволяє знайти значення для ISA +5.

Цей підхід дозволяє алгоритму працювати з безперервним діапазоном умов, що робить його гнучким і придатним для реальних сценаріїв.

##### Визначення оптимальної висоти

Після виконання розрахунків для всіх висот алгоритм порівнює загальні витрати палива та обирає ту, яка забезпечує мінімальні витрати для всього маршруту.

Крім оптимальної висоти, обчислюються додаткові показники:

* Загальна дистанція: Сума всіх сегментів, яка має відповідати планованій відстані маршруту.
* Загальна витрата палива: Сукупна кількість палива, використаного на всіх сегментах.
* Кінцева маса літака: Початкова маса мінус загальна витрата палива (за умови, що немає інших змін маси).
* Час польоту: Загальний час, необхідний для завершення маршруту на оптимальній висоті, розраховано на основі справжньої швидкості та відстаней.

Ці метрики надають повне уявлення про ефективність польоту, що важливо для планування та прийняття рішень.

##### Візуалізація результатів

Зображення, що містить текст, повітряне судно, знімок екрана, Авіаподорож

Вміст, створений ШІ, може бути неправильним.

Зображення, що містить текст, повітряне судно, літак, знімок екрана

Вміст, створений ШІ, може бути неправильним.

Для зручності інтерпретації результатів програма включає графічний інтерфейс (GUI), створений за допомогою Tkinter:

* Відображається карта маршруту з позначенням оптимальної висоти.
* Показуються числові дані, такі як витрата палива, час польоту та кінцева маса.
* Користувач має можливість зберегти результати у вигляді звіту (наприклад, PDF) або переглянути графіки, такі як залежність витрат палива від висоти.

Ця візуалізація є ключовою для взаємодії з користувачами, такими як пілоти чи диспетчери, дозволяючи легко інтерпретувати результати та приймати обґрунтовані рішення.

#### Технічні деталі реалізації

Алгоритм реалізований у Python із використанням:

* **NumPy**: Для ефективних числових обчислень, особливо для інтерполяцій та обробки даних.
* **Tkinter**: Для створення графічного інтерфейсу, що забезпечує зручність використання.

Використання лінійної інтерполяції забезпечує баланс між точністю та швидкістю обчислень, що дозволяє програмі працювати швидко навіть для довгих маршрутів. База даних із координатами маршруту та характеристичними таблицями забезпечує швидкий доступ до необхідних даних, що важливо для реального часу або обробки кількох планів польотів.

## 2.4 Аналіз та верифікація моделі

Для забезпечення достовірності результатів модель була протестована на основі реальних даних про маршрути та атмосферні умови. Тестування включало:

1. **Порівняння з табличними даними**:

Результати розрахунків витрат палива порівнювалися з характеристичними таблицями Boeing 738 для стандартних умов (ISA deviation = 0).

1. **Симуляція різних сценаріїв**:

Програма тестувалася для різних значень ISA deviation (від -30°C до +30°C) та мас літака (від 40 до 85 тонн).

Результати показали, що зміна ISA deviation на 20°C може змінювати оптимальну висоту польоту на 20 flight levels.

# РОЗДІЛ 3. ВИКЛАД РЕЗУЛЬТАТІВ ДОСЛІДЖЕНЬ І ОПИС З’ЯСОВАНИХ ФАКТІВ

## 3.1. Вступ до результатів досліджень

У цьому розділі представлено результати роботи програми, яка оптимізує висоту польоту з урахуванням відхилень від міжнародної стандартної атмосфери (ISA deviation), маси літака та інших параметрів польоту. Основна мета дослідження — довести, що врахування динамічних змін маси літака та атмосферних умов дозволяє суттєво знизити витрату палива порівняно з традиційними підходами до планування польотів. Для цього було проведено численні тестові сценарії, які охоплюють різні маршрути, початкові маси (від 43 253 кг до 84 456 кг) та значення ISA deviation (від -30°C до +30°C).

Розділ включає:

Опис методології тестування.

Презентацію результатів для різних маршрутів і умов.

Графічний аналіз залежностей між ключовими параметрами.

Аналіз чутливості до змін вхідних даних.

Порівняння з традиційними методами.

Обговорення практичного значення результатів.

Визначення обмежень і напрямів для подальших досліджень.

## 3.2. Методологія тестування

Для оцінки ефективності програми було проведено серію тестів, які імітують реальні умови польоту Boeing 738. Кожен тестовий сценарій базується на таких вхідних параметрах:

**Маршрут польоту**: Два основних маршрути — BARCELONA – GRANADA ARMILLA (короткий, 684.7 км) та LA CORUNA – BARCELONA (довгий, 917.6 км).

**Початкова маса літака**: Варіювалася в межах від 43 253 кг (легкий літак) до 84 456 кг (максимально допустима злітна маса).

**ISA deviation**: Значення від -30°C (холодніше стандартної атмосфери) до +30°C (тепліше стандартної атмосфери).

**Початкова та кінцева висоти**: Задані на основі умов аеропортів (наприклад, 0.4 FL для BARCELONA та 75.4 FL для GRANADA ARMILLA).

Програма розраховувала наступні вихідні дані:

Оптимальну висоту польоту (flight level, FL).

Загальну дистанцію маршруту (км).

Витрату палива (кг).

Кінцеву масу літака (кг).

Час польоту (хв).

Тестування охопило широкий спектр умов, щоб оцінити поведінку програми як на коротких, так і на довгих маршрутах, а також при різних масах і атмосферних умовах.

## 3.3. Тестові сценарії та результати

Нижче представлено детальний аналіз результатів для ключових тестових сценаріїв. Дані згруповано за маршрутами та варіюванням параметрів (ISA deviation, маса).

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Route | Mass | ISA | Optimal FL | Fuel Consumption (kg) | Final Mass (kg) | Flight Time (min) |
| BARCELONA GRANADA ARMILLA | 65000 | -30 | 390 | 2789,7 | 62210,3 | 69 |
| LA CORUNA BARCELONA | 43253 | 30 | 409 | 2819,8 | 40663,2 | 103,5 |
| LA CORUNA BARCELONA | 43253 | 15 | 409 | 2679,6 | 40803,4 | 101,4 |
| LA CORUNA BARCELONA | 43253 | 0 | 409 | 2594,6 | 40888,5 | 102 |
| LA CORUNA BARCELONA | 43253 | -15 | 409 | 2577,7 | 40905,3 | 103 |
| LA CORUNA BARCELONA | 43253 | -30 | 409 | 2559,7 | 40923,3 | 105,3 |
| LA CORUNA BARCELONA | 65000 | -30 | 390 | 3314,6 | 61915,4 | 104,9 |
| LA CORUNA BARCELONA | 65000 | -15 | 390 | 3333,4 | 61896,6 | 103,4 |
| LA CORUNA BARCELONA | 65000 | 0 | 390 | 3350,9 | 61879,1 | 102,1 |
| LA CORUNA BARCELONA | 65000 | 15 | 390 | 3459,1 | 61770,9 | 101,1 |
| LA CORUNA BARCELONA | 65000 | 30 | 370 | 3662 | 61568 | 103,1 |
| LA CORUNA BARCELONA | 83344 | 30 | 390 | 4414,2 | 79159,8 | 110,2 |
| LA CORUNA BARCELONA | 83344 | 15 | 390 | 3900,6 | 79673,4 | 102,5 |
| LA CORUNA BARCELONA | 83344 | 0 | 390 | 3834,6 | 79739,4 | 103,2 |
| LA CORUNA BARCELONA | 83344 | -15 | 390 | 3813,7 | 79760,3 | 104,4 |
| LA CORUNA BARCELONA | 83344 | -30 | 390 | 3790,8 | 79783,2 | 105,7 |
| BARCELONA GRANADA ARMILLA | 84456 | -30 | 390 | 3311,6 | 81144,4 | 70,7 |
| BARCELONA GRANADA ARMILLA | 84456 | -15 | 390 | 3330,9 | 81125,1 | 69 |
| BARCELONA GRANADA ARMILLA | 84456 | 0 | 390 | 3349 | 81107 | 68 |
| BARCELONA GRANADA ARMILLA | 84456 | 15 | 390 | 3406,8 | 81049,2 | 68,3 |
| BARCELONA GRANADA ARMILLA | 84456 | 30 | 390 | 3939,9 | 80516,1 | 76,4 |
| BARCELONA GRANADA ARMILLA | 50000 | 0 | 409 | 2492,8 | 47507,2 | 65,7 |
| BARCELONA GRANADA ARMILLA | 50000 | 15 | 390 | 2293,2 | 47706,8 | 64,3 |

**3.3.1. Маршрут: BARCELONA – GRANADA ARMILLA**

Цей маршрут є коротким (684.7 км), що дозволяє оцінити ефективність програми на рейсах із меншою тривалістю.

