**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ФРАНКА**

Факультет прикладної математики та інформатики

(повне найменування назва факультету)

Кафедра теорії оптимальних процесів

(повна назва кафедри)

**Магістерська робота**

**“**.NET–бібліотека математичних застосунків AppliedMathLibrary**”**

Виконав: студент групи ПМАМ – 21с

спеціальності

124 «системний аналіз»

(шифр і назва спеціальності)

Байцар Р. М.

(підпис) (прізвище та ініціали)

Керівник Мельничин А. В.

(підпис) (прізвище та ініціали)

Рецензент

(підпис) (прізвище та ініціали)

Львів – 2022

ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ФРАНКА

Факультет прикладної математики та інформатики

Кафедра теорії оптимальних процесів

Спеціальність 124 «системний аналіз»

(шифр і назва)

«ЗАТВЕРДЖУЮ»  
Завідувач кафедри

" " 2022 року

ЗАВДАННЯ

НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТУ

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Байцару Ростиславу Мироновичу\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

( прізвище, ім'я, по батькові)

1 .Тема роботи розробка та впровадження “.NET–бібліотека математичних застосунків AppliedMathLibrary\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

керівник роботи Мельничин Андрій Володимирович, к. т. н., доцент кафедри теорії оптимальних процесів ,

( прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені Вченою радою факультету від " " 2022 року №

2. Строк подання студентом роботи 12 грудня 2022 р.

3. Вихідні дані до роботи

Шахно С.М. Практикум з чисельних методів / С.М. Шахно, А.Т. Дудикевич, С.М. Левицька / − Львів, 2013. − 432 с.

Зеліско В.Р. Основи лінійної алгебри і аналітичної геометрії. / В.Р. Зеліско, Г.В. Зеліско / – Львів: Львівський національний університет імені Івана Франка, 2011. – 326 с.

Сеньо П.С. Теорія ймовірностей та математична статистика / П.С. Сеньо / – К.: Центр навчальної літератури, 2004. – 448 с.

4. Зміст магістерської роботи (перелік питань, які потрібно розробити)

Розробити запланований перелік математичних методів і структур мовою C#,\_\_\_\_

Оформити програмний продукт в .NET бібліотеку на платформі NuGet\_\_\_\_\_\_\_\_\_\_

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

6. Консультанти розділів роботи

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | | |
|  |  | завдання видав | | завдання прийняв |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  | |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

7. Дата видачі завдання 2 вересня 2022 р.

Керівник роботи \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Мельничин А. В.

( підпис ) (прізвище та ініціали)

**КАЛЕНДАРНИЙ ПЛАН**

Студент\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Байцар Р. М.

( підпис ) (прізвище та ініціали)

|  |  |  |  |
| --- | --- | --- | --- |
| № з/п | Назва етапів магістерської роботи | Строк виконання етапів роботи | Примітка |
| 1 | Планування та вибір методів для програмної реалізації | Вересень |  |
| 2 | Дослідження теоретичного матеріалу | Вересень - Листопад |  |
| 3 | Програмна реалізація математичних модулів | Вересень - Листопад |  |
| 4 | Оформлення роботи | Грудень |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Зміст**

[**Магістерська робота** 1](#_Toc121659017)

[**Вступ** 5](#_Toc121659018)

[**Розділ I. аналіз предметної області та постановка задачі** 6](#_Toc121659019)

[***1.1.*** ***Постановка задачі*** 6](#_Toc121659020)

[***1.2.*** ***Вибір платформи .NET*** 7](#_Toc121659021)

[***1.3.*** ***Структура бібліотеки*** 8](#_Toc121659022)

[***1.4.*** ***Основні визначення та використані технології*** 9](#_Toc121659023)

[**Розділ II. Технічна документація** 11](#_Toc121659024)

[***2.1.*** ***Програмний об’єкт – обгортка Результат*** 11](#_Toc121659025)

[***2.2.*** ***Програмний об’єкт Матриця*** 12](#_Toc121659026)

[***2.3.*** ***Програмний об’єкт Вектор*** 15](#_Toc121659027)

[***2.4.*** ***Програмний об’єкт Точка*** 18](#_Toc121659028)

[***2.5.*** ***Програмний об’єкт Поліном*** 18](#_Toc121659029)

[***2.6.*** ***Метод Гауса*** 21](#_Toc121659030)

[***2.7.*** ***Методи Парето та Слейтера*** 22](#_Toc121659031)

[***2.8.*** ***Статистичні методи*** 23](#_Toc121659032)

[***2.9.*** ***Методи розв’язування нелінійних алгебраїчних рівнянь*** 25](#_Toc121659033)

[***2.10.*** ***Методи чисельного інтегрування*** 26](#_Toc121659034)

[***2.11.*** ***Методи поліноміального інтерполювання*** 28](#_Toc121659035)

[**Розділ III. Приклад Застосування** 30](#_Toc121659036)

[***3.1.*** ***Інструкція з встановлення бібліотеки*** 30](#_Toc121659037)

[***3.2.*** ***Тестова програма*** 31](#_Toc121659038)

[**Розділ IV. тестування** 33](#_Toc121659039)

[***4.1.*** ***Модульні тести*** 33](#_Toc121659040)

[***4.2.*** ***Тест продуктивності*** 34](#_Toc121659041)

[**Висновки** 37](#_Toc121659042)

[**Список використаних джерел** 38](#_Toc121659043)

# **Вступ**

У зв’язку з великою популярністю .NET платформи, сфера її застосування зростає, зростає також й потреба в готових програмних рішеннях, в тому числі – бібліотек.

Дана бібліотека є ідейним аналогом популярної Python бібліотеки “NumPy”. Потреба в появі бібліотеки виникла з відсутності схожого рішення для мови C#.

У період здобування ступеня бакалавра, мені довелося виконувати безліч лабораторних робіт з яких я запозичав цілі блоки коду копіюючи їх з одного завдання в інше. Вирішенням проблеми копіювання і є дана бібліотека.

Окрім цього вона включає досвід, здобутий протягом років навчання на факультеті «прикладної математики та інформатики» та поширює його в середовище .NET спільноти.

# **Розділ I. аналіз предметної області та постановка задачі**

## ***Постановка задачі***

Усі реалізовані методи та об’єкти у даній роботі базуються на моєму досвіді набутому у процесі здобуття ступеня бакалавра «системного аналізу», факультету прикладної математики (applied math).

Від назви факультету і походить назва бібліотеки. Оскільки термінологія, найменування та реалізація певних методів породжена термінологією, використовуваною викладачами мого університету, підкреслити це у назві я вважаю доречним.

Необхідно розробити бібліотеку, що реалізовує основні методи та об’єкти, що найбільш поширені при розв’язуванні більш складних та нетривіальних задач.

Вирішити проблему копіювання коду та надати безліч готовий рішень.

Забезпечити високу надійність та продуктивність коду, в той же час простоту та зрозумілість.

Найбільш поширеними та часто використовуваними об’єктами є:

* Вектор
* Точка
* Матриця
* Поліном

Ці прости математичні абстракції формалізують абсолютну більшість проблем, що вирішує прикладна математика.

