20 РОКІВ

ЛЬВІВСЬКОМУ ДЕРЖАВНОМУ УНІВЕРСИТЕТУ ВНУТРІШНІХ СПРАВ

Львівський державний університет внутрішніх справ

Тарас РУДИЙ

Олег ЗАЧЕК

Чисельні методи

Навчальний посібник

Львів

2025

УДК 519.6
Р 83

Рекомендовано до друку та розміщення в електронних сервісах ЛьвДУВС
Вченою радою Львівського державного університету внутрішніх справ
(протокол від 29 грудня 2025 року № 6)

Р е ц е н з е н т и:

Щур І. З., доктор технічних наук, професор
(Національний університет «Львівська політехніка»)

Борецька І. Б., кандидат технічних наук, доцент
(Національний лісотехнічний університет України)

Огірко О. І., кандидат технічних наук, доцент
(Львівський державний університет внутрішніх справ)

Рудий Т. В., Зачек О. І.

Р 83 Чисельні методи : навчальний посібник. Львів : Львівський державний університет внутрішніх справ, 2025. 184 с.

ISBN 978-617-511-439-1

Навчальний посібник є складовою навчально-методичного забезпечення цієї дисципліни, а основна мета видання – допомогти здобувачам освітнього ступеня «бакалавр» засвоїти основні підходи до проєктування та розроблення проєктів програмних кодів у розв'язанні практичних задач мовою програмування С++ та із застосуванням математичного пакету MathCad Prime 9.0. Акцентовано на практичному розв'язанні одних і тих же задач із застосуванням математичного пакету MathCAD та розробленні проєктів програмних кодів мовою С++.

Для здобувачів першого (бакалаврського) рівня вищої освіти спеціальності 126 «Інформаційні системи та технології» (F6 «Інформаційні системи і технології»).

The textbook «Numerical Methods» is a component of the educational and methodological support of this discipline, and the main goal of the textbook is to help applicants for the «bachelor» degree to master the basic approaches to designing and developing software code projects when solving practical problems in the C++ programming language and using the MathCad Prime 9.0 mathematical package. The main emphasis in the textbook is on the practical solution of the same problems using the MathCAD mathematical package and developing software code projects in the C++ language.

For applicants of the first (bachelor's) level of higher education in the specialty: 126 «Information Systems and Technologies» (F6 «Information Systems and Technologies»).

УДК 519.6

ISBN 978-617-511-439-1

ЗМІСТ

ВСТУП

Дисципліна «Чисельні методи» – це фундаментальна математична дисципліна, навчальний матеріал якої ґрунтується на використанні знань із математичного аналізу, лінійної алгебри тощо, і є основою для розроблення алгоритмів, обґрунтування вибору методів розв’язання задач, проєктування програмних кодів та інформаційних систем.

Основний акцент зосереджено на комплексному вивченні технологій програмного реалізування типових алгоритмів, використовуючи сучасні чисельні методи для розв’язання задач у галузі математичного аналізу, лінійної алгебри, оброблення експериментальних даних із застосуванням пакету MathCad Prime 9.0 та розробленні проєктів програмних кодів мовою С++.

Програма навчальної дисципліни «Чисельні методи» складена відповідно до освітньо-професійної програми підготовки здобувачів освітнього ступеня «бакалавр» галузі знань 12 «Інформаційні технології» спеціальності 126 «Інформаційні правоохоронні системи».

Зважаючи на фаховий напрям підготовки здобувачів вищої освіти як майбутніх інженерів та обсяг і місце дисципліни в освітній програмі, основна увага у посібнику зосереджена на викладенні матеріалу та набутті знань, важливих для початку роботи здобувачами з PTC MathCad Prime 9.0, які є необхідними для розв’язання задач наближеними методами.

У результаті вивчення навчальної дисципліни відповідно до вимог освітньо-професійної програми здобувачі вищої освіти повинні набути необхідних компетентностей.

Реалізуванню мети і завдань дисципліни «Чисельні методи» підпорядковано і структуру посібника, яка спрямована на досягнення логічної послідовності викладення теоретичного матеріалу курсу, підготовано базу

для встановлення зв’язку між теоретичним матеріалом і лабораторним практикумом.

Наразі рівень підготовки випускників ЗВО визначається тим набором спеціалізованих програмних інструментів, якими вони володіють на професійному рівні. Тому для здобувачів освітнього ступеня бакалавра, окрім глибинних знань із фундаментальних дисциплін, перше місце посідає завдання навчання технологіям, які повсюдно використовуються у бізнесі, науці та промисловості.

Один із таких інструментів – програмний пакет Mathcad Prime 9.0. MathCad – один із найбільш поширених застосунків для виконання інженерних розрахунків і тому став своєрідним стандартом.

Чому для вивчення дисципліни «Чисельні методи» укладачі посібника обрали Mathcad Prime 9.0?

MathCad – система, комп’ютерної алгебри з класу систем автоматизованого проєктування, орієнтована на створення інтерактивних документів з обчисленнями та візуальним супроводом, відрізняється легкістю використання та застосування для колективної роботи. Дуже популярна у технічних ЗВО та активно використовується студентами та викладачами.

Mathcad був задуманий і спочатку створений Алленом Раздовом з Массачусетського технологічного інституту, співзасновником компанії Mathsoft, яка з 2006 року є частиною корпорації PTC (Parametric Technology Corporation).

Робота у MathCad здійснюється у межах робочого аркуша, на якому рівняння та вирази відтворюються графічно, на противагу текстам програмних кодів у мовах програмування високого рівня. У створенні MathCad-документів використовується принцип WYSIWYG (*What You See Is What You Get* – «що бачиш, те й отримуєш»).

У цьому класі програмного забезпечення є багато аналогів різного спрямування та принципу побудови. Найчастіше MathCad порівнюють з такими програмними комплексами, як Maple, Mathematica, MATLAB. Проте об’єктивне

порівняння ускладняється з огляду на різне призначення програм та ідеологію їх використання.

Система Maple, наприклад, призначена головно для виконання аналітичних (символьних) обчислень і має для цього один із найпотужніших у своєму класі арсенал спеціалізованих процедур та функцій. Така комплектація для більшості користувачів, які стикаються з необхідністю виконання математичних розрахунків середнього рівня складності, є надмірною. Можливості Maple орієнтовані на користувачів – професійних математиків. Розв'язання завдань у середовищі Maple вимагають як уміння оперувати довільною функцією, так і знання методів розв'язання, які у ньому вмонтовані: у багатьох вмонтованих функціях Maple фігурує аргумент, який задає метод розв'язання.