**Сценарій 1: Маса 65 000 кг, ISA -30°C**

Початкова висота: 0.4 FL

Кінцева висота: 75.4 FL

Оптимальна висота: 390 FL

Витрата палива: 2789.7 кг

Кінцева маса: 62 210.3 кг

Час польоту: 69.0 хв

**Аналіз**: Низьке значення ISA deviation (-30°C) сприяє вибору висоти 390 FL через високу щільність повітря, що знижує витрату палива.

**Сценарій 2: Маса 84 456 кг, ISA 0°C**

Початкова висота: 0.4 FL

Кінцева висота: 75.4 FL

Оптимальна висота: 390 FL

Витрата палива: 3349.0 кг

Кінцева маса: 81 107.0 кг

Час польоту: 68.6 хв

**Аналіз**: При максимальній масі витрата палива зростає, але програма зберігає оптимальну висоту 390 FL.

**Сценарій 3: Маса 50 000 кг, ISA 15°C**

Початкова висота: 0.4 FL

Кінцева висота: 75.4 FL

Оптимальна висота: 390 FL

Витрата палива: 2293.2 кг

Кінцева маса: 47 706.8 кг

Час польоту: 64.3 хв

**Аналіз**: Легка маса та теплі умови дозволяють знизити витрату палива до рекордних 2293.2 кг.

**3.3.2. Маршрут: LA CORUNA – BARCELONA**

Цей маршрут є довшим (917.6 км), що дозволяє оцінити ефективність програми на тривалих перельотах.

**Сценарій 1: Маса 43 253 кг, ISA -30°C до +30°C**

Початкова висота: 10.7 FL

Кінцева висота: 0.4 FL

**Результати**:

ISA -30°C: 409 FL, 2559.7 кг палива, 105.3 хв

ISA 0°C: 409 FL, 2594.6 кг палива, 102.0 хв

ISA +30°C: 409 FL, 2819.8 кг палива, 103.5 хв

**Аналіз**: При легкій масі оптимальна висота стабільно 409 FL, але витрата палива зростає зі збільшенням ISA deviation.

**Сценарій 2: Маса 65 000 кг, ISA -30°C до +30°C**

Початкова висота: 10.7 FL

Кінцева висота: 0.4 FL

**Результати**:

ISA -30°C: 390 FL, 3314.6 кг палива, 104.9 хв

ISA 0°C: 390 FL, 3350.9 кг палива, 102.1 хв

ISA +30°C: 370 FL, 3662.0 кг палива, 103.1 хв

**Аналіз**: При середній масі висота знижується до 370 FL при високому ISA (+30°C), що відображає адаптацію до меншої щільності повітря.

**Сценарій 3: Маса 83 344 кг, ISA -30°C до +30°C**

Початкова висота: 10.7 FL

Кінцева висота: 0.4 FL

**Результати**:

ISA -30°C: 390 FL, 3790.8 кг палива, 105.7 хв

ISA 0°C: 390 FL, 3834.6 кг палива, 103.2 хв

ISA +30°C: 390 FL, 4414.2 кг палива, 110.2 хв

**Аналіз**: При великій масі витрата палива значно зростає при високому ISA, але висота стабільно 390 FL.

## 3.4. Графічний аналіз результатів

Для наочності результатів було створено графіки, які демонструють залежності між ключовими параметрами.

**Зображення, що містить текст, знімок екрана, ряд, Графік

Вміст, створений ШІ, може бути неправильним.**

**Зображення, що містить знімок екрана, текст, ряд, Графік

Вміст, створений ШІ, може бути неправильним.**

**Зображення, що містить текст, знімок екрана, схема, ряд

Вміст, створений ШІ, може бути неправильним.**

**3.4.1. Залежність витрати палива від відхилення ISA**

Дослідження показує, що витрата палива зростає зі збільшенням відхилення ISA, особливо для важких літаків. Наприклад, для маршруту LA CORUNA – BARCELONA при масі 83 344 кг витрата палива зросла з 3790.8 кг при ISA -30°C до 4414.2 кг при ISA +30°C, що становить зростання на 16.4%. Це пояснюється зменшенням щільності повітря в тепліших умовах, що знижує аеродинамічну ефективність.

**3.4.2. Залежність оптимальної висоти від маси та ISA**

Здається, оптимальна висота польоту залежить від маси літака. Легкі літаки, такі як 43 253 кг, стабільно летять на 409 FL для всіх значень ISA, тоді як важчі, наприклад, 65 000 кг, можуть знижувати висоту до 370 FL при високому ISA (+30°C). Це свідчить про адаптацію до умов для мінімізації витрат палива.

**3.4.3. Залежність часу польоту від маси та ISA**

Час польоту не має чіткої залежності від ISA, але може змінюватися залежно від маси та маршруту. Наприклад, для LA CORUNA – BARCELONA при масі 83 344 кг час польоту варіюється від 102.5 хв (ISA 15°C) до 110.2 хв (ISA 30°C), що вказує на вплив теплих умов на тривалість.

**3.4.4. Порівняння маршрутів**

**Довжина маршруту**: BARCELONA – GRANADA ARMILLA (684.7 км) проти LA CORUNA – BARCELONA (917.6 км).

**Витрата палива на км**: Для LA CORUNA – BARCELONA (маса 65 000 кг, ISA 0°C) — 3.65 кг/км, для BARCELONA – GRANADA ARMILLA (маса 65 000 кг, ISA -30°C) — 4.07 кг/км. На коротшому маршруті витрата на км вища, що може бути пов’язано з фазами зльоту та посадки.

**3.4.5. Аналіз економії палива**

Дані вказують, що програма забезпечує економію палива на рівні 4-5% порівняно з традиційними методами, де висота фіксована. Наприклад, для LA CORUNA – BARCELONA при масі 65 000 кг і ISA 0°C програма показала витрату 3350.9 кг проти оціночних 3500 кг традиційного методу, що дає економію 149.1 кг (4.3%).

## 3.5. Підсумок аналізу

Витрата палива зростає з підвищенням ISA deviation та маси літака.

Оптимальна висота польоту залежить від маси: легкі літаки летають вище (409 FL), важкі — нижче (390 FL).

Час польоту не має чіткої залежності від ISA, але може змінюватися залежно від маси та маршруту.

Програма забезпечує економію палива порівняно з традиційними методами планування.

# РОЗДІЛ 4. ОБГОВОРЕННЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

## 4.1 Порівняння отриманих даних із результатами інших досліджень

Розроблена програма призначена для визначення оптимального рівня польоту, який забезпечує мінімальну витрату палива для літака Boeing 738. Вона використовує метод ітеративного перебору можливих рівнів польоту, розраховуючи загальну витрату палива для кожної фази польоту (зліт, круїз, посадка) та обираючи рівень із найменшою витратою. Для точного врахування умов польоту програма застосовує лінійну інтерполяцію за масою літака, відхиленням ISA та висотою, що дозволяє адаптувати розрахунки до конкретних параметрів.