Оскільки дані об’єкти вміють взаємодіяти з іншими об’єктами та володіють певними властивостями, ці властивості потрібно формалізувати в коді. Додавання матриць породжує нову матрицю, множення поліномів утворює новий поліном. Всю базову алгебраїчну взаємодію потрібно передбачити.

Також для розв’язування окремих, простих, але поширених задач необхідно реалізувати наступні методи:

* Метод Гауса
* Методи Парето та Слейтера
* Методи розв’язування алгебричних нелінійних рівнянь (метод Ньютона та січних)
* Чисельне інтегрування
* Чисельне інтерполювання методом Лагранжа
* Обчислення оберненої матриці
* Деякі статистичні методи (наприклад середнє значення, медіана, мода, мат. очікування, дисперсія, тощо)

Передбачити масштабованість бібліотеки та здатність до впровадження нових методів. Реалізувати систему версій, релізів, забезпечити ведення журналу змін.

## ***Вибір платформи .NET***

Оскільки .NET для мене це основна платформа написання програмного забезпечення, а мої знання в інших середовищах розробки обмежені, питання вибору навіть не стояло.

В процесі проходження переддипломної практики мені довелося працювати в команді досвідчених C# розробників, що сприяло обміну досвідом та дозволило покращити якість кінцевого результату.

Окрім того, на просторах NuGet важко знайти схожу бібліотеку, що розв’язує поставлений список задач.

## ***Структура бібліотеки***

Усі методи та об’єкти розділено по окремих директоріях, а сама структура має ієрархічну, деревовидну структуру.

В процесі розвитку бібліотеки та з виходом версії 2.0.0 відбилися певні зміни у файловій структурі проекту. Це необхідно для кращого та більш гармонійного структурування об’єктів. Оскільки неможливо передбачити якою має бути фінальна структура – цей процес перетворюється у неперервне реструктурування, щоб забезпечити найкраще розміщення наявних файлів та об’єктів. Це також є великим недоліком, оскільки зміни у файловій структурі породжують несумісність версій бібліотеки.

Наступний малюнок відображає зміну в будові бібліотеки

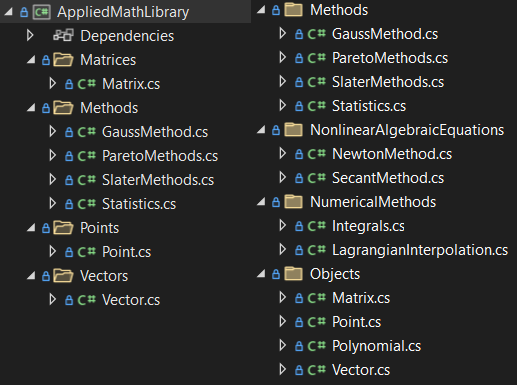


Рис. 1.1. Порівняння файлової структура бібліотеки AppliedMathLibrary v1.0.0 (зліва) та v2.0.0 (справа)

Легко зауважити що версія 2.0.0 містить більшу кількість об’єктів. З’явилися методи обчислення нелінійних алгебричних рівнянь, інтегралів, інтерполювання методом Лагранжа. Існуючі об’єкти матриця, вектор, точка та новий поліном тепер розташовані в папці Objects, що відображає предметну суть перелічених об’єктів.

## ***Основні визначення та використані технології***

При написанні програмної бібліотеки доведеться познайомитися з новими термінами, які я не використовував у своїх попередніх курсових роботах.

*Бібліотека* *(library)* – збірка об'єктів чи підпрограм для вирішення близьких за тематикою задач. У комп'ютерних науках бібліотеки розділяють на статичні та динамічні.

У випадку моєї роботи мова йде про динамічну бібліотеку.

*Динамічні бібліотеки* (або розподілена бібліотека (shared library), або бібліотеками, що динамічно підключаються (Dynamic Link Library, DLL). Це окремі файли, що надають програмі набір використовуваних функцій для завантажування на етапі виконання при зверненні програми до ОС із заявкою на виконання функції з бібліотеки. Якщо необхідна бібліотека вже завантажена в оперативну пам'ять, програма використовуватиме завантажену копію бібліотеки. Такий підхід дозволяє зекономити час і пам'ять, оскільки декілька програм використовують одну копію бібліотеки, вже завантажену в пам'ять.

*NuGet* – система керування пакунками, розроблена компанією Microsoft. Створений у 2010–му, NuGet фактично є єдиним пакетним менеджером .NET світу, популярність якого не уступає пакетним менеджерам JavaScript та Python. NuGet повністю інтегрований у Visual Studio, де має простий на зручний графічний інтерфейс.

*.NET* – програмна технологія, запропонована фірмою Microsoft як платформа для створення програмних рішень будь якої складності.

*Статичні методи (або властивості)* – це методи, виклик яких здійснюється від імені класу в якому вони знаходяться. Таким чином метод абстрагується від екземпляру класу та дозволяє працювати з будь якими вхідними параметрами. Вважається потокобезпечним (thread–safe).

*Інкапсуляція –* В парадигмі ООП програмування, означає приховування від кінцевого користувача деталей реалізації і неможливість впливати на внутрішні процеси інкапсульованого об’єкту.

*Модульне тестування (Unit testing)* – це метод тестування програмного забезпечення, який полягає в окремому тестуванні кожного модуля коду програми. Модулем називають найменшу частину програми, яка може бути протестованою. В об'єктно–орієнтованому програмуванні ця найменша частина — метод. Unit–тести розробляються разом з частиною коду яку вони зобов’язані тестувати. Зазвичай unit–тести необхідні для того, щоб упевнитися, що код відповідає вимогам архітектури та має очікувану поведінку.

*Тест продуктивності (benchmark)* – як і unit–тест має на меті перевірити програму на швидкодію на об’єм використаної пам’яті. Це дозволяє виокремити кращі та гірші методи для вирішення певної задачі, або знайти області аргументів коли той чи інший метод є доцільнішим для використання. Кращими є ті методи що споживають менше пам’яті та працюють швидше.

*Git* – система контролю версій що дозволяє фіксувати кожен крок написання програми та зберігати його на віддаленому репозиторію.

*Microsoft Visual Studio 2022* – потужне IDE для розробки програмних продуктів на базі .NET. Версія 2022 є необхідною для використання .NET 6.

*.NET* — це модульна платформа для розробки програмного забезпечення різного типу з відкритим вихідним кодом. Варто зауважити, що у версії 6 порушена зворотна сумісність з попередніми версіями.

*BenchmarkDotNet* – бібліотека з відкритим програмним кодом для написання тестів продуктивності.

*xUnit* – популярний в середовищі .NET фреймворк для написання unit–тестів. Надає широкий спектр можливостей у тестуванні.

*FluentAssertions* – бібліотека – розширення xUnit для написання тестів сценарії яких більше походять на людську мову (в даному випадку англійську). Приклад тесту з перевіркою результату виконання методами FluentAssertions: “actualResult.Should().BeEquivalentTo(expectedResult);”.