Те ж можна сказати і про Mathematica. Це одна з найпотужніших систем, яка має надзвичайно велику функціональну наповненість. Mathematica має високу швидкість обчислень, але вимагає вивчення досить своєрідної мови програмування.

Розробники MathCad наголосили на розширенні системи відповідно до потреб користувача. MathCad, на відміну від Maple, спочатку створювався для числового розв'язання математичних задач, він орієнтований на розв'язання завдань властиво *прикладної*, а не *теоретичної* математики, коли необхідно отримати результат без заглиблення у математичну суть задачі.

Символьний процесор MathCad, на відміну від оригінального Maple (MuPAD), штучно обмежений (доступно приблизно 300 функцій), але цього у більшості випадків цілком достатньо для розв'язання інженерних задач.

Основна відмінність MathCad від аналогічних програм – це графічний, а не текстовий режим введення математичних та логічних виразів. Для набору команд, функцій, формул можна використовувати як клавіатуру, так і піктограми на численних спеціальних панелях інструментів. У будь-якому разі формули матимуть звичний, аналогічний до книжкового вигляд. Тобто особливої підготовки для набору виразів не потрібно.

Обчислення з введеними формулами здійснюються за бажанням користувача: або миттєво, одночасно з набором, або за командою. Звичайні формули обчислюються зліва праворуч і зверху вниз (за схожістю до читання тексту). Довільні змінні, вирази, параметри можна змінювати, спостерігаючи за відповідними змінами результату у режимі реального часу. Це дозволяє організувати інтерактивні обчислювальні документи.

В інших програмах обчислення здійснюються у режимі програмного інтерпретатора, який трансформує у формули, введені у вигляді тексту, команди. Для користування MathCad можна взагалі не бути знайомим із програмуванням у тому або іншому вигляді.

MathCad проєктувався як засіб програмування без програмування, але, якщо виникає така потреба, MathCad має досить прості для опанування інструменти програмування, які дають можливість реалізовувати дуже складні алгоритми у разі, коли вмонтованих засобів розв'язання завдання не вистачає, а також коли необхідно виконувати циклічні розрахунки.

Окремо слід відзначити можливість використання у розрахунках Mathcad Prime 9.0 величин з розмірностями, причому можна вибрати систему одиниць: СІ, СГС, МКС, англійську, або створити власну. Результати обчислень, зрозуміло, також набувають відповідної розмірності. Користь від такої можливості важко переоцінити, оскільки значно спрощується відстеження помилок у розрахунках, особливо у фізичних та інженерних.

Автори використали термін, який трапився нам уперше. Це *Engineering Notebook* (Інженерний Блокнот). Як трактувати цей термін і який зміст розробники у нього вкладали, було незрозумілим. Окрім того, у літературі нема опису роботи з пакетом Mathcad Prime 9.0 навіть на рівні інтерфейсу, який радикально відрізняється від версій 14 та 15 пакетів MathCad. Проте у процесі практичної роботи дещо з'ясувалося.

Отже, у нашому розумінні, *Engineering Notebook* – це робочий простір, який об'єднує ядра для числових та символьних обчислень, графічний та текстовий процесор, систему інтегрування з іншими програмними

продуктами, підтримку OLE-технологій. Таке поєднання цих інструментальних засобів у межах єдиного функціоналу створює потужний програмний механізм для розв'язання інженерних задач та формування звітів і документації.

За результатами вивчення дисциплін «Алгоритмізація та програмування» і «Об'єктно-орієнтоване програмування» здобувачі уже мають достатній рівень знань для використання при розв'язанні тих же задач мовою програмування С++. Тому, у процесі виконання лабораторного практикуму, будуть виконувати завдання з використанням пакету Mathcad Prime та мови програмування С++.

Теоретична частина посібника має довідково-методичний характер і може використовуватися здобувачами як у повному обсязі, так і окремими частинами залежно від конкретної задачі, яка розв'язується ними у цей момент, а також і у процесі подальшого навчання та професійної діяльності.

У посібнику на численних прикладах подано, як розв'язуються задачі числового аналізу (розв'язання лінійних, нелінійних рівнянь та їх систем, диференціювання та інтегрування функцій, розв'язання диференційних рівнянь тощо). Числові методи розглядаються лише у тому обсязі, який необхідний для розуміння роботи відповідних вмонтованих функцій середовища програмування.

Посібник розроблено насамперед для користувачів, які вперше вивчають пакет MathCad.

Очевидно, що деякі питання розглядаються подекуди двічі – спершу з метою найпростішого розв'язання отриманого завдання (аби користувач не загубив його мету), а вдруге з використанням додаткових можливостей (форматування) системи.

Опанувати усі можливості пакету MathCad дуже важко і загалом не потрібно. Той необхідний мінімум, який може знадобитися на практиці для здобувачів або практичних працівників поліції, можна цілковито осягнути протягом кількох вечорів. Усе решта забезпечиться вашою інтуїцією та практичним досвідом.

Автори висловлюють щиру подяку рецензентам

Особливо хочемо висловити подяку д. т. н., професору Паранчукові Ярославу Степановичу, з яким постійно консультувалися у процесі роботи над посібником і враховували його поради.

Укладачі завжди будуть раді обміну думками та дискусії стосовно викладеного матеріалу.

РОЗДІЛ 1

ВСТУП ДО ЧИСЕЛЬНИХ МЕТОДІВ. ЗАГАЛЬНІ ПОНЯТТЯ

1.1. Загальні відомості про пакет MathCad Prime

Наразі є багато математичних пакетів. Поважно трактується у сфері інженерної спільноти пакет (застосунок) MathCad Prime. Він призначений для виконання науково-інженерних розрахунків довільної складності і доволі простий у опануванні. MathCad Prime є інтегрованою системою програмування, орієнтованою на проведення математичних, інженерно-технічних, статистичних та економічних розрахунків.

Нами використовується версія MathCad Prime 9.0, оскільки вона надана університетові за ліцензією виробника.

MathCad Prime містить: текстовий процесор, обчислювальний, символьний та графічний процесори, довідкову систему.

Текстовий процесор слугує для введення та редагування текстів.

Обчислювальний процесор здійснює числові обчислення за заданими математичними формулами, має великий набір вмонтованих математичних функцій, забезпечує обчислення сум та добутків, означених/неозначених інтегралів, похідних тощо.

Символьний процесор дозволяє отримувати результати обчислень у символьному (аналітичному) вигляді.

Графічний процесор слугує для створення графіків функцій однієї та двох змінних.