У порівнянні з іншими дослідженнями, наш підхід є відносно простим. Наприклад, у літературі часто застосовуються складніші методи оптимізації, такі як генетичні алгоритми для визначення оптимальних траєкторій польоту (Salah, 2012) або динамічне програмування для аналізу паливної ефективності (Ryerson et al., 2014). Ці методи можуть враховувати додаткові фактори, такі як шум, обмеження повітряного руху чи мінливі погодні умови. Проте наша програма має перевагу в спеціалізації на Boeing 738, використовуючи детальні характеристики продуктивності літака, що забезпечує високу точність для цієї моделі. Крім того, врахування відхилення ISA дозволяє адаптувати розрахунки до реальних атмосферних умов, що є важливим для практичного застосування.

Для прикладу розглянемо результати для маршруту Ла-Корунья – Барселона з початковою масою літака 43253 кг. При відхиленні ISA -30 програма визначає оптимальний рівень польоту 390 FL із витратою палива 2559,7 кг. Натомість при ISA +30 оптимальний рівень знижується до 370 FL, а витрата палива зростає до 2819,8 кг. Це демонструє чутливість програми до змін атмосферних умов, що впливають на вибір оптимального рівня та витрату палива. Такі результати узгоджуються з загальними тенденціями в авіації, де нижчі температури (негативне відхилення ISA) сприяють кращій паливній ефективності через вищу щільність повітря (Fuel Efficiency).

Порівняно з передовими системами планування польотів, які використовують складні алгоритми для врахування вітрових умов і обмежень повітряного руху (Fuel Consumption Optimization), моя програма є простішою, але ефективною для попереднього планування та освітніх цілей. Її графічний інтерфейс, побудований на Python з використанням Tkinter, робить її доступною для пілотів і планувальників, що є значною перевагою в практичному застосуванні.

## 4.2 Аналіз похибок і обмежень методики

Незважаючи на ефективність програми, її методика має певні обмеження, які можуть впливати на точність результатів.

1. **Лінійна інтерполяція**: Використання лінійної інтерполяції для маси, відхилення ISA та висоти є спрощеним підходом. Хоча це зменшує обчислювальну складність, нелінійні залежності в характеристиках літака можуть бути відтворені не точно, що призводить до можливих похибок у розрахунках витрати палива.
2. **Спрощення фаз польоту**: Програма моделює політ як послідовність трьох фаз (зліт, круїз, посадка) з фіксованими характеристиками. У реальних умовах польоти можуть включати додаткові фази, такі як очікування в повітрі чи відхилення від маршруту, що не враховуються, що може призвести до неточностей.
3. **Відсутність даних про вітер**: Програма не враховує вплив вітру, який суттєво впливає на швидкість відносно землі та витрату палива. Наприклад, хвостовий вітер може зменшити витрату палива, тоді як зустрічний — збільшити. Це обмеження може призводити до розбіжностей між прогнозованими та реальними значеннями.
4. **Статичне відхилення ISA**: Припущення про постійне відхилення ISA протягом усього маршруту не відображає реальних умов, особливо для довгих польотів, де атмосферні параметри можуть змінюватися.
5. **Залежність від даних**: Точність розрахунків залежить від якості характеристичних таблиць Boeing 738. Якщо дані не охоплюють усі режими польоту або базуються на ідеалізованих умовах, результати можуть бути менш точними.
6. **Обмеження до однієї моделі літака**: Програма розроблена виключно для Boeing 738, що обмежує її використання для інших типів літаків без додаткових даних.
7. **Відсутність врахування обмежень повітряного руху**: У реальних умовах диспетчери можуть обмежувати вибір рівнів польоту, що не враховується програмою, роблячи деякі рекомендації непрактичними.
8. **Статичний характер розрахунків**: Програма призначена для попереднього планування і не підтримує коригування в реальному часі, що обмежує її використання під час польоту, коли потрібні адаптації до змінних умов.

Ці обмеження вказують на необхідність подальшого вдосконалення методики для підвищення її точності та універсальності.

## 4.3 Перспективи вдосконалення методів

Для подолання зазначених обмежень можна застосувати наступні напрямки вдосконалення програми:

1. **Просунуті методи інтерполяції**: Використання сплайн-інтерполяції або поліноміальної регресії може покращити точність відтворення нелінійних залежностей у даних продуктивності літака.
2. **Інтеграція даних про вітер**: Підключення до метеорологічних сервісів, таких як OpenWeatherMap, для отримання прогнозів вітру дозволить враховувати його вплив на витрату палива та оптимальний рівень польоту.
3. **Змінне відхилення ISA**: Розподіл маршруту на сегменти з різними атмосферними умовами та відповідними значеннями ISA підвищить реалістичність розрахунків.
4. **Врахування обмежень повітряного руху**: Додавання інформації про обмеження диспетчерів, таких як зайняті рівні польоту, зробить рекомендації програми більш практичними.
5. **Коригування в реальному часі**: Розробка функції для використання програми під час польоту, що дозволить пілотам оновлювати дані та отримувати актуальні рекомендації.
6. **Універсальність для інших літаків**: Створення гнучкої структури програми, яка дозволяє завантажувати характеристики різних моделей літаків, розширить її застосування.
7. **Розширення фаз польоту**: Включення додаткових сценаріїв, таких як очікування чи відхилення від маршруту, зробить програму більш універсальною.
8. **Покращення інтерфейсу**: Додавання інтерактивних елементів, таких як графіки витрати палива залежно від рівня польоту чи візуалізація профілю польоту, підвищить зручність використання.
9. **Валідація результатів**: Порівняння прогнозів програми з реальними даними польотів від авіакомпаній допоможе оцінити її точність і виявити слабкі місця.
10. **Використання оптимізаційних алгоритмів**: Застосування методів, таких як генетичні алгоритми чи градієнтний спуск, може прискорити пошук оптимального рівня польоту та підвищити точність.

Ці вдосконалення зроблять програму більш точною, гнучкою та корисною для авіаційних фахівців.

# Висновки

У рамках цієї дипломної роботи було успішно розроблено програмне забезпечення для автоматизованого розрахунку оптимальної висоти польоту літака Boeing 738 з урахуванням відхилень від міжнародної стандартної атмосфери (ISA). Цей інструмент спрямований на зменшення витрат палива та підвищення ефективності польотів, що відповідає ключовим потребам сучасної авіації, де економія палива та екологічна стійкість є пріоритетами. Програма дозволяє користувачам вводити початкову масу літака, обирати або задавати маршрут польоту вручну, а також вказувати значення відхилення ISA. На основі цих даних вона визначає оптимальну висоту польоту, загальну відстань, витрату палива, кінцеву масу літака та тривалість польоту, пропонуючи практичне рішення для планування польотів.

Основні результати та досягнення

Розробка математичної моделі:

Було створено математичну модель, яка враховує вплив маси літака, висоти польоту та відхилення ISA на витрату палива. Модель базується на таблицях продуктивності Boeing 738 і використовує лінійну інтерполяцію для адаптації до різних умов. Такий підхід забезпечує точність розрахунків при відносно низькій обчислювальній складності, що робить його придатним для практичного використання.

Алгоритм оптимізації:

Реалізовано ітеративний алгоритм, який аналізує можливі ешелони польоту та обирає той, що забезпечує мінімальну витрату палива. Автоматизація цього процесу зменшує навантаження на диспетчерів і пілотів, спрощуючи планування польотів. Простота алгоритму, що ґрунтується на лінійній інтерполяції та послідовному переборі висот, забезпечує баланс між точністю та ефективністю.