# **Розділ II. Технічна документація**

## ***Програмний об’єкт – обгортка Результат***

Існують випадки коли метод, що обчислює складну та довготривалу задачу заходить у ситуацію коли стає очевидно, що розв’язати дану задачу успішно неможливо, тому потрібно завершити процес виконання. Рішення з викиданням винятку (exception) цілком прийнятне та робоче, але існує краще, при якому користувачу немає необхідності ловити та обробляти виняток.

У версії 2.0.0 було представлено нову структуру Result, що виступає обгорткою над результатом виконання метода і може бути як успішною так і невдалою. Оскільки це структура, Result не може бути рівним null, це підвищує надійність використання такого підходу.

Наступні властивості та методи описують структуру Result:

IsFailure – властивість що вказує чи є результат невдалим;

IsSuccess – властивість що вказує чи є результат успішним.

Дані дві властивості є взаємо оберненими і не можуть мати однакове значення одночасно.

Error – властивість що містить текстовий опис причини через яку результат виконання невдалий. У випадку, якщо результат успішний це поле пусте.

Value – властивість що містить результат успішного виконання методу. У випадку, якщо результат невдалий, містить значення за замовчуванням.

static Result<T> Success<T>(T value) – метод, що створює успішний результат зі значенням value типу T (T може бути числом, матрицею, вектором чи будь яким іншим типом даних у C#).

static Result<T> Failure<T>(string error) – метод, що створює невдалий результат з причиною error.

Даний об’єкт – обгортка покращує якість коду, оскільки робить методи більш дескриптивними та передбачуваними. Також це запобігає небажаним винятковим ситуаціям.

Простий приклад створення невдалого результату виконання операції:

return Result.Failure<Matrix>("Час операції вийшов");

Приклад роботи з результатом у головному потоці:

if (receivedResult.IsFailure) // do action

Варто зазначити, що даний підхід не заміняє викидання винятків, а лише доповнює його. В деяких ситуаціях винятки є більш доречним або єдино можливим рішенням.

## ***Програмний об’єкт Матриця***

Матриця – це сукупність математичних величин, певним способом розміщених у прямокутній таблиці.

Бібліотека реалізовує довільну матрицю розміром n x m, елементами якої є дійсні числа (тип double). Очевидно, що n та m повинні бути більшими за 0.

Для створення об’єкту Matrix реалізовано наступні конструктори:

Matrix(int n, int m) – створює матрицю n x m де всі елементи 0;

Matrix(int n, int m, params double[] values) – створює матрицю n x m де всі елементи передані через параметр values. Очікується що кількість елементів values буде рівна n\*m;

Matrix(int n, int m, IEnumerable<double> values) – аналогічно з попереднім конструктором;

Matrix(int n) – створює матрицю n x n де всі елементи 0;

Matrix(int n, params double[] values) – створює матрицю n x n де всі елементи передані через параметр values. Очікується що кількість елементів values буде рівна n\*n;

Matrix(int n, IEnumerable<double> values) – аналогічно з попереднім конструктором;

Matrix(Matrix matrix) – створює нову матрицю на основі переданої (конструктор копіювання);

Matrix(params Vector[] vectors) – створює матрицю з векторів де кожен вектор стає рядком матриці. Очікується, що всі вектори однакової розмірності та передано принаймні один вектор;

Властивості об’єкту Matrix:

bool IsSquare – повертає true якщо матриця квадратна;

int Rows – повертає кількість рядків матриці (число n);

int Columns – повертає кількість стовпців матриці (число m);

double this[int i, int j] – індексатор. Дозволяє звертатися до кожного елементу матриці як до двовимірного масиву;

Методи об’єкту Matrix:

static Matrix Transpose(Matrix matrix) – створює нову транспоновану матрицю на основі переданої. Статичний метод;

Matrix Transpose() – транспонує матрицю;

List<Vector> ToVectors() – розкладає матрицю на масив векторів, де кожен вектор це рядок матриці;

Matrix Subtract(Matrix matrix) – віднімає від матриці передану. Результатом операції є нова матриці. Очікується, що передана матриця схожої розмірності;

static Matrix Subtract(Matrix A, Matrix B) – аналогічно до попереднього методу повертає різницю двох матриць. Статичний метод;

static Matrix operator –(Matrix A, Matrix B) – аналогічно до попередніх двох методів повертає різницю двох матриць. Статичний метод. Дозволяє використовувати оператор “–” (наприклад A – B);

Vector MultiplyBy(Vector vector) – множення матриці на вектор справа. Результатом є новий вектор. Очікується, що розмірність вектора рівна кількості стовпців матриці;

static Vector Multiply(Matrix matrix, Vector vector) – аналогічно до попереднього, множення матриці на вектор справа. Статичний метод;

static Vector operator \*(Matrix matrix, Vector vector) – аналогічно до попередніх двох методів, множення матриці на вектор справа. Статичний метод. Дозволяє використовувати оператор “\*” (наприклад M \* v);

static Matrix Divide(Matrix matrix, double scalar) – статичний метод що повертає нову матрицю, кожен елемент якої поділений на переданий скаляр. Очевидно що скаляр не може бути рівним 0;

Matrix DivideBy(double scalar) – по аналогії з попереднім методом створює нову матрицю це кожен елемент матриці поділений на скаляр, що не рівний 0;

static Matrix operator /(Matrix matrix, double scalar) – аналогічно до попередніх двох, дозволяє використовувати оператор “/” при діленні матриці на скаляр, що нерівний 0. Повертає нову матрицю;

static Matrix CreateIdentityMatrix(int n) – статичний метод що створює прямокутну матрицю розмірності n, де всі елементи 0, окрім елементів головної діагоналі, які рівні 1;

Result<double> CalculateDeterminant() – рекурсивним чином обчислює визначник матриці. Повертає результат обчислення;

static Result<double> CalculateDeterminant(Matrix matrix) – аналогічно до попереднього обчислює визначник матриці. Повертає результат обчислення. Статичний метод;

Result<Matrix> CalculateInverse() – обчислює обернену матрицю рекурсивним чином обчислюючи мінори даної матриці. Повертає результат обчислення;

static Result<Matrix> CalculateInverse(Matrix matrix) – ідентичний статичний метод обчислення оберненої матриці. Повертає результат обчислення;

IEnumerator<double> GetEnumerator() – дозволяє використовувати матрицю в методі foreach.

Загалом матриця не зазнала суттєвих змін у новій версії, було лише додано нові методи: ділення на скаляр, створення одиничної матриці, обчислення визначника та обертання матриці.

## ***Програмний об’єкт Вектор***

Вектор — математичний об'єкт, який характеризується величиною і напрямком. Найбільш поширена математична абстракція.

Бібліотека реалізовує довільний вектор розміром n , елементи якого є дійсні числа (тип double). Очевидно, що n повинно бути більшим за 0.