Крім того, рівняння та графіки у пакеті MathCad Prime взаємодіють в режимі реального часу. При зміні довільних даних, змінних або рівнянь MathCad Prime негайно перераховує математичні вирази та переформатовує графіки.

1.2. Програмне вікно MathCad Prime

Програмне вікно містить такі елементи:

- кнопка MathCad Prime;
- стрічка;
- панель швидкого доступу;
- контекстне меню;
- робоча область;
- рядок стану.

Під час активування пакета MathCad Prime відкривається порожній документ, до якого можна додати текст, рівняння, графіки та зображення. Загальний вигляд вікна пакету MathCad Prime подано на рис. 1.1.

Програмне вікно містить такі елементи: кнопка MathCad Prime, Стрічка, панель швидкого доступу, контекстне меню, робоча область, рядок стану. Розглянемо функціональне призначення цих елементів.

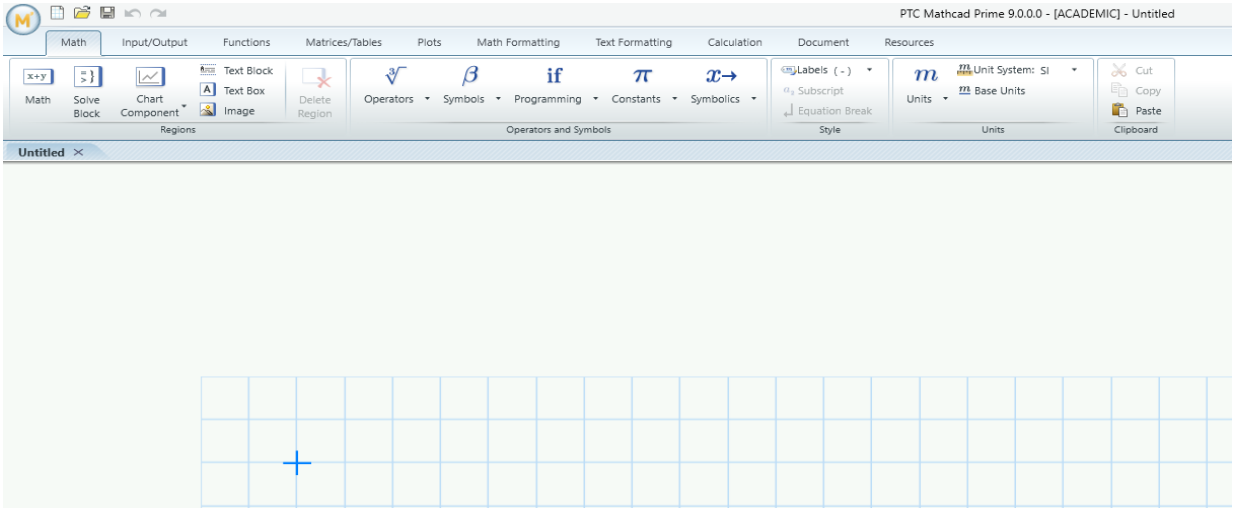


Рис. 1.1. Вікно пакета MathCad Prime

Кнопка MathCad Prime (рис. 1.2) містить команди для роботи з файлами:

- *New* (Створити);
- *Open* (Відкрити);
- *Save* (Зберегти);

- *Save as* (Зберегти як);
- *Print* (Друк);
- *Close* (Закрити);
- *Exit* (Вихід) – вихід із пакету MathCad, якщо немає необхідності у збереженні документа.

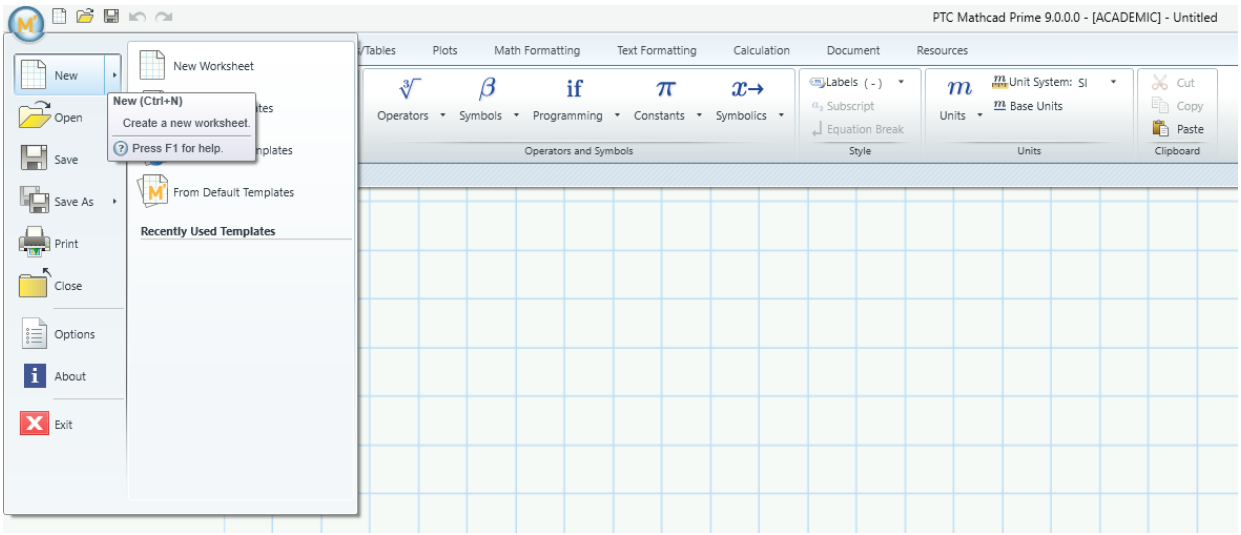


Рис. 1.2. Кнопка MathCad Prime

Стрічка, як елемент багатьох сучасних програмних застосунків, складається з вкладок (інтерфейсних груп), на яких згруповані відповідні команди. На вкладках *Стрічки* відтворюються кнопки (піктограми), призначені для виконання тих чи інших команд, причому кнопки згруповані у контекстно-залежні меню.

Основною відмінністю MathCad Prime від попередніх версій, з пункту бачення користувача, є інтерфейс, який ґрунтується на застосуванні *Стрічки*. Ті операції, які у попередніх версіях здійснювалися за допомогою набірних панелей і діалогових вікон, тепер виконуються за допомогою елементів керування у *Стрічці*.

Робочу область можна налаштувати, зменшивши або збільшивши стрічку і додавши команди, які найчастіше використовуються користувачем, на панель швидкого доступу.