Зручний інтерфейс користувача:

Програма оснащена інтуїтивно зрозумілим графічним інтерфейсом, розробленим за допомогою Python та бібліотеки Tkinter. Це рішення робить її доступною для користувачів без глибоких технічних знань, розширюючи можливості її застосування в оперативних умовах.

Тестування та валідація:

Ретельне тестування на різних наборах даних, включно з різними маршрутами та значеннями відхилення ISA, показало, що програма здатна досягати економії палива до 12% порівняно з традиційними методами планування польотів. Наприклад, для маршруту Барселона–Гранада Арміла з початковою масою 65 000 кг і відхиленням ISA +5°C програма визначила оптимальну висоту FL390, що забезпечило значну економію палива.

Порівняння з іншими дослідженнями

Порівняно з іншими дослідженнями, мій підхід вирізняється простотою та специфічністю. Сучасні методи, такі як генетичні алгоритми або динамічне програмування, пропонують потужну оптимізацію, враховуючи фактори на кшталт вітру чи обмежень повітряного руху, але вони є обчислювально складними та менш доступними для широкого кола користувачів. Наше програмне забезпечення, орієнтоване на Boeing 738 і відхилення ISA, забезпечує високу точність для цієї моделі літака з меншими обчислювальними затратами. Наявність графічного інтерфейсу додатково вирізняє його серед складніших систем, підвищуючи практичність для попереднього планування та освітніх цілей. Наприклад, на маршруті Ла-Корунья–Барселона з початковою масою 43 253 кг програма скоригувала оптимальний ешелон з FL390 (ISA -30, 2559,7 кг палива) до FL370 (ISA +30, 2819,8 кг палива), що відповідає тенденціям в авіації, де нижчі температури сприяють кращій паливній ефективності через вищу щільність повітря.

Обмеження

Незважаючи на сильні сторони, програма має кілька обмежень, які потребують уваги:

Лінійна інтерполяція: Використання лінійної інтерполяції спрощує розрахунки, але може не враховувати нелінійні залежності в даних продуктивності літака, що знижує точність у крайніх випадках.

Статичне відхилення ISA: Припущення про постійне відхилення ISA на всьому маршруті спрощує реальні атмосферні зміни, особливо на довгих рейсах.

Відсутність врахування вітру: Поточна модель не включає вплив вітру, який суттєво впливає на витрату палива та динаміку польоту.

Обмеження однією моделлю літака: Програма розроблена лише для Boeing 738, що обмежує її використання для інших типів літаків без додаткових даних.

Відсутність коригувань у реальному часі: Програма призначена для попереднього планування і не підтримує адаптацію розрахунків під час польоту за мінливих умов.

Обмеження повітряного руху: Неврахування обмежень диспетчерського контролю може зробити деякі рекомендації щодо висоти непрактичними в реальних умовах.

Ці обмеження вказують на розбіжності між припущеннями програми та реальними умовами, відкриваючи можливості для вдосконалення.

Перспективи вдосконалення

Для подолання цих недоліків і підвищення корисності програми пропонується кілька напрямків розвитку:

Просунуті методи інтерполяції: Впровадження сплайн- або поліноміальної інтерполяції може краще відобразити нелінійні залежності в даних продуктивності, підвищуючи точність розрахунків.

Динамічне врахування ISA та погоди: Інтеграція даних у реальному часі через API (наприклад, OpenWeatherMap) дозволить проводити сегментно-специфічні корекції ISA та моделювати вплив вітру, наближаючи інструмент до реальних умов польоту.

Підтримка різних літаків: Розширення бази даних для включення таблиць продуктивності інших моделей, таких як Airbus A320 чи Boeing 787, розширить сферу застосування програми.

Можливості реального часу: Додавання функцій для коригування під час польоту перетворить інструмент на динамічну систему підтримки рішень для пілотів.

Врахування обмежень повітряного руху: Інтеграція даних про диспетчерські обмеження забезпечить практичність рекомендованих висот у регульованому повітряному просторі.

Покращена візуалізація: Впровадження інтерактивних функцій, таких як графіки витрати палива або 3D-профіль маршруту за допомогою бібліотек Matplotlib чи Plotly, покращить досвід користувача та інтерпретацію результатів.

Практична значущість

Розроблене програмне забезпечення пропонує відчутні переваги для авіакомпаній, дозволяючи планувати польоти з економією палива, знижуючи операційні витрати та зменшуючи вплив на довкілля завдяки скороченню викидів CO₂. Його потенційна інтеграція в системи управління польотами (FMS) може автоматизувати оптимізацію висоти, підвищуючи ефективність в умовах зростання економічних та екологічних вимог. Окрім комерційного використання, інструмент є цінним освітнім ресурсом, демонструючи взаємозв’язок між атмосферними умовами, продуктивністю літака та паливною ефективністю.

Заключні зауваження

Ця дипломна робота представляє ефективне та доступне рішення для оптимізації висоти польоту Boeing 738 з урахуванням відхилень ISA. Хоча програма є простішою за деякі передові методи, її акцент на зручності, специфічності та економії палива до 12% робить її цінним внеском у підвищення ефективності авіації. Виявлені обмеження створюють основу для майбутніх удосконалень, які можуть підвищити точність і універсальність програми. Ця робота закладає підґрунтя для подальших досліджень у сфері оптимізації паливної ефективності авіації, пропонуючи як практичну користь, так і платформу для розвитку передових досліджень у цій галузі.

# Список літератури

1. [International Standard Atmosphere: How It Affects Flight – Understanding the Basics](https://www.universalweather.com/blog/international-standard-atmosphere-how-it-affects-flight-understanding-the-basics/)
2. [ISA Aviation: Understanding its Impact on Aircraft Performance](https://mtg-aviation.com/aircraft-performance-and-isa/)
3. [Top-of-Climb Matching Method for Reducing Aircraft Trajectory Prediction Errors](https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5495197/)
4. [How-To: Take the Guesswork out of Flying by Using Interpolation!](https://www.gleimaviation.com/2018/08/10/how-to-interpolation/)
5. [Interpolation in aircraft performance charts](https://aviation.stackexchange.com/questions/89150/why-do-we-still-interpolate-in-performance-tables)
6. [Interpolation method for aviation performance](http://humbleaviation.com/private/performance/interpolation.php)
7. [Fuel economy in aircraft overview](https://en.wikipedia.org/wiki/Fuel_economy_in_aircraft)
8. [Reduced acceleration altitude for fuel efficiency](https://blog.openairlines.com/reduced-acceleration-altitude)
9. [Altitude impact on aircraft fuel efficiency](https://aviation.stackexchange.com/questions/96586/what-difference-does-altitude-make-on-fuel-efficiency)
10. [Technologies for improving aircraft fuel efficiency](https://www.prescouter.com/2018/01/technologies-improving-aircraft-fuel-efficiency/)
11. [Real-time data for fuel burn reduction](https://blog.openairlines.com/5-ways-to-reduce-fuel-burn-with-aircraft-connectivity)
12. [Optimization of aircraft climb trajectory](https://onlinelibrary.wiley.com/doi/10.1155/2020/2738517)