Для створення об’єкту Vector реалізовано наступні конструктори:

Vector(int n) – створює вектор розміром n, де всі елементи 0;

Vector(params double[] values) – створює вектор на основі переданих значень параметру values. Розмірність n обчислюється з кількості переданих значень;

Vector(Vector vector) – створює копію переданого вектора

(конструктор копіювання);

Vector(Point point) – створює вектор на основі точки. Точка від веткора відрізняється лише методами (поведінкою), тому це фактично конструктор копіювання.

Властивості об’єкту Vector:

int Dimension – повертає розмірність вектора;

double this[int i] – індексатор. Дозволяє звертатися до кожного елементу вектора як до масиву.

Методи об’єкту Vector:

bool CompareByValueWith(Vector vector) – порівнює два вектори за значеннями. Якщо значення відповідних координат векторів рівні то вектори рівні. Очікує вектори однакових розмірностей;

static bool CompareByValue(Vector vector1, Vector vector2) – аналогічно до попереднього методу. Статичний метод;

static bool Comparable(Vector vector1, Vector vector2) – визначає чи два вектори можна порівнювати. Статичний метод;

bool ComparableWith(Vector vector) – аналогічно до попереднього методу, визначає чи можна порівнювати вектор з переданим;

Vector Subtract(Vector vector) – віднімає від вектора переданий вектор. Очікує вектор тієї самої розмірності. Повертає новий вектор;

static Vector Subtract(Vector vector1, Vector vector2) – аналогічно до попереднього методу, повертає різницю двох векторів. Статичний метод;

static Vector operator –(Vector vector1, Vector vector2) – аналогічно до попереднього методу, повертає різницю двох векторів. Статичний метод. Дозволяє використовувати оператор “–” (наприклад v1 – v2);

static Vector Sum(Vector vector1, Vector vector2) – повертає новий вектор, що є результатом додавання двох переданих векторів. Очікує, що вектори однакової розмірності. Статичний метод;

Vector Add(Vector vector) – аналогічно до попереднього методу, повертає суму двох векторів;

static Vector operator +(Vector vector1, Vector vector2) – аналогічно до попередніх двох методів, повертає суму двох векторів. Статичний метод. Дозволяє використовувати оператор “+” (наприклад v1 + v2);

Vector DivideBy(double scalar) – дозволяє розділити всі елементи вектора на скаляр. Повертає новий вектор. Скаляр не може бути рівним 0;

static Vector Divide(Vector vector, double scalar) – аналогічно до попереднього методу. Статичний метод;

static Vector operator /(Vector vector, double scalar) – аналогічно до попередніх двох методів. Статичний метод. Дозволяє використовувати оператор “/” (наприклад v / a);

Vector MultiplyBy(double scalar) – домножає скаляр до вектора. Повертає новий вектор;

static Vector Multiply(Vector vector, double scalar) – аналогічно до попереднього методу. Статичний метод;

static Vector operator \*(Vector vector, double scalar) – аналогічно до попередніх двох методів. Статичний метод. Дозволяє використовувати оператор “\*” (наприклад v \* a);

double Norm() – обчислює евклідову норму даного вектора.

static double Norm(Vector vector) – обчислює евклідову норму переданого вектора. Статичний метод;

double DistanceTo(Vector vector) – обчислює евклідову відстань до переданого вектора. Очікує, що вектори однакової розмірності;

static double DistanceBetween(Vector vector1, Vector vector2) – аналогічно до попереднього методу обчислює відстань між векторами. Статичний метод;

IEnumerator<double> GetEnumerator() – дозволяє використовувати вектор в методі foreach.

Загалом жодних змін цей об’єкт у версії 2.0.0 не зазнав. Було виправлено незначні помилки у внутрішній роботі деяких методів, проведено оптимізації використання пам’яті, покращено документацію.

## ***Програмний об’єкт Точка***

Точка позначає точне положення в просторі. Це матеріальний об'єкт, що позначає місце або стан.

В даній бібліотеці точка схожа на вектор, але має значно менший функціонал.

Конструктори об’єкту Point:

Point(int n) – створює точку розміром n, де всі елементи 0;

Point(params double[] values) – створює точку на основі переданих значень параметру values. Розмірність n обчислюється з кількості переданих значень;

Point(Point point) – створює копію переданої точки (конструктор копіювання);

Point(Vector vector) – створює точку на основі переданого вектора. Точка від веткора відрізняється лише методами (поведінкою), тому це фактично конструктор копіювання.

Властивості об’єкту Point:

int Dimension – повертає розмірність точки;

double this[int i] – індексатор. Дозволяє звертатися до кожного елементу точки як до елементу масиву;

Методи об’єкту Point:

IEnumerator<double> GetEnumerator() – дозволяє використовувати точку в методі foreach.

Як і з вектором була проведена робота по оптимізації роботи з пам’яттю, покращенню документації, виправленню незначних помилок.

## ***Програмний об’єкт Поліном***

Поліном це досить часто використовувана математична сутність, що описує велику множину задач і/або допомагає їх вирішувати. Наприклад в даній бібліотеці поліном необхідний для методів чисельного інтерполювання описаного далі в цьому розділі.

Програмна реалізація описана класом Polynomial. Будь який поліном можна однозначно описати його коефіцієнтами. Це дозволяє динамічно створювати поліном довільної розмірності за допомогою наступних конструкторів:

Polynomial(IEnumerable<double> coefficients) – створює поліном з масиву злічуваних дійсних чисел;

Polynomial(params double[] coefficients) – створює поліном з масиву дійсних чисел переданих в конструктор, розділених комою або як об’єкт типу Array;

Polynomial(Polynomial other) – конструктор копіювання. Створює новий поліном з ідентичними коефіцієнтами полінома other.

Усі конструктори очікують що буде передано принаймні один коефіцієнт для створення полінома, а самі коефіцієнти передані у порядку від наймолодшого до найстаршого степеня, тобто a0, a1, a2, …, an.

Також для зручної роботи з поліномом реалізовано наступні методи та властивості:

int HighestPower – властивість що повертає степінь найвищого ненульового коефіцієнта;

double this[int i] – індексатор. Дозволяє звертатися до кожного елементу полінома як до елементу масиву;

static Polynomial Multiply(Polynomial first, Polynomial second) – статичний метод множення двох поліномів, результатом якого є новий поліном;

static Polynomial Multiply(Polynomial polynomial, double scalar) – статичний метод множення полінома на скаляр, результатом якого є новий поліном;

Polynomial Multiply(Polynomial other) – множення даного полінома на поліном other, результатом є новий поліном;

Polynomial MultiplyBy(double scalar) – множення даного полінома на скаляр, результатом є новий поліном;

static Polynomial operator \*(Polynomial first, Polynomial second) – оператор «\*» для множення двох поліномів, результатом є новий поліном;

static Polynomial operator \*(Polynomial polynomial, double scalar) – оператор «\*» для множення полінома на скаляр справа, результатом є новий поліном;

static Polynomial operator \*(double scalar, Polynomial polynomial) – оператор «\*» для множення полінома на скаляр зліва, результатом є новий поліном;

static Polynomial Sum(Polynomial first, Polynomial second) – статичний метод сумування двох поліномів, результатом є новий поліном;