Панель швидкого доступу містить команди, які є актуальними для користувача. Команди можна додавати та вилучати з панелі. Панель швидкого доступу може знаходитись під або над стрічкою.

У Контекстному меню, якщо клацнути на документі правою кнопкою миші, можна отримати доступ до актуальних команд.

Рядок стану містить номери сторінок документа, кольоровий кружечок, який показує стан кнопок: *«Знайти»*, *«Замінити на»*, *«Параметри пошуку»*, *«Регулятор масштабування»*, *«Подання сторінки»* та *«Режим чорновика»*.

Робота з довідкою (рис. 1.3).

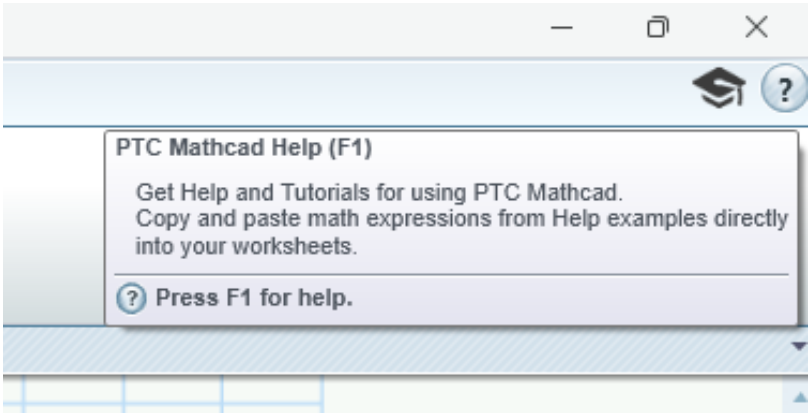


Рис. 1.3. Контекстне меню *Help*

Центр допомоги надає доступ до довідки та прикладів MathCad Prime. Команди для роботи з довідковою системою зосереджені у меню *Help* (Довідка). Центр допомоги відкривається активуванням значка у правому верхньому куті стрічки MathCad або натисканням клавіші *F1*.

Пакет MathCad містить практичні приклади, розроблені для того, щоб продемонструвати використання обчислювальних можливостей MathCad, а також вмонтованих функцій та операторів.

Робоче поле програмного вікна MathCad знаходиться під стрічкою і у ньому розташовується одне або кілька вікон MathCad-документів.

Можна відкрити кілька документів одночасно. Кожен відкритий документ розгортається у новій вкладці під стрічкою. Щоб перейти до потрібного документа, активуйте його вкладку.

У нижній частині програмного вікна знаходиться *Рядок станів*, в якому під час підведення курсора миші відтворюються підказки про функціональне призначення елементів на панелі інструментів, режим обчислень MathCad та номер поточної сторінки.

Створення MathCad-документа може полягати у введенні та редагуванні математичних виразів, формуванні графічних об'єктів, виконанні математичних розрахунків, введенні текстових коментарів, пояснень тощо. Введення математичних виразів до MathCad-документу та виконання математичних і символьних обчислень здійснюється у математичній або текстовій області.

Кожен MathCad-документ може містити довільну кількість *математичних областей* та *текстових областей*, які об'єднуються робочим простором *Engineering Notebook*.

Math (Математична область) містить конструкції пакета MathCad, необхідні для реалізування того або іншого алгоритму розв'язання задачі. У цій же області будуються графіки. Курсор у математичній області має форму *синього хрестика*.

Text Box (Текстова область) може включати довільні символи (українські, латинські, грецькі) як коментарі до обчислювальних фрагментів. Для створення текстової області необхідно активувати вкладку *Document* (рис. 1.4).

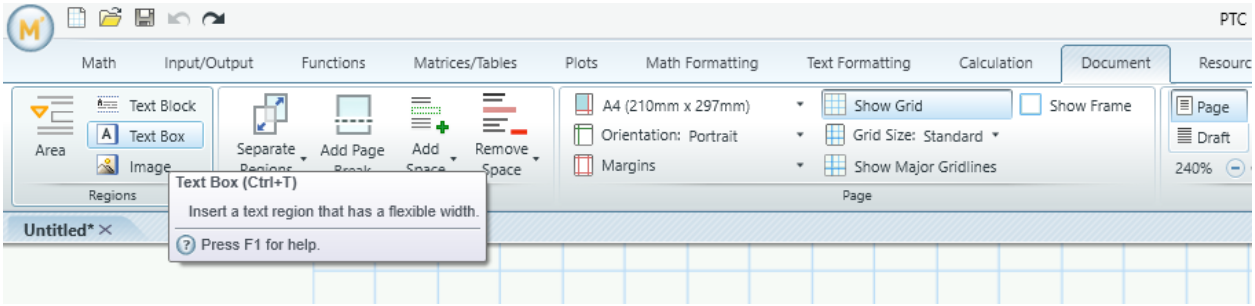


Рис. 1.4. Активування команди *Text Box*

До MathCad-документ можна додавати текст і зображення.

Текстова область може бути створена так: установіть курсор у потрібному місці документа; на вкладці *Math* у розділі *Regions* активуйте *Text Box* або *Text Block*.

Текст буде відтворено у прямокутнику, який утворить текстову область. Введіть текст у поле. За потреби текстова область автоматично розширюється під час введення тексту. Для введення порожнього рядка натисніть *Enter*. Щоб завершити режим введення тексту, клацніть лівою клавішею миші поза межами текстової області.

Наявні такі типи текстових областей.

Text Box. Розташовується за всією шириною сторінки. Ця область не перекривається з іншими областями MathCad-документа. Блоки тексту використовуються для вставлення великих параграфів у сторінку.

Text Block (Текстове поле). Цю область можна переміщати документом, і вона може перекриватися іншими областями. Текстові поля використовуються для розміщення коротких приміток та підзаголовків для математичних виразів.

Нагадуємо, що за замовчуванням установлюється *математична область* введення.

Для редагування текстової області потрібно активувати її, клацнувши на ній лівою клавішею миші. З'явиться курсор у вигляді вертикальної риски. За допомогою вкладки *Text Formatting* можна змінити тип шрифту, розмір, стиль та колір елементів текстового поля.

Примітка. Налаштування стилю шрифту, означені у вкладці *Text Formatting,* будуть застосовуватися до всіх нових областей тексту у межах актуального MathCad-документу.

1.3. Стрічковий інтерфейс пакету MathCad Prime

Стрічковий інтерфейс MathCad Prime містить вкладки у вигляді рядка меню. Кожна вкладка містить *розділи*, які об'єднані у *групи*. Розділ, своєю чергою, містить *оператори* і *значки*, сукупність яких називатимемо *палітрою інструментів*.