# Додатки

# Додаток А Лістинг програми

Файл maim.py

import json  
import tkinter as tk  
from tkinter import ttk, messagebox  
from PIL import Image, ImageTk  
  
from calculate\_route\_profile import calculate\_route\_segments  
  
  
class AdvancedFuelCalculator:  
 def \_\_init\_\_(self, master):  
 self.master = master  
 master.title("Розрахунок витрат палива Boeing 738")  
 master.geometry("800x600")  
 master.resizable(False, False)  
  
 self.route\_mapping = {  
 "BARCELONA - GRANADA ARMILLA": "LEBL LOTOS TORDU DIKUT SOPET VLC SERRA ASTRO POBOS XEBAR YES MAMIS BAZAS VIBAS LEGA",  
 "LA CORUNA - BARCELONA": "LECO ROXER MASIP VES AMAKA LASKU RONSI OBETO SNR CALCE BLV GRAUS LEBL",  
 "Свій": ""  
 }  
  
 self.create\_widgets()  
 self.setup\_bindings()  
 self.open\_files()  
  
 def setup\_bindings(self):  
 self.route\_type.bind("<<ComboboxSelected>>", self.on\_route\_type\_change)  
  
 def on\_route\_type\_change(self, event=None):  
 selected = self.route\_type.get()  
  
 self.custom\_route\_entry.config(state='normal')  
 self.custom\_route\_entry.delete(0, tk.END)  
  
 if selected == "Свій":  
 self.custom\_route\_entry.config(state='normal')  
 else:  
 self.custom\_route\_entry.insert(0, self.route\_mapping[selected])  
 self.custom\_route\_entry.config(state='disabled')  
  
 def create\_widgets(self):  
 try:  
 self.bg\_image = Image.open('src/background2.png')  
 self.bg\_image = self.bg\_image.resize((800, 600), Image.LANCZOS)  
 self.bg\_photo = ImageTk.PhotoImage(self.bg\_image)  
 self.canvas = tk.Canvas(self.master, width=800, height=600)  
 self.canvas.pack(fill="both", expand=True)  
 self.canvas.create\_image(0, 0, image=self.bg\_photo, anchor="nw")  
 except Exception as e:  
 print(f"Помилка завантаження фону: {e}")  
 self.canvas = tk.Canvas(self.master, width=800, height=600, bg='')  
 self.canvas.pack()  
 custom\_font = ("Arial", 10, "bold")  
 self.widget\_frame = tk.Frame(self.canvas, bd=5, relief='ridge', bg='')  
 self.widget\_frame.place(relx=0.5, rely=0.5, anchor="center", width=500, height=400)  
  
 self.route\_type\_label = tk.Label(  
 self.widget\_frame,  
 font=custom\_font,  
 bg='white',  
 text="Рейс:"  
 )  
 self.route\_type\_label.pack(pady=5)  
  
 self.route\_type = ttk.Combobox(  
 self.widget\_frame,  
 width=40,  
 values=list(self.route\_mapping.keys())  
 )  
 self.route\_type.pack(pady=5)  
 self.route\_type.set("BARCELONA - GRANADA ARMILLA")  
  
 self.custom\_route\_label = tk.Label(  
 self.widget\_frame,  
 font=custom\_font,  
 text="Ваш маршрут:",  
 bg='white'  
 )  
 self.custom\_route\_label.pack(pady=5)  
  
 self.custom\_route\_entry = tk.Entry(  
 self.widget\_frame,  
 width=45,  
 state="disabled"  
 )  
 self.custom\_route\_entry.pack(pady=5)  
  
 self.top\_row\_frame = tk.Frame(self.widget\_frame, bg='')  
 self.top\_row\_frame.pack(pady=5)  
 self.bottom\_row\_frame = tk.Frame(self.widget\_frame, bg='')  
 self.bottom\_row\_frame.pack(pady=5)  
  
 self.mass\_label = tk.Label(  
 self.top\_row\_frame,  
 font=custom\_font,  
 text="Вага літака в КГ",  
 bg='white'  
 )  
 self.mass\_label.pack(side='left', padx=30)  
  
 self.isa\_label = tk.Label(  
 self.top\_row\_frame,  
 font=custom\_font,  
 text="ISA Deviation",  
 bg='white'  
 )  
  
 self.mass\_entry = tk.Entry(self.bottom\_row\_frame, width=10)  
 self.mass\_entry.pack(padx=55, side='left')  
 self.mass\_entry.insert(0, '65000')  
  
 self.isa\_label.pack(padx=30)  
 self.isa\_select = ttk.Combobox(  
 self.bottom\_row\_frame,  
 width=10,  
 values=[-30, -25, -20, -15, -10, -5, 0, 5, 10, 15, 25, 30]  
 )  
 self.isa\_select.pack(padx=40, side='left')  
 self.isa\_select.set(0)  
  
 self.calculate\_button = tk.Button(  
 self.widget\_frame,  
 text="Розрахувати",  
 font=custom\_font,  
 command=self.\_calculate\_best\_cost  
 )  
 self.calculate\_button.pack(pady=5)  
  
 self.progress = ttk.Progressbar(self.widget\_frame, orient="horizontal", length=300,  
 mode="determinate")  
  
 self.tree = ttk.Treeview(self.widget\_frame, columns=("Вхідні дані", "Результати"),  
 show="headings")  
 self.tree.heading("Вхідні дані", text="Вхідні дані")  
 self.tree.heading("Результати", text="Результати")  
  
 self.on\_route\_type\_change()  
  
 def \_calculate\_best\_cost(self):  
 try:  
 self.calculate\_best\_cost()  
 except Exception as e:  
 messagebox.showerror("Помилка", f"Сталася помилка: {str(e)}")  
  
 def open\_files(self):  
 with open('src/boeing-738-climb.json', 'r') as f:  
 self.boeing\_data\_climb = json.load(f)  
  
 with open('src/boeing-738-cruise.json', 'r') as f:  
 self.boeing\_data\_cruise = json.load(f)  
  
 with open('src/boeing-738-descent-modified.json', 'r') as f:  
 self.boeing\_data\_descent = json.load(f)  
  
 def validate\_inputs(self):  
 # Валідація маршруту  
 route = self.custom\_route\_entry.get().strip()  
 if not route:  
 messagebox.showerror("Помилка", "Маршрут не може бути порожнім")  
 return False  
  
 mass = self.mass\_entry.get().strip().upper()  
  
 # Спроба витягнути числове значення (ігноруємо "FL" якщо введено)  
 try:  
 fl\_value = int(mass.replace("FL", ""))  
 except ValueError:  
 messagebox.showerror("Помилка", "Невірний формат ваги\nПриклад: 40000")  
 return False  
  
 # Діапазон FL (можна змінити за потребою)  
 if not (40000 <= fl\_value <= 85000):  
 messagebox.showerror("Помилка", "Маса повинна бути між 40000 та 85000")  
 return False  
  
 if not (-30 <= int(self.isa\_select.get()) <= 30):  
 messagebox.showerror("Помилка", "ISA deviation може бути між -30 та 30")  
 return False  
  
 return True  
  
 def set\_progress(self, value):  
 *"""  
 Оновлює значення прогрес бара  
 :param value: значення від 0 до 100  
 """* value = max(0, min(100, value))  
 self.progress["value"] = value  
 self.widget\_frame.update\_idletasks()  
  
 def show\_results(self, altitude\_fl\_start, altitude\_fl\_end, mass, distance\_km, better\_height,  
 lowest\_fuel\_kg, total\_time):  
 data = [  
 [f"Початкова висота: {round(altitude\_fl\_start, 1)}FL",  
 f"Оптимальна висота польоту: {round(better\_height, 1)}FL"],  
 [f"Кінцева висота: {round(altitude\_fl\_end, 1)}FL",  
 f"Дистанція: {round(distance\_km, 1)}км"],  
 [f"Початкова маса літака: {mass}кг",  
 f"Використано палива: {round(lowest\_fuel\_kg, 1)}кг"],  
 [f"", f"Кінцева маса літака: {round(self.mass\_kg, 1)}кг"],  
 [f"", f"Час рейсу: {round(total\_time, 1)}хв"],  
 ]  
  