Polynomial Add(Polynomial other) – метод що додає до поточного полінома поліном other, результатом є новий поліном;

static Polynomial operator +(Polynomial first, Polynomial second) – оператор «+» для додавання двох поліномів, результатом є новий поліном;

static Polynomial Subtract(Polynomial first, Polynomial second) – віднімання від полінома first поліном second, результатом є новий поліном;

Polynomial Subtract(Polynomial other) – віднімання від поточного полінома поліном other, результатом є новий поліном;

static Polynomial operator -(Polynomial first, Polynomial second) – оператор «-», віднімає від полінома first поліном second, результатом є новий поліном;

static Polynomial Divide(Polynomial polynomial, double scalar) – статичний метод, що ділить кожен коефіцієнт поліному на переданий скаляр. Скаляр повинен бути відмінним від нуля. Повертає новий поліном;

Polynomial DivideBy(double scalar) – ділить кожен коефіцієнт поліному на скаляр, відмінний від нуля. Повертає новий поліном;

static Polynomial operator /(Polynomial polynomial, double scalar) – оператор «/», ділить кожен коефіцієнт поліному на скаляр, відмінний від нуля. Повертає новий поліном;

double[] ToArray() – метод, що повертає масив коефіцієнтів поліному у порядку від наймолодшого степеня до найстаршого;

IEnumerator<double> GetEnumerator() – дозволяє використовувати поліном в методі foreach.

## ***Метод Гауса***

Метод Гауса – класичний метод для розв’язування систем лінійних алгебричних рівнянь.

В даній бібліотеці метод представлений статичним класом GaussMethod, що повністю інкапсулює логіку розв’язування СЛАР.

Розв’язування відбувається зведенням вхідної матриці до верхньотрикутної, що дозволяє також знайти визначник вхідної матриці як побічний ефект.

Розв’язок відбувається викликом методу SolveMatrixSystem.

Сигнатура методу змінилася у версії 2.0.0 і має наступний вигляд:

static Result<Vector> SolveMatrixSystem(Matrix A, Vector b, out double detA, CancellationToken ct = default) – де A це квадратна матриця розмірності n x n, b – вектор стовпець розмірності n. Вихідний параметр detA визначник матриці A, обчислений в ході розв’язання системи.

Цей статичний метод повертає результат розв’язування СЛАР що містить вектор розв’язок або причину невдачі розв’язку задачі.

Також у новій версії було додано підтримку токенів скасування, це дозволяє безпечно припинити виконання методу, якщо в якийсь момент часу нам перестає бути важливим результат виконання. Це дозволяє зекономити ресурси комп’ютера. За замовчуванням метод закінчує своє виконання якщо процес триває довше 5 секунд. Щоб змінити цю поведінку, необхідно передати параметром новий, відмінний від default, токен.

Оскільки перед початком роботи вхідні параметри копіюються, метод можна вважати thread–safe (потоко - безпечним).

## ***Методи Парето та Слейтера***

Метод Парето є одним з найпростіших та найпопулярніших у теорії прийняття рішень. Кожна раціональна людина по повсякденному житті користується цим методом підсвідомо. Обрати найкращу альтернативу з множини доступних і є оптимальним рішенням.

У бібліотеці метод Парето представлений статичним класом ParetoMethods та має наступні статичні методи:

static bool BetterByParetoThan(this Vector current, Vector vector) – метод розширення об’єкта Vector. Оцінює чи поточний вектор альтернатив є кращим за переданий вектор за принципом Парето. Очікує що два вектори однакової розмірності;

static bool CompareByPareto(Vector vector1, Vector vector2) – аналогічно до попереднього, оцінює чи vector1 є кращим за vector2 за принципом Парето;

static List<Vector> BestByPareto(IEnumerable<Vector> vectors) – метод, що знаходить найкращі альтернативи з множини переданих. Ітераційний метод, що послідовно відкидає доміновані за Парето альтернативи. Очікує що всі вектори однакової розмірності;

При згадці метода Парето, зазвичай зустрічається метод Слейтера, що є тим самим методом відбору альтернатив зі строгішою умовою: краща альтернатива повинна бути кращою за всіма параметрами.

Статичний клас SlaterMethods містить наступні статичні методи:

static bool BetterBySlaterThan(this Vector current, Vector vector) – метод розширення об’єкта Vector. Оцінює чи поточний вектор альтернатив є кращим за переданий вектор за принципом Слейтера. Очікує що два вектори однакової розмірності;

static bool CompareBySlater(Vector vector1, Vector vector2) – аналогічно до попереднього, оцінює чи vector1 є кращим за vector2 за принципом Слейтера;

static List<Vector> BestBySlater(IEnumerable<Vector> vectors) – метод, що знаходить найкращі альтернативи з множини переданих. Ітераційний метод, що послідовно відкидає доміновані за Слейтером альтернативи. Очікує що всі вектори однакової розмірності.

Жодних змін ці два об’єкти у версії 2.0.0 не зазнали.

## ***Статистичні методи***

Досить часто при роботі з масивами даних доводиться проводити статистичну оцінку цих даних.

Для вирішення цієї проблеми у бібліотеці присутній статичний клас Statistics, що поєднює в собі наступні статистичні методи:

static double Mean(IEnumerable<double> items) – метод, що знаходить середнє значення переданого масиву чисел;

static double Median(IEnumerable<double> items) – метод, що знаходить медіану переданого масиву чисел, значення, що розташоване посередині масиву;

static double[] Mode(IEnumerable<double> items) – метод, що знаходить моду переданого масиву чисел, значення, що зустрічаються найчастіше. Мод може бути декілька;

static double MathExpectation(IEnumerable<double> items, IEnumerable<double> vectorProbability) – метод, що обчислює математичне сподівання переданого масиву чисел. Параметр vectorProbability позначає розподіл ймовірностей відповідних елементів масиву items. Очікується, що кількість елементів у двох вхідних масивах рівна, а сума ймовірностей рівна 1;

static double Var(IEnumerable<double> items) – метод, що обчислює дисперсію переданого масиву чисел. Центральним моментом є середнє значення;

static double Var(IEnumerable<double> items, IEnumerable<double> vectorProbability) – аналогічно до попереднього методу, обчислює дисперсію масиву чисел, при заданому розподілі ймовірностей елементів масиву items. Очікується, що кількість елементів у двох вхідних масивах рівна, а сума ймовірностей рівна 1;

static double Std(IEnumerable<double> items) – метод, що обчислює стандартне відхилення (середнє квадратичне відхилення) переданого масиву чисел;

static double Std(IEnumerable<double> items, IEnumerable<double> vectorProbability) – аналогічно до попереднього методу, обчислює стандартне відхилення масиву чисел, при заданому розподілі ймовірностей елементів масиву items. Очікується, що кількість елементів у двох вхідних масивах рівна, а сума ймовірностей рівна 1;

static List<double> GenerateRandomArray(int length, double min = 0, double max = 1) – метод, що створює масив заданої довжини, елементами якого є випадкові числа на відрізку [min; max];

static List<T> RandomSampleByRule1<T>(IEnumerable<T> generalSample, int n) – метод, що випадковим відбором без повернення бере з переданого масиву даних n елементів. Результатом є новий масив даних;

static List<T> RandomSampleByRule2<T>(IEnumerable<T> generalSample, int n) – аналогічно до попереднього методу, бере з переданого масиву даних n елементів. Метод є повільнішим, проте потребує менше пам’яті для роботи.