Стрічковий інтерфейс версії MathCad Prime 9.0 подано на рис. 1.5.

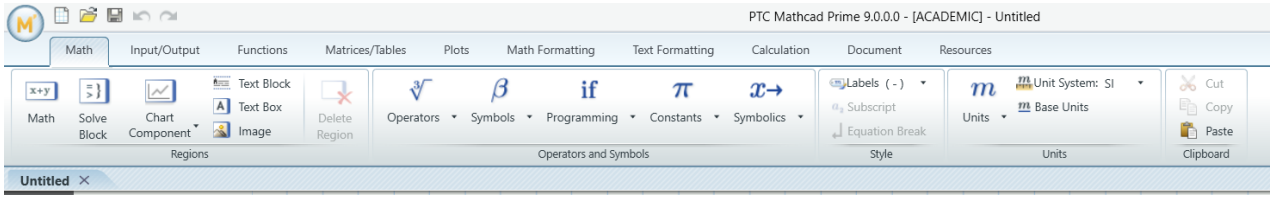


Рис. 1.5. Стрічковий інтерфейс версії MathCad Prime 9.0

У стрічці MathCad Prime доступні для використання такі вкладки:

- *Math* (*Математика)*;
- *Input/Output* (*Введення/Виведення)*;
- *Functions* (*Функції)*;
- *Matrices/Tables* (*Матриці/Таблиці)*;
- *Plots* (*Графіки)*;
- *Math Formatting* (*Форматування формул)*;
- *Text Formatting* (*Форматування тексту)*;
- *Calculation* (*Розрахунок)*;
- *Document* (*Документ)*;
- *Resources* (*Ресурси)*.

Вкладка Math (*Математика*)

Вкладка *Math* містить такі розділи: *Operators* (*Оператори*), *Symbols* (*Символи*), *Programming* (*Програмування*,) *Constants* (*Константи*), *Symbolic Operators and Keywords* (*Символьні операції*) (рис. 1.6).

Розділ *Operators* містить математичні оператори, які розподілені за категоріями: Algebra (*Алгебра)*, Equation Break (*Розрив формули*), *Calculus Comparison (Математичний аналіз)*, *Definition and Evaluation (Визначення та обчислення), Engeneering* (*Проєктування*), *Vector and Matrix* (*Вектори і матриці)* (рис. 1.6).

За допомогою цих знаків можна вводити у документи MathCad практично всі відомі математичні символи та оператори.

За допомогою цих знаків можна вводити у документи MathCad практично всі відомі математичні символи та оператори.

Розділ *Symbols* містить великі та малі літери грецького алфавіту, математичні символи та символи грошових одиниць.

Розділ *Programming Operators* містить оператори, які використовуються під час реалізування модульного програмування.

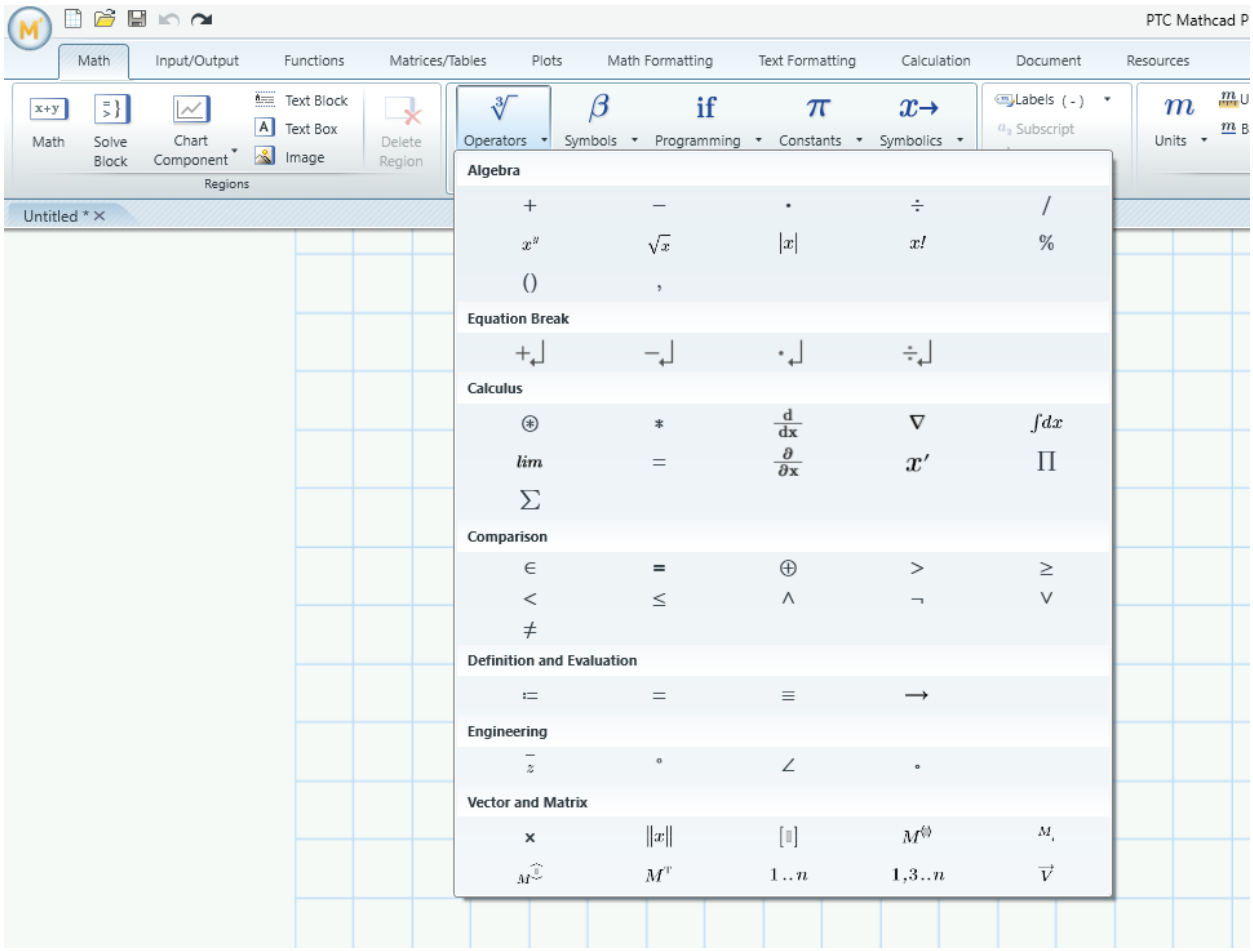


Рис. 1.6. Палітра інструментів розділу *Operators* вкладки *Math*

Розділ *Constants* містить математичні та фізичні константи.