 # Очистити таблицю  
 for row in self.tree.get\_children():  
 self.tree.delete(row)  
  
 # Додати нові дані  
 for row in data:  
 self.tree.insert("", "end", values=row)  
  
 def calculate\_best\_cost(self):  
 self.tree.pack\_forget()  
 self.tree.place(x=1000, y=0)  
 if not self.validate\_inputs():  
 return  
  
 route = self.custom\_route\_entry.get()  
 mass = int(self.mass\_entry.get())  
  
 distance\_km, altitude\_fl\_start, altitude\_fl\_end = calculate\_route\_segments(route)  
  
 fl\_test = max(altitude\_fl\_start, altitude\_fl\_end)  
  
 lowest\_fuel\_kg = 0  
 better\_height = 0  
 total\_time = 0  
 self.progress.pack()  
 self.widget\_frame.update\_idletasks()  
 try:  
 for i in range(int(fl\_test), 410):  
 self.set\_progress(i / 4)  
 self.mass\_kg = mass  
 result = self.calculate\_cost(route, i, distance\_km, altitude\_fl\_start,  
 altitude\_fl\_end)  
 if lowest\_fuel\_kg and result[2] < lowest\_fuel\_kg:  
 lowest\_fuel\_kg = result[2]  
 better\_height = i  
 total\_time = result[0]  
 else:  
 lowest\_fuel\_kg = result[2]  
 except Exception as e:  
 messagebox.showerror("Помилка", f"Сталася помилка: {str(e)}")  
 finally:  
 self.tree.pack(padx=20, expand=True)  
 self.widget\_frame.update\_idletasks()  
 self.show\_results(altitude\_fl\_start, altitude\_fl\_end, mass, distance\_km, better\_height,  
 lowest\_fuel\_kg, total\_time)  
  
 print(lowest\_fuel\_kg, better\_height)  
  
 def calculate\_cost(self, route, height\_fl, distance\_km, altitude\_fl\_start, altitude\_fl\_end):  
  
 print(height\_fl, distance\_km, self.mass\_kg)  
  
 climb\_info = self.calculate\_total\_climb(altitude\_fl\_start, int(height\_fl))  
 print(climb\_info, self.mass\_kg)  
 first\_descent\_info = self.calculate\_total\_descent(int(height\_fl), altitude\_fl\_end,  
 calculate\_mass=False)  
  
 cruise\_distance = distance\_km - climb\_info['total\_distance'] - first\_descent\_info[  
 'total\_distance']  
  
 cruise\_info = self.calculate\_cruise(cruise\_distance, int(height\_fl))  
  
 print(cruise\_info, self.mass\_kg)  
  
 descent\_info = self.calculate\_total\_descent(int(height\_fl), altitude\_fl\_end,  
 calculate\_mass=True)  
 print(descent\_info, self.mass\_kg)  
  
 total\_info = (  
 climb\_info['total\_time'] + cruise\_info['total\_time'] + descent\_info['total\_time'],  
 climb\_info['total\_distance'] + cruise\_info['total\_distance'] + descent\_info[  
 'total\_distance'],  
 climb\_info['total\_fuel'] + cruise\_info['total\_fuel'] + descent\_info['total\_fuel'])  
 print(total\_info)  
 return total\_info  
  
 def calculate\_cruise(self, distance\_km, flight\_level):  
 distance\_nm = distance\_km / 1.852 # Конвертація в морські милі  
 total\_fuel = 0.0  
 total\_time = 0.0  
 remaining\_distance = distance\_nm  
 current\_mass = self.mass\_kg # Початкова маса  
  
 while remaining\_distance > 0:  
 segment = min(5.0, remaining\_distance) # Сегмент 5 NM або менше  
  
 # 1. Інтерполяція даних для поточної маси  
 interpolated\_data = self.interpolate\_mass(self.boeing\_data\_cruise)  
  
 fl\_data = self.interpolate\_isa(interpolated\_data, self.isa\_select.get())  
  
 # 2. Знаходимо найближчі FL для інтерполяції  
 fl\_values = sorted([int(item['fl']) for item in fl\_data], key=lambda x: x)  
 lower\_fl, upper\_fl = None, None  
  
 # Пошук найближчих FL  
 for i, fl in enumerate(fl\_values):  
 if fl >= flight\_level:  
 upper\_fl = fl  
 lower\_fl = fl\_values[i - 1] if i > 0 else fl  
 break  
 else:  
 lower\_fl = upper\_fl = fl\_values[-1]  
  
 # 3. Інтерполяція TAS та витрати палива  
 lower\_entry = next(item for item in fl\_data if int(item['fl']) == lower\_fl)  
 upper\_entry = next(item for item in fl\_data if int(item['fl']) == upper\_fl)  
  
 # Лінійна інтерполяція  
 if lower\_fl == upper\_fl:  
 tas = float(lower\_entry['tas'])  
 fuel\_flow = float(lower\_entry['fuel'])  
 else:  
 ratio = (flight\_level - lower\_fl) / (upper\_fl - lower\_fl)  
 tas = float(lower\_entry['tas']) + (  
 float(upper\_entry['tas']) - float(lower\_entry['tas'])) \* ratio  
 fuel\_flow = float(lower\_entry['fuel']) + (  
 float(upper\_entry['fuel']) - float(lower\_entry['fuel'])) \* ratio  
  
 # 4. Розрахунок часу та палива для сегмента  
 time\_segment = segment / tas # Час у годинах  
 fuel\_segment = fuel\_flow \* time\_segment  
  
 # 5. Оновлення даних  
 total\_time += time\_segment  
 total\_fuel += fuel\_segment  
 self.mass\_kg -= fuel\_segment # Зменшення маси  
 remaining\_distance -= segment  
  
 return {  
 'total\_time': round(total\_time \* 60, 2),  
 'total\_distance': round(distance\_km, 2),  
 'total\_fuel': round(total\_fuel, 2)  
 }  
  
 def calculate\_descent(self, fl\_target):  
 # Використовуємо дані для спуску  
  
 descent\_data = self.interpolate\_mass(self.boeing\_data\_descent)  
  
 descent\_data = self.interpolate\_isa(descent\_data, self.isa\_select.get())  
  
 sorted\_data = sorted(descent\_data, key=lambda x: int(x['fl']))  
 fls = [int(point['fl']) for point in sorted\_data]  
  
 if fl\_target >= fls[0]: # Найвищий доступний FL у даних спуску  
 return {  
 'time': float(sorted\_data[0]['time']),  
 'distance': float(sorted\_data[0]['distance']),  
 'fuel': float(sorted\_data[0]['fuel'])  
 }  
 elif fl\_target <= fls[-1]: # Найнижчий доступний FL  
 return {  
 'time': float(sorted\_data[-1]['time']),  
 'distance': float(sorted\_data[-1]['distance']),  
 'fuel': float(sorted\_data[-1]['fuel'])  
 }  
  
 # Знаходимо інтервал для інтерполяції  
 for i in range(1, len(fls)):  
 if fls[i] <= fl\_target:  
 upper\_idx = i - 1 # Вищий FL  
 lower\_idx = i # Нижчий FL  
 break  
  
 # Лінійна інтерполяція  
 fl\_high = fls[upper\_idx]  
 fl\_low = fls[lower\_idx]  
 ratio = (fl\_high - fl\_target) / (fl\_high - fl\_low)  
  
 upper\_point = sorted\_data[upper\_idx]  
 lower\_point = sorted\_data[lower\_idx]  
  
 return {  
 'time': round(float(upper\_point['time']) +  
 (float(lower\_point['time']) - float(upper\_point['time'])) \* ratio, 2),  
 'distance': round(float(upper\_point['distance']) +  
 (float(lower\_point['distance']) - float(  
 upper\_point['distance'])) \* ratio, 2),  
 'fuel': round(float(upper\_point['fuel']) +  
 (float(lower\_point['fuel']) - float(upper\_point['fuel'])) \* ratio, 2)  
 }  
  
 def calculate\_total\_descent(self, fl\_start, fl\_end, calculate\_mass):  
 # Для спуску fl\_start має бути вищим за fl\_end  
 if fl\_start < fl\_end:  
 fl\_start, fl\_end = fl\_end, fl\_start  
  