## ***Методи розв’язування нелінійних алгебраїчних рівнянь***

Постановка цієї задачі звучить доволі просто: знайти точку перетину функцією осі OX або знайти x з рівняння F(x) = 0.

Тим не менше в теорії чисельних методів це завдання не таке тривіальне і потребує деяких обчислювальних затрат.

Для розв’язування даної задачі в бібліотеку з версією 2.0.0 було додано два нових методи:

* метод Ньютона, представлений статичним класом NewtonMethod;
* метод січних (хорд), розміщений у статичному класі SecantMethod.

Метод Ньютона містить наступний метод розв’язування рівнянь:

static Result<double> SolveEquation(Func<double, double> f, Func<double, double> fDerivative, double x = 1, [Range(0.1, 1)] double eps = Constants.Epsilon, CancellationToken ct = default) – це статичний метод, що шукає нуль заданої функції f  
() з точністю eps (за замовчуванням eps = 0.000001). Для коректної роботи необхідно вказати похідну функції f’ (параметр fDerivative, ). Якщо користувач знає початкове наближення його слід вказати як параметр x, інакше за початкове наближення буде взято 1.

Як і в методі Гауса – додано підтримку токенів скасування. Це необхідно, оскільки не кожна функція f перетинається з віссю OX, відповідно й не має розв’язку задачі. Таким чином, за логікою алгоритму, метод знайде точку, що найближче лежить до осі OX та буде стрибати в її околі, проте умова закінчення ітераційного процесу не виконається ніколи і через 5 секунд (за замовчуванням) токен скасування закінчить виконання процесу.

Метод Ньютона описується наступною ітераційною формулою:

У випадку коли користувач не може вказати похідну функції аналітично, доцільно використати метод січних. Цей метод використовує більше часу для обчислення, але маєш менш жорсткі вхідні умови.

Сигнатура методу січних має наступний вигляд:

static Result<double> SolveEquation(Func<double, double> f, double x = 1, [Range(0.1, 1)] double eps = Constants.Epsilon, CancellationToken ct = default) – аналогічно методу Ньютона, цей має ті самі властивості, проте тут відсутній параметр fDerivative. Це означає, що похідна початкової функції обчислюється чисельно на кожній ітерації. Такий підхід дещо сповільнює роботу методу, проте, як показують тести продуктивності, не критично. Таким чином метод січних це хороша альтернатива методу Ньютона.

Формульне представлення методу січних:

## ***Методи чисельного інтегрування***

Інтегрування це важлива та поширена задача яка часто зустрічається у повсякденному житті оскільки це, фактично, множення нелінійної функції та довжину інтервалу на якому ця функція визначена.

Ідея чисельного інтегрування полягає у розбитті фігури, що описується функцією на безліч прямокутників та сумуванні їхніх площ. За теорією, чим дрібніше розбиття, тим точніший результат обчислення.

Клас Integrals містить наступні методи:

static Result<double> RectanglesMethod(Func<double, double> f, double a, double b, IntegrationOptions options = IntegrationOptions.Full, [Range(Constants.Epsilon, 1)] double h = Constants.Epsilon, CancellationToken ct = default) – цей метод розбиває підінтегральну функцію f, на заданому проміжку інтегрування [a, b], на безліч прямокутників шириною h (за замовчуванням h = 0.000001) та рахує їхню суму зі специфічними опціями інтегрування (параметр options). Висота кожного прямокутника визначається з середини основи. Токен скасування має значення 5 секунд за замовчуванням. Метод повертає результат обчислення інтегралу;

static Result<double> TrapeziumMethod(Func<double, double> f, double a, double b, IntegrationOptions options = IntegrationOptions.Full, [Range(Constants.Epsilon, 1)] double h = Constants.Epsilon, CancellationToken ct = default) – ідентичний попередньому метод. Єдина відмінність це розбиття підінтегральної фігури на безліч трапецій та сумування їхніх площ.

Обидва методи параметризовані інтегральними опціями, що змінюють результат обчислення. У версії 2.0.0 присутні наступні 4 опції типу IntegrationOptions:

* Full – обчислює повне значення інтегралу на проміжку [a, b] сумуючи всі доданки за модулем.
* UpperOX – обчислює площу лише на тих інтервалах [a, b] де підінтегральна фігура розташована вище осі OX.
* LowerOX – на противагу до попередньої опції, обчислює значення інтегралу де фігура розташована нижче осі OX.
* UpperMinusLower – від площі фігури що розташована вище осі OX віднімає площу фігури що розташована нижче осі OX.

Наступний малюнок ілюструє принцип роботи цих чотирьох опцій

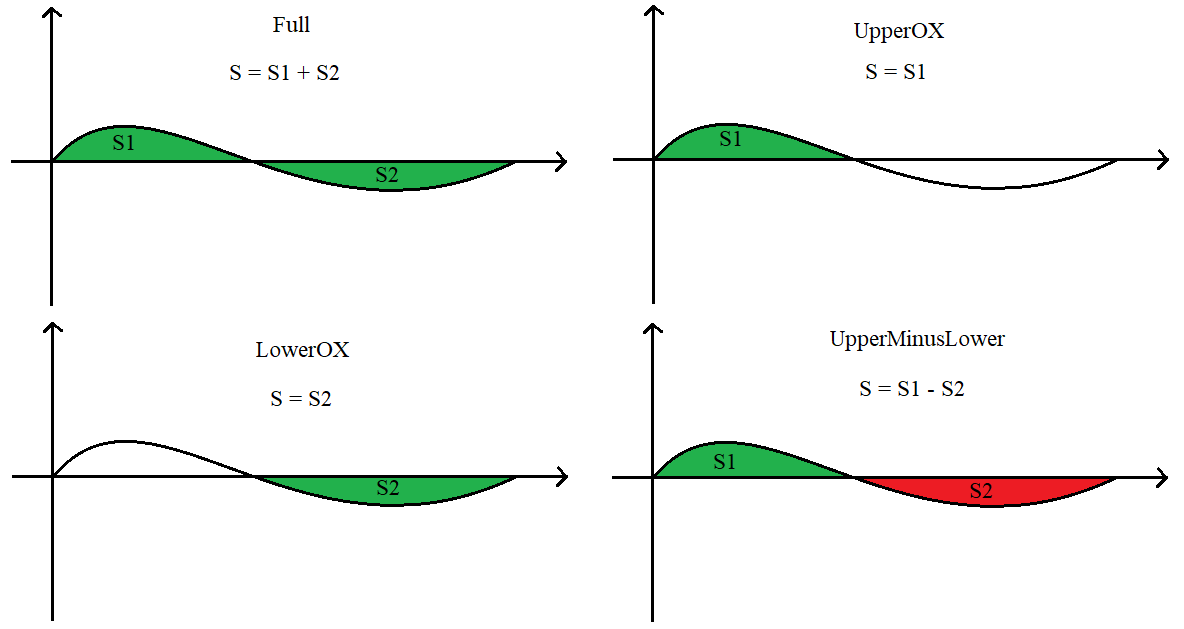


Рис. 2.1. Ілюстрація принципу обчислення площі при заданні різних опцій інтегрування.