Розділ *Symbolic Operators and Keywords* містить оператор символьного виведення, ключові слова, які визначають функції символьних обчислень, а також модифікатори – спеціальні елементи, які використовуються для зміни поведінки ключових слів.

Розділ *Units* містить повний набір одиниць вимірювання за міжнародною системою *CI*, американською системою одиниць (*USCS*) та системою «сантиметр – грам – секунда».

Математичні формули та вирази у MathCad-документі вводяться в область *Math*, яка встановлюється за замовчуванням. Щоб розпочати роботу, встановіть вказівник мишки у потрібному місці документа і клацніть лівою клавішою. На екрані встановиться маркер уведення у вигляді синього хрестика. З цього місця можна розпочинати набір формул.

За необхідності вирази та функції *Math* можна редагувати та форматувати, використовуючи вкладку *Math Formatting*.

1.4. Обчислення математичних виразів

Константи

У пакеті MathCad використовуються такі типи констант:

- *цілочислові константи* (наприклад, 15, – 40, 0 тощо);
- *дійсні константи*, які можуть записуватися в одній з двох форм: з фіксованою десятковою крапкою (наприклад, 4.235) та в експоненційній формі, записується у вигляді $a \times 10P$, де a – цілочислова константа або дійсна константа з фіксованою крапкою, P – порядок. Крапка означає операцію множення (клавіша *). Для введення порядку натиснути ^ (тильда) – операція піднесення до степеня.

Приклад.

$12.3*10^{-5}$ – десяткова константа з мантисою (12.3) та порядком (-5).

- *комплексні константи*, що записуються у вигляді $z = a + bi$, причому між величиною уявної частини b та уявною одиницею i не ставиться знак множення;
- *рядкові константи* – довільна послідовність символів (у тому числі українські та грецькі літери), записані у лапках (наприклад, «Це рядкова константа»);
- *вмонтовані константи*, які поділяються на математичні та фізичні константи.

За замовчуванням математичні константи e та π при обчисленнях у MathCad враховуються з точністю до 15-ти значущих цифр.

Математичні константи подано у табл. 1.

Таблиця 1

Математичні константи

Константа	**Найменування**	**Значення**
∞	Нескінченність	$1*10^{307}$
e	Основа натурального логарифму	2.71828182845905
π	Pi	3.14159265358979

Змінні

Змінні є іменованими об'єктами, яким надається деяке значення, що може змінюватися під час виконання обчислень. Імена змінних називають ідентифікаторами.

Імена змінних, імена функцій, назви одиниць виміру, імена констант загалом прийнято називати ідентифікаторами.

Ідентифікатори у пакеті MathCad – це набір латинських та грецьких літер, цифр, знаків підкреслення. Великі та малі літери визначають імена різних змінних. Починатися ідентифікатор має лише з літери. Використання в ідентифікаторі символів кирилиці та пробілів заборонено.

Грецький символ вводиться наступним чином: Вкладка *Math*→розділ *Symbols*.

У пакеті MathCad можна використовувати такі змінні: *прості, індексовані* (масиви), *дискретні* (ранжовані), *системні* та *розмірні*.

У пакеті MathCad змінну не потрібно попередньо описувати: її тип визначається автоматично при наданні змінній конкретного значення. Неініціалізовані (неозначене значення) змінні виокремлюються на екрані червоним кольором.

Проста змінна – змінна, значення якої визначається однією величиною.

Приклад ідентифікаторів: *X*, *alfa*, *Alfa*, *beta*.

Нижні індекси в іменах простих змінних (наприклад, a_x, c_x) вводяться таким чином: Вкладка *Math* → група *Style* → кнопка *Subskripts*

Системні змінні (вмонтовані змінні) визначають параметри функціонування MathCad при обчисленнях, а також при роботі з масивами (табл. 1.2).

Таблиця 1.2

Системні змінні

Ім'я	**Значення за замовчуванням**	**Застосування**
ORIGIN	0	Визначає індекс першого елемента масиву. Можна задати їй інше значення у робочому документі, наприклад, ORIGIN:=1 або встановити значення ORIGIN:=1 на вкладці *Calculation*
TOL	0.001	Визначає точність збіжності деяких функцій, таких як інтеграл, похідна, *odesolve* та *root*
CTOL	0.001	Визначає, наскільки точно має виконуватися обмеження в блоці розв'язанняння, щоб розв'язок вважався прийнятним під час використання функцій *find*, *minerr*, *minimize*, *maximize*

Розмірні змінні – числові змінні та функції, які мають розмірність і містять значення у системі вимірювання фізичних величин за міжнародною системою *CI*, американською системою одиниць (*USCS*) та системою «сантиметр-грам-секунда» (СГС). Зроблено це для спрощення інженерних і фізичних розрахунків, які MathCad виконує з перетворенням одиниць виміру фізичних величин.

Вставити одиницю виміру можна з переліку одиниць вимірів. Для цього необхідно увійти до відповідної категорії одиниць вимірювання,

яка знаходиться у розділі *Units* вкладки *Math* і вказівником мишки обрати потрібні одиниці вимірювання.

Одиниці вимірювання можуть бути введені у математичний вираз і з клавіатури, наприклад,

L:=10 m.

Літера *m* розпізнається пакетом MathCad як одиниця виміру метр.

Індексовані змінні відповідають масивам MathCad. Вони будуть детально розглянуті у подальших розділах.

1.4.1. Оператори

Оператори у MathCad є елементами середовища, за допомогою яких можна створювати математичні вирази. До них належать символи арифметичних операцій, символи обчислення сум, добутків, похідних, інтегралів тощо.

Оператор присвоєння. Значення змінної можна ініціалізувати за допомогою оператора присвоювання, який записується за використання такої синтаксичної конструкції:

<Ідентифікатор_змінної> **:=** <Вираз>

Символ **:=** присвоєння можна ввести з клавіатури або використавши вкладку *Math* → розділ *Operators* → категорія *Definition and Evaluation*.

Якщо змінній присвоюється початкове значення за допомогою оператора **″:=″**, тоді таке присвоєння називається локальним. До цього присвоєння змінна не означена і її не можна використовувати. Оператор присвоювання використовується для означення змінних, функцій, одиниць вимірювання або системних змінних.