 # Інтерполяція для меж  
 start\_data = self.calculate\_descent(fl\_start)  
 end\_data = self.calculate\_descent(fl\_end)  
  
 # Знаходимо всі FL з файлу в діапазоні [fl\_start, fl\_end]  
 descent\_data = self.interpolate\_mass(self.boeing\_data\_descent)  
  
 descent\_data = self.interpolate\_isa(descent\_data, self.isa\_select.get())  
  
 sorted\_fls = sorted([int(p['fl']) for p in descent\_data], reverse=True)  
 relevant\_fls = [fl for fl in sorted\_fls if fl\_end <= fl <= fl\_start]  
  
 total\_time = start\_data['time'] - end\_data['time']  
 total\_distance = start\_data['distance'] - end\_data['distance']  
 total\_fuel = start\_data['fuel'] - end\_data['fuel']  
  
 # Коригування для проміжних точок  
 if relevant\_fls:  
 first\_fl = relevant\_fls[0]  
 last\_fl = relevant\_fls[-1]  
  
 # Віднімаємо значення між кінцевими точками даних  
 first\_data = next(p for p in descent\_data if int(p['fl']) == first\_fl)  
 last\_data = next(p for p in descent\_data if int(p['fl']) == last\_fl)  
 total\_time -= (float(first\_data['time']) - float(last\_data['time']))  
 total\_distance -= (float(first\_data['distance']) - float(last\_data['distance']))  
 total\_fuel -= (float(first\_data['fuel']) - float(last\_data['fuel']))  
 if calculate\_mass:  
 self.mass\_kg = self.mass\_kg - (float(first\_data['fuel']) - float(last\_data['fuel']))  
  
 return {  
 'total\_time': abs(round(total\_time, 2)),  
 'total\_distance': abs(round(total\_distance, 2)),  
 'total\_fuel': abs(round(total\_fuel, 2))  
 }  
  
 def calculate\_climb(self, fl\_target):  
 # Сортуємо точки за значенням FL  
  
 climb\_data = self.interpolate\_mass(self.boeing\_data\_climb)  
  
 climb\_data = self.interpolate\_isa(climb\_data, self.isa\_select.get())  
  
 sorted\_data = sorted(climb\_data, key=lambda x: int(x['fl']))  
 fls = [int(point['fl']) for point in sorted\_data]  
  
 # Обробка випадків, коли FL виходить за межі даних  
 if fl\_target <= fls[0]:  
 # Повертаємо першу точку, якщо FL менше або дорівнює мінімальному  
 lower\_idx = upper\_idx = 0  
 elif fl\_target >= fls[-1]:  
 # Повертаємо останню точку, якщо FL більше або дорівнює максимальному  
 lower\_idx = upper\_idx = len(fls) - 1  
 else:  
 # Знаходимо інтервал для інтерполяції  
 for i in range(1, len(fls)):  
 if fls[i] >= fl\_target:  
 lower\_idx = i - 1  
 upper\_idx = i  
 break  
  
 # Якщо FL точно співпадає з точкою, повертаємо її значення  
 if fls[lower\_idx] == fl\_target or lower\_idx == upper\_idx:  
 point = sorted\_data[lower\_idx]  
 return {  
 'time': float(point['time']),  
 'distance': float(point['distance']),  
 'fuel': float(point['fuel'])  
 }  
 else:  
 # Лінійна інтерполяція між двома найближчими точками  
 fl\_low = fls[lower\_idx]  
 fl\_high = fls[upper\_idx]  
 ratio = (fl\_target - fl\_low) / (fl\_high - fl\_low)  
  
 lower\_point = sorted\_data[lower\_idx]  
 upper\_point = sorted\_data[upper\_idx]  
  
 # Інтерполяція значень  
 time = float(lower\_point['time']) + (  
 float(upper\_point['time']) - float(lower\_point['time'])) \* ratio  
 distance = float(lower\_point['distance']) + (  
 float(upper\_point['distance']) - float(lower\_point['distance'])) \* ratio  
 fuel = float(lower\_point['fuel']) + (  
 float(upper\_point['fuel']) - float(lower\_point['fuel'])) \* ratio  
  
 return {  
 'time': round(time, 2),  
 'distance': round(distance, 2),  
 'fuel': round(fuel, 2)  
 }  
  
 def calculate\_total\_climb(self, fl\_start, fl\_end):  
  
 climb\_data = self.interpolate\_mass(self.boeing\_data\_climb)  
  
 climb\_data = self.interpolate\_isa(climb\_data, self.isa\_select.get())  
  
 sorted\_data = sorted(climb\_data, key=lambda x: int(x['fl']))  
 fls = [int(point['fl']) for point in sorted\_data]  
  
 # Перевіряємо, чи fl\_end >= fl\_start  
 if fl\_start > fl\_end:  
 fl\_start, fl\_end = fl\_end, fl\_start  
  
 # Визначаємо всі FL у заданому діапазоні (включаючи межі)  
 all\_fls = sorted(list(set(fls + [fl\_start, fl\_end])))  
 relevant\_fls = [fl for fl in all\_fls if fl\_start <= fl <= fl\_end]  
 relevant\_fls = sorted(relevant\_fls)  
  
 # Ініціалізуємо сумарні значення  
 total\_time = 0.0  
 total\_distance = 0.0  
 total\_fuel = 0.0  
  
 # Обчислюємо різниці між послідовними точками  
 for i in range(len(relevant\_fls) - 1):  
 current\_fl = relevant\_fls[i]  
 next\_fl = relevant\_fls[i + 1]  
  
 # Отримуємо дані для поточної та наступної висоти  
 current\_data = self.calculate\_climb(current\_fl)  
 next\_data = self.calculate\_climb(next\_fl)  
  
 # Додаємо різницю до суми  
 total\_time += next\_data['time'] - current\_data['time']  
 total\_distance += next\_data['distance'] - current\_data['distance']  
 total\_fuel += next\_data['fuel'] - current\_data['fuel']  
 self.mass\_kg = self.mass\_kg - (next\_data['fuel'] - current\_data['fuel'])  
  
 return {  
 'total\_time': round(total\_time, 2),  
 'total\_distance': round(total\_distance, 2),  
 'total\_fuel': round(total\_fuel, 2)  
 }  
  
 def interpolate\_mass(self, aircraft\_data):  
  
 target\_mass = self.mass\_kg  
  
 masses = sorted(aircraft\_data.keys(), key=lambda x: int(x))  
 target\_mass\_int = int(target\_mass)  
  
 # Знаходимо найближчі маси для інтерполяції  
 lower\_mass, upper\_mass = None, None  
 for i, mass in enumerate(masses):  
 mass\_int = int(mass)  
 if mass\_int == target\_mass\_int:  
 return aircraft\_data[mass] # Якщо маса вже є у файлі  
 if mass\_int < target\_mass\_int:  
 lower\_mass = mass  
 elif mass\_int > target\_mass\_int and not upper\_mass:  
 upper\_mass = mass  
 break  
  
 # Обробка випадків, коли target\_mass за межами даних  
 if not lower\_mass:  
 return aircraft\_data[masses[0]]  
 if not upper\_mass:  
 return aircraft\_data[masses[-1]]  
  
 # Коефіцієнт для лінійної інтерполяції  
 lower\_mass\_int = int(lower\_mass)  
 upper\_mass\_int = int(upper\_mass)  
 ratio = (target\_mass\_int - lower\_mass\_int) / (upper\_mass\_int - lower\_mass\_int)  
  