## ***Методи поліноміального інтерполювання***

Цікавим класом задач є побудова інтерполяційних поліномів заданих скінченною кількість пар значень функцій та їх аргументів. Теоретично відомо, що шукану функцію f можна як завгодно точно наблизити інтерполяційним поліномом p, якщо функція f неперервна та гладка на визначеному проміжку.

В даній бібліотеці інтерполювання представлене статичним класом LagrangianInterpolation, що містить наступні методи:

static Result<Func<double, double>> CreatePolinimial(IEnumerable<double> xAxisValues, IEnumerable<double> yAxisValues, CancellationToken ct = default) – статичний метод, що шукає коефіцієнти інтерполяційного поліному та будує функцію що наближує початкову функцію, задану масивом значень цією функції (параметр yAxisValues) та масивом аргументів (параметр xAxisValues). Очікується що ці два масиви мають однакову розмірність, та значення під однаковим індексом відповідні (тобто ). Повертає результат створення інтерполяційної функції p (). Підтримує токен скасування (5 секунд за замовчуванням);

static Result<double[]> CalculatePolynomialCoefficients(IEnumerable<double> xAxisValues, IEnumerable<double> yAxisValues, CancellationToken ct = default) – метод, ідентичний попередньому, але результатом роботи якого є масив коефіцієнтів інтерполяційного полінома. Коефіцієнти подані в порядку він найменшого степеня до найвищого (a0, a1, a2, …, an).

# **Розділ III. Приклад Застосування**

## ***Інструкція з встановлення бібліотеки***

Оскільки бібліотека вільно поширюється пакетним менеджером NuGet, знайти її можна за ім’ям «AppliedMathLibrary».

Покрокова інструкція із встановлення у Visual Studio:

1. створити новий проект у Visual Studio на основі .NET 6 або відкрити існуючий. Важливо саме .NET 6 або вище;
2. клікнувши правою кнопкою мишки по відкритому проекті у solution explorer, вибрати Manage NuGet Packages..;
3. у відкритому вікні, на вкладці Browse ввести у вікно пошуку «AppliedMathLibrary»;
4. встановити знайдену бібліотеку натиском кнопки Install.

A screenshot of a computer

Description automatically generated with medium confidence

Рис. 3. 1. Встановлення бібліотеки AppliedMathLibrary.

Станом на грудень 2022 року актуальною версією бібліотеки є 2.0.0.

## ***Тестова програма***

Для того щоб зрозуміти як саме працює бібліотека та яку проблему вона вирішує, потрібно написати програму з наступними вимогами:

* Користувач вводить матрицю та вектор вільних членів СЛАР.
* Розв’язати СЛАР та вивести результат на екран.

Для вирішення проблеми написано просту консольну аплікацію, результат роботи якої представлений на наступному рисунку

Graphical user interface, text

Description automatically generated  
Рис. 3. 2. Результат роботи тестової програми.

Вихідний код програми зображений на Рис. 3. 3.

Text

Description automatically generated  
Рис. 3. 3. Вихідний код тестової програми.

Як видно з малюнку 4, весь код програми, що розв’язує складну математичну задачу помістився у декілька лінійок. Додатково метод SolveMatrixSystem встановив визначник введеної матриці.

Ще один приклад пошуку оберненої матриці та роботі з матрицями зображений на насптупному рисунку

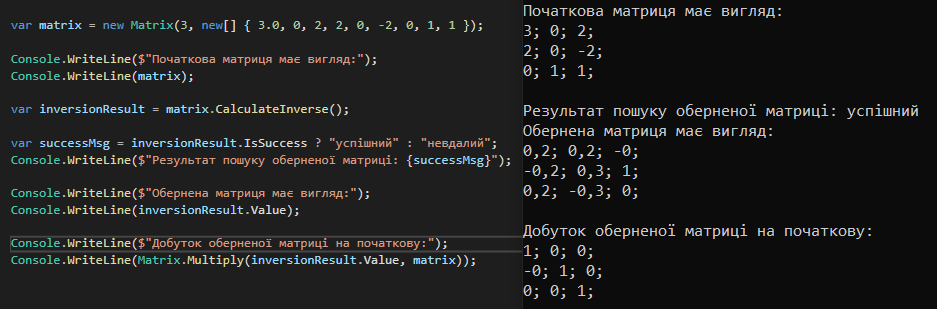


Рис. 3. 4. Приклад програми, що працює з оберненою метрицею.

Також на даному прикладі видно взаємодію з об’єктом Result. Оскільки поле IsSuccess тримає значення true, можна говорити про успішне обчислення оберненої матриці, і як свідчить останній рядок програми, множення оберненої матриці на початкову дало одиничну матрицю.

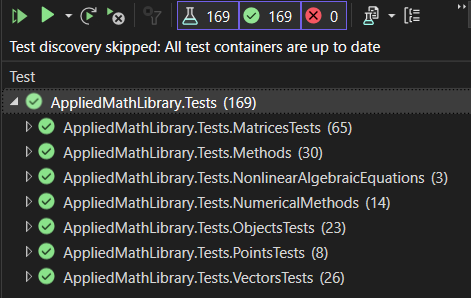
Як і будь який ітераційний метод, деякі значення накопичують похибку заукруглення. Звідти й зв’являються значення, такі як -0, проте насправді це значення приблизно рівне -0,000001. Зазвичай такою похибкою можна знехтувати.

# **Розділ IV. тестування**

## ***Модульні тести***

У момент написання будь якої бібліотеки немає можливості практично застосувати написаний метод. Єдиним рішенням проблеми є модульні тести.

Для забезпечення високої надійності рішень, що бібліотека пропонує, написано понад сотню тестів з різноманітними сценаріями використання.

  
Рис. 4. 1. Модульні тести бібліотеки AppliedMathLibrary.

Також тести відіграють важливу роль для збереження надійності коду, роблять код толерантним до змін. Оскільки при розширенні кодової бази передбачається внесення змін, тести показують порушення в першопочатковій логіці програми. Це може захистити від ненавмисного пошкодження початкового коду.

## ***Тест продуктивності***

Для вирішення деяких проблем користувач може спіткнутися з обмеженням у часі, або оперативній пам’яті комп’ютера, варто оцінити ефективність методів.

Для вирішення цієї проблеми слугують так звані бенчмарки (або тести продуктивності).

Для прикладу оцінимо два схожі між собою методи: RandomSampleByRule1 та RandomSampleByRule2.