Крім оператора звичайного присвоювання, у пакеті MathCad є ще оператор глобального присвоювання ″≡″, який вставляється після імені змінної за допомогою клавіші (~) або кнопкою (≡) вкладки *Math* → розділ *Operators* → категорія *Definition and Evaluation*. За призначенням таке присвоєння

еквівалентне звичайному, але послідовність виконання областей (блоків) у документі зі знаком глобального присвоєння є іншою.

Спочатку в MathCad-документі у звичайній послідовності згори донизу і зліва праворуч виконуються всі області з оператором глобального присвоєння, а потім знову спочатку, у цій же послідовності всі інші області. У результаті цього, якщо всередині MathCad-документа є оператор $x \equiv 3$, тоді змінна x буде доступною вже з початку MathCad-документа.

Однак є певні обмеження. Заборонено перевизначати глобальне присвоювання за допомогою локального означення, а також:

- не дозволено мати більше одного глобального означення тієї ж змінної у одному аркуші MathCad-документа або на базовому аркуші, який містить інші аркуші;
- не можна використовувати оператор глобального присвоювання всередині блоку розв'язання.

Оператор виведення. Щоб вивести на екран значення змінної або виразу, достатньо після виразу натиснути клавішу ″=″. На екрані після знаку ″=″ виводиться числове значення змінної або виразу як результат виконання правої частини. Стосовно оператора виведення у літературі можна зустріти термін «оператор оцінювання».

Оператор масштабування $x \cdot y$. Масштабує x на y. Якщо y є функцією, вона застосовується до x. В іншому випадку x та y множаться. x та y – це скалярні величини, масиви, функції або одиниці вимірювання.

Оператор масштабування візуалізується лише тоді, коли завантажений з математичної області розділу *Engineering*.

Коли ви вводите такі вирази, як $5y$, оператор масштабування автоматично вставляється між 5 та y, і він діє як оператор множення. Однак, якщо ви хочете масштабувати x на y, вам потрібно вставити оператор масштабування з математичної області. Без оператора масштабування *Engineering Notebook* обробляє xy як окрему змінну.

Коли результат містить одиниці вимірювання, оператор масштабування вставляється між результатом та одиницею вимірювання. Цей оператор масштабування не можна видалити.

Ми вперше використали термін *Engineering Notebook* (Інженерний Блокнот). У PTC Mathcad Prime – це функціонал, який дозволяє створювати професійно оформлені, інтерактивні документи, які об'єднують математичні рівняння, текст, графіки та дані для детального документування інженерних розрахунків, забезпечуючи їх зрозумілість та легкість спільного використання у команді, особливо для складних задач і проєктів. Це не просто калькулятор, а потужний інструмент для розв'язання складних математичних задач та їх візуалізування у єдиному робочому аркуші (*worksheet*). Таким чином, *Engineering Notebook* у Mathcad Prime – це робочий простір для створення повноцінних інженерних звітів та документації.

1.4.2. Математичні оператори

Математичними операторами в MathCad є: додавання, віднімання, множення, ділення та піднесення до степеня. Оператори можна вводити з клавіатури, або використовуючи:

Вкладка *Math* → розділ *Operators* → категорія *Algebra* (рис. 1.7).

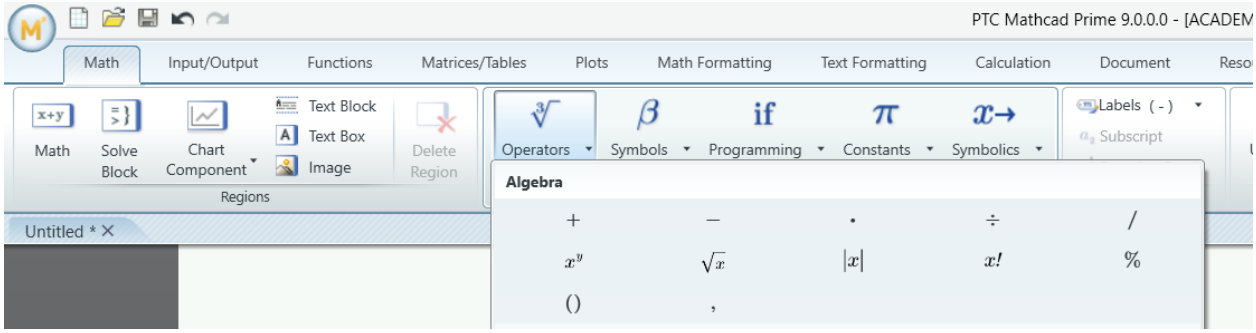


Рис. 1.7. Математичні оператори. Вкладка *Math* → розділ *Operators* → категорія *Algebra*

Особливістю у використанні |x| та |M| є те, що у першому випадку це модуль *x*, а у другому – визначник квадратної матриці *M* (при активуванні з панелі *Matrices/Tables* у MathCad-документі візуалізується подвійними вертикальними рисками).

1.4.3. Введення математичних виразів

Математичні формули та вирази в MathCad-документі вводяться у математичну область. Математична область встановлюється за замовчуванням при завантаженні MathCad-документу. Для активування редактора формул достатньо поставити вказівник миші у довільному вільному місці вікна та клацнути лівою клавішею. Початкова точка введення позначається курсором (синім перехрестям, хрестиком).

Прості логічні вирази – це два арифметичні вирази (числові константи, змінні, функції, вирази), які об'єднані символами логічних відношень. Символи логічних відношень можна вводити з клавіатури або вкладка *Math* → розділ *Operators* → категорія *Comparizon*. Якщо умова, яка подається простим логічним виразом, у результаті аналізу приймає значення «істина», тоді такий вираз набуває значення одиниці, інакше – нуля.

Складені логічні вирази – це прості логічні вирази, які об'єднані символами логічних операцій:

∧ – знак операції логічного множення (*and*);

∨ – знак операції логічного додавання (*or*);

⊕ – знак операції «виключне АБО»;

¬ – знак операції заперечення (*not*).

Вставлення цих символів у логічні вирази виконується через вкладку *Math* → розділ *Operators* → категорія *Comparizon*.

1.4.4. Стандартні (вмонтовані) функції

У пакеті MathCad (*Engineering Notebook*) міститься понад 700 вмонтованих функцій.

Для звернення до функції необхідно задати її позначення (ім'я), а потім аргумент. Якщо аргументів кілька, всі вони вводяться через кому. Весь набір вмонтованих функцій формується за категоріями відповідно до функціонального призначення. Спочатку з клавіатури набирається ім'я функції,

а потім у круглих дужках аргумент (якщо аргументів декілька, тоді вони відділяються комами).