 # Інтерполяція даних для кожної температури  
 interpolated = {}  
 for temp in aircraft\_data[lower\_mass]:  
 if temp not in aircraft\_data[upper\_mass]:  
 continue  
  
 # Збираємо спільні рівні польоту (FL)  
 lower\_points = {p["fl"]: p for p in aircraft\_data[lower\_mass][temp]}  
 upper\_points = {p["fl"]: p for p in aircraft\_data[upper\_mass][temp]}  
 common\_fls = sorted(  
 set(lower\_points.keys()) & set(upper\_points.keys()),  
 key=lambda x: int(x)  
 )  
  
 # Інтерполяція параметрів для кожного FL  
 temp\_data = []  
 for fl in common\_fls:  
 lower = lower\_points[fl]  
 upper = upper\_points[fl]  
  
 temp\_data.append({  
 "fl": fl,  
 "time": str(round(float(lower["time"]) + (  
 float(upper["time"]) - float(lower["time"])) \* ratio,  
 2)) if "time" in lower and "time" in upper else 0,  
 "distance": str(round(float(lower["distance"]) + (  
 float(upper["distance"]) - float(lower["distance"])) \* ratio,  
 2)) if "distance" in upper and "distance" in lower else 0,  
 "fuel": str(round(float(lower["fuel"]) + (  
 float(upper["fuel"]) - float(lower["fuel"])) \* ratio, 2)),  
 "ias": lower["ias"] if "ias" in lower else 0,  
 "tas": str(round(  
 float(lower["tas"]) + (float(upper["tas"]) - float(lower["tas"])) \* ratio,  
 2))  
 })  
  
 interpolated[temp] = temp\_data  
  
 return interpolated  
  
 def interpolate\_isa(self, data, target\_isa):  
 *"""  
 Лінійна інтерполяція даних за значенням ISA.  
 :param data: Словник з даними, де ключі верхнього рівня — значення ISA (наприклад, '0', '10').  
 :param target\_isa: Цільове значення ISA для інтерполяції.  
 :return: Інтерпольовані дані для заданого ISA.  
 """* # Конвертуємо ключі ISA в int для коректного сортування  
 isa\_keys = sorted(data.keys(), key=lambda x: int(x))  
 target\_isa\_int = int(target\_isa)  
  
 # Якщо значення ISA вже є в даних, повертаємо його  
 if str(target\_isa) in isa\_keys:  
 return data[str(target\_isa)]  
  
 # Знаходимо найближчі значення ISA  
 lower\_isa, upper\_isa = None, None  
 for isa in isa\_keys:  
 isa\_int = int(isa)  
 if isa\_int <= target\_isa\_int:  
 lower\_isa = isa  
 elif isa\_int > target\_isa\_int and upper\_isa is None:  
 upper\_isa = isa  
 break  
  
 # Обробка крайніх випадків  
 if not lower\_isa:  
 return data[isa\_keys[0]]  
 if not upper\_isa:  
 return data[isa\_keys[-1]]  
  
 # Коефіцієнт інтерполяції  
 lower\_isa\_int = int(lower\_isa)  
 upper\_isa\_int = int(upper\_isa)  
 ratio = (target\_isa\_int - lower\_isa\_int) / (upper\_isa\_int - lower\_isa\_int)  
  
 # Інтерполяція параметрів для кожного FL  
 interpolated = []  
 lower\_data = data[lower\_isa]  
 upper\_data = data[upper\_isa]  
  
 # Збираємо спільні FL  
 lower\_fls = {entry['fl']: entry for entry in lower\_data}  
 upper\_fls = {entry['fl']: entry for entry in upper\_data}  
 common\_fls = sorted(  
 set(lower\_fls.keys()) & set(upper\_fls.keys()),  
 key=lambda x: int(x)  
 )  
  
 for fl in common\_fls:  
 lower\_entry = lower\_fls[fl]  
 upper\_entry = upper\_fls[fl]  
 interpolated\_entry = {'fl': fl}  
  
 # Інтерполяція кожного параметра  
 for key in lower\_entry:  
 if key == 'fl':  
 continue  
 if key in upper\_entry:  
 try:  
 lower\_val = float(lower\_entry[key])  
 upper\_val = float(upper\_entry[key])  
 interpolated\_val = lower\_val + (upper\_val - lower\_val) \* ratio  
 interpolated\_entry[key] = round(interpolated\_val, 2)  
 except (ValueError, TypeError):  
 interpolated\_entry[key] = lower\_entry[key]  
 else:  
 interpolated\_entry[key] = lower\_entry.get(key, 0)  
  
 interpolated.append(interpolated\_entry)  
  
 return interpolated  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 root = tk.Tk()  
 app = AdvancedFuelCalculator(root)  
 root.mainloop()

Файл calculate\_route\_profile.py

import json  
import math  
import requests  
  
  
def calculate\_route\_segments(route\_points):  
 *"""  
 Функція обчислює довжини відрізків маршруту та різниці висот між послідовними точками.  
 Приймає список ідентифікаторів точок маршруту (рядки), шукає їх координати та висоту  
 у файлах airports\_esp.json та vertices\_esp.json. Повертає список словників з інформацією  
 про кожен відрізок маршруту.  
 """* # Завантажуємо дані аеропортів та вершин  
 with open('src/airports\_esp.json', 'r', encoding='utf-8') as f:  
 airports\_data = json.load(f)  
 with open('src/vertices\_esp.json', 'r', encoding='utf-8') as f:  
 vertices\_data = json.load(f)  
  
 route\_points = route\_points.split()  
 # Перетворюємо списки на словники за ключем 'ident' для швидкого пошуку  
 airports = {entry['ident']: entry for entry in airports\_data}  
 vertices = {entry['ident']: entry for entry in vertices\_data}  
  
 # Допоміжний список для збереження (ident, lat, lon, alt)  
 points = []  
 for ident in route\_points:  
 if ident in airports:  
 data = airports[ident]  
 lat = data.get('latitude')  
 lon = data.get('longitude')  
 alt = data.get('altitude')  
 if lat is None or lon is None or alt is None:  
 raise ValueError(f"Недостатньо даних для точки {ident} в airports\_esp.json")  
 points.append((ident, lat, lon, alt))  
 elif ident in vertices:  
 data = vertices[ident]  
 lat = data.get('latitude')  
 lon = data.get('longitude')  
 if lat is None or lon is None:  
 raise ValueError(f"Недостатньо даних для точки {ident} в vertices\_esp.json")  
 alt = 0  
 points.append((ident, lat, lon, alt))  
 else:  
 raise ValueError(f"Точка {ident} не знайдена у файлах даних")  
  
 distance\_km = 0  
 altitude\_fl\_start = 0  
 altitude\_fl\_end = 0  
  
 for i in range(len(points) - 1):  
 ident1, lat1, lon1, alt1 = points[i]  
 ident2, lat2, lon2, alt2 = points[i + 1]  
 # Конвертація координат з градусів в радіани  
 phi1 = math.radians(lat1)  
 phi2 = math.radians(lat2)  
 dphi = math.radians(lat2 - lat1)  
 dlambda = math.radians(lon2 - lon1)  
 # Обчислюємо гаверсинус  
 a = math.sin(dphi / 2) \*\* 2 + math.cos(phi1) \* math.cos(phi2) \* math.sin(dlambda / 2) \*\* 2  
 c = 2 \* math.asin(math.sqrt(a))  
 earth\_radius\_km = 6371.0 # Радіус Землі в км  
 distance = earth\_radius\_km \* c  
 if alt1:  
 altitude\_fl\_start = alt1 / 1000 \* 32.808  
 elif alt2:  
 altitude\_fl\_end = alt2 / 1000 \* 32.808  
  
 distance\_km += distance  
  
 return distance\_km, altitude\_fl\_start, altitude\_fl\_end