Завдання полягає в наступному: для спільної вибірки з 1000 елементів здійснити відбір 100, 500 та 900 елементів.

Результат оцінки зображену на малюнку:

Calendar

Description automatically generated  
Рис. 4. 2. Результат тестування на продуктивність двох схожих методів.   
1 us = 0.000001 sec.

Як видно із результатів метод RandomSampleByRule1 працює швидше, у всіх трьох випадках, проте виділяє більше оперативної пам’яті.

Більш цікавими з точки зору поведінки з пам’яттю та швидкодією будуть методи добавлені у версію 2.0.0.

Розглянемо як працюють методи обчислення інтегралів для різних функцій, на різних проміжках інтегрування, з різними інтеграційними опціями.

Наступний рисунок зображає результат інтегрування методом прямокутників та трапецій функцій sin(x) та на відповідних проміжках та [-10000, 10000]. Крок h = 0,000001.

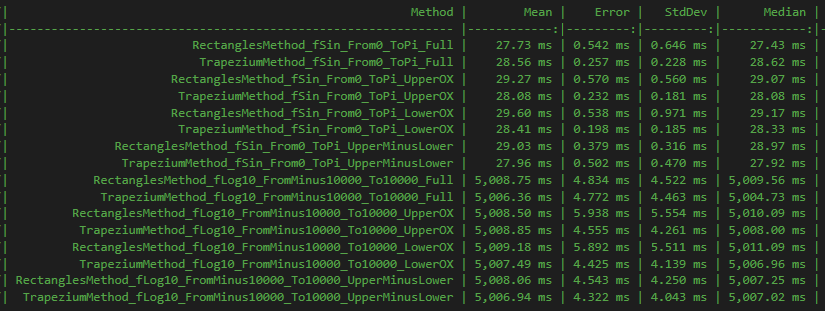


Рис. 4. 3. Результат тестування функцій інтегрування.

Як видно з рисунку, навіть на великих проміжках інтегрування метод працює швидко, а опції інтегрування не впливають на швидкодію.

Пошук оберненої матриці видає наступні результати тестування

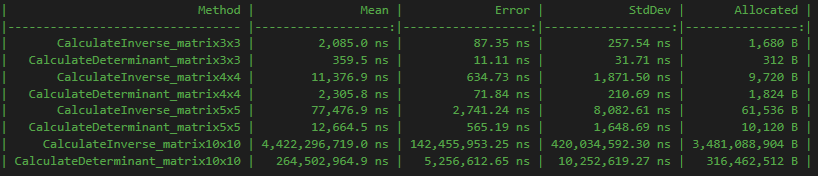


Рис. 4. 4. Результат тестування функцій пошуку оберненої матриці та пошуку визначника.

Ці результати кардинально відрізняються від попередніх тестів. Тут зі збільшенням розмірності матриці відбувається швидкий ріст використаних ресурсів. Також, оскільки для пошуку матриці обчислюється її визначник, результати такого пошуку наведені на тому ж рисунку. На кожній ітерації методу створюється нова матриця меншої розмірності для пошуку мінорів. Саме через створення нових матриць витрачається багато пам’яті. Але така «жертва» необхідна для забезпечення принципу thread-safe.

Наступними тестами оцінимо роботу двох схожих методів для розв’язку нелінійних алгебричних рівнянь, порівняємо роботу методу Ньютона та методу січних (хорд).

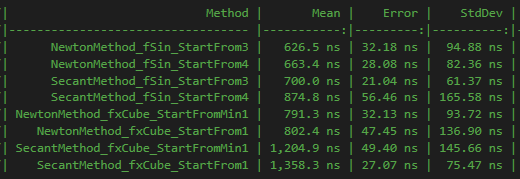


Рис. 4. 4. Результат тестів продуктивності методів Ньютона та січних

Для оцінки вибрано дві різні функції sin(x) з різними початковими наближеннями (з точки 3 та 4, розв’язком є точка 3.14) та функція (з початковими наближеннями з точки -1 та 1, розв’язком є точка 0).

З результату видно, що метод Ньютона працює швидше у всіх випадках, проте метод січних не значно уступає. Обидва методи працюють швидко. 1 ns = 0.000000001 sec.

Як і з обчисленням інтегралів, пам’ять майже не використовується, оскільки методи пере використовують свої внутрішні змінні.

# **Висновки**

Бібліотека розроблена та знаходиться у вільному доступі для всіх бажаючих.

Особливо корисною буде для студентів – бакалаврів прикладної математики. Безліч рішень реалізовано на основі літератури університету, це може допомогти у розумінні тих чи інших методів.

На даний момент доступна вже друга «велика» версія даного продукту під номером 2.0.0. Додано безліч нових методів, що розширюють базовий функціонал, виправлено дрібні неполадки, що були допущені у версії 1.0.0, покращено досвід користування методами, проте дані зміни є несумісними з попередньою версією.

Кожен бажаючий може вплинути на розвиток даного продукту, запропонувавши свої ідеї для втілення, або допомогу у виправлені помилок.

Вихідний код бібліотеки знаходиться у вільному доступі за посиланнями:

<https://github.com/Rostik18/AppliedMathLibrary>  
 Сторінка бібліотеки на сайті пакетного менеджера NuGet:  
 <https://www.nuget.org/packages/AppliedMathLibrary/>

# **Список використаних джерел**

1. Шахно С.М. Практикум з чисельних методів / С.М. Шахно, А.Т. Дудикевич, С.М. Левицька / − Львів, 2013. − 432 с.

2. Зеліско В.Р. Основи лінійної алгебри і аналітичної геометрії. / В.Р. Зеліско, Г.В. Зеліско / – Львів: Львівський національний університет імені Івана Франка, 2011. – 326 с.

3. Сеньо П.С. Теорія ймовірностей та математична статистика / П.С. Сеньо / – К.: Центр навчальної літератури, 2004. – 448 с.

4. Василик О.І. З теорії і методів вибіркових обстежень. Навчальний посібник / О.І. Василик, Т.О. Яковенко / – Київський національний університет імені Тараса Шевченка, ВПЦ "Київський університет", 2010. – 206 с.

5. Install and manage packages in Visual Studio using the NuGet Package Manager. [Електронний ресурс]. – Режим доступу: [https://docs.microsoft.com/en–us/nuget/consume–packages/install–use–packages–visual–studio](https://docs.microsoft.com/en-us/nuget/consume-packages/install-use-packages-visual-studio)

6. Inverse of a matrix. [Електронний ресурс]. – Режим доступу: <https://www.mathsisfun.com/algebra/matrix-inverse.html>

7. Байцар Р. М. .NET–бібліотека математичних застосунків AppliedMathLibrary / Байцар Р. М. // Міжнародна студентська наукова конференція з питань прикладної математики та комп’ютерних наук (МСНКПМКН – 2022), 5 - 6 травня 2022 р. – Львів:2022. - С. 88 - 89. [Електронний ресурс]. – Режим доступу: <https://ami.lnu.edu.ua/wp-content/uploads/2022/05/ISSCAMCS-2022.pdf>