Позначення (ідентифікатор) функції можна ввести з клавіатури або обрати потрібну функцію у вкладці *Functions* відповідної категорії (рис 1.8).

Для цього необхідно активувати піктограму *All Functions* вкладки *Functions*. Пізніше у розкривному списку потрібно вибрати необхідну категорію функцій і підтвердити свій вибір подвійним натисканням лівої клавіші мишки.

Для пошуку необхідної функції ви можете встановити перелік функцій у алфавітному порядку та загальний перелік категорій функцій.

Зазначимо, що усі дії у розкривному списку підтверджуються подвійним натисканням лівої клавіші мишки.

Цей процес проілюстровано на рис. 1.8.

Проте, слід пам’ятати, що при зверненні до вмонтованих функцій за іменем вони чутливі до регістру.

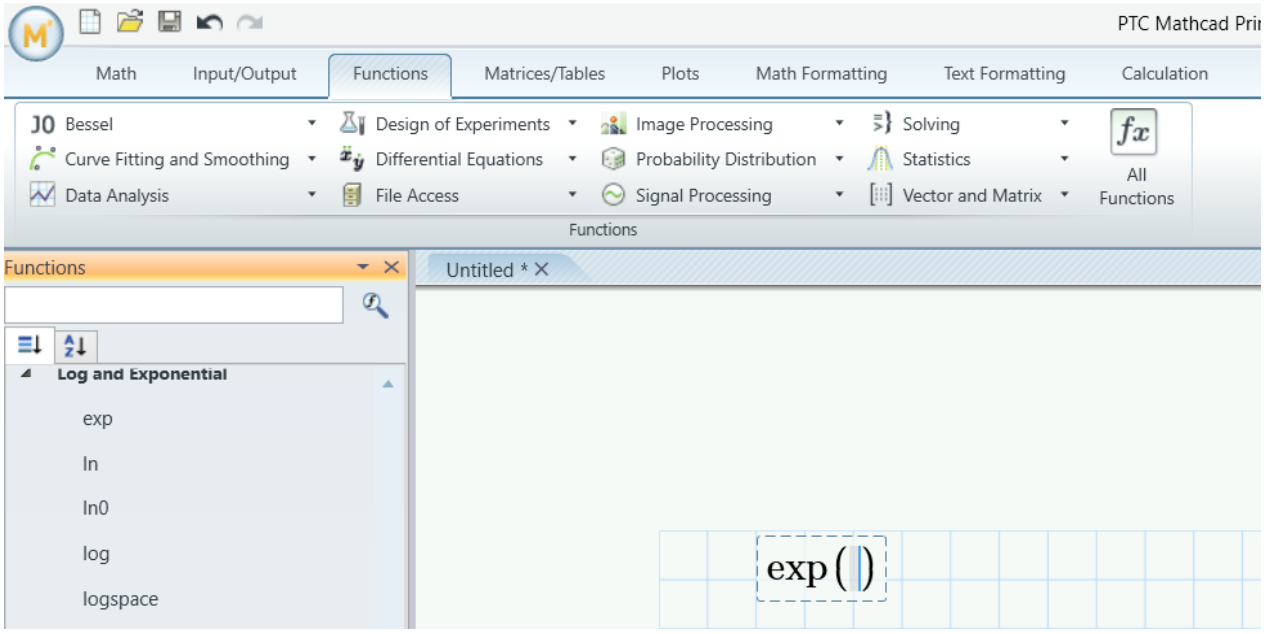


Рис. 1.8. Пошук функції через вкладку *Functions*

Щоб отримати довідку щодо вмонтованої функції:

- введіть ім’я або вставте функцію через вкладку *Functions*;

- активуйте ім’я функції, а не її аргументи, а потім натисніть клавішу *F1* (відкриється відповідний розділ довідки);
- усі функції *Engineering Notebook*, вмонтовані або означені користувачем, обмежені використанням максимум 16 аргументів.

Багато поширених операцій подані відповідними математичними нотаціями, і тому вважаються операторами, описаними окремо.

До слова, математична нотація – це універсальна система символів (цифри, літери, оператори, спеціальні знаки) та правил використання для компактного, точного запису математичних об’єктів, ідей, рівнянь та операцій, що є мовою математики, фізики, інженерії та інших наук, яка дозволяє чітко відтворювати складні концепції.

У *Engineering Notebook* пакету MathCad доступна значна кількість математичних функцій, найуживаніші з яких подані нижче.

Потрібно пам’ятати, що тригонометричні функції у MathCad використовують значення аргументу в радіанах. Для задавання значення аргументу в градусах потрібно аргумент помножити на масштабний множник π/180. Так само для отримання результату в градусах потрібно аргумент помножити на масштабний множник 180/π.

Тригонометричні функції

Назва функції	**Запис**	**Пояснення**
acos	acos(z)	повертає кут (у радіанах), косинус якого є z
acot	acot(z)	повертає кут (у радіанах), котангенс якого є z
acsc	acsc(z)	повертає кут (у радіанах), косеканс якого є z
angle	angle(z,y)	повертає кут (у радіанах) між віссю абсцис і точкою (z, y). Значення z і y повинні бути дійсними
asec	asec(z)	повертає кут (у радіанах), секанс якого є z

asin	asin(z)	повертає кут (у радіанах), синус якого дорівнює z
atan	atan(z)	повертає кут (у радіанах), тангенс якого є z
atan2	atan2(z,y)	повертає кут (у радіанах) між віссю абсцис і точкою (z, y). Значення z і y повинні бути дійсними
cos	cos(z)	повертає косинус z
cot	cot(z)	повертає котангенс z
csc	csc(z)	повертає косеканс z
sec	sec(z)	повертає секанс z
sin	sin(z)	повертає синус z
sinc	sinc(z)	повертає значення sin(z)/z
tan	tan(z)	повертає тангенс z

Експонента та логарифмічні функції

Назва функції	**Запис**	**Пояснення**
exp	e^x або exp(x)	повертає число e, піднесене до степеня x
ln	ln(x)	повертає натуральний логарифм (з основою e) від числа x
log	log(x, a) або log(x)	повертає логарифм з основою a від числа x, а якщо a немає, тоді a = 10

Спеціальні функції:

floor(x) – обчислює найбільше ціле, яке не перевищує число *x*;

round(x) – заокруглює число *x* до цілого;

round(x, n) – заокруглює число *x* до *n*-го розряду у дробовій частині, якщо $n > 0$ або до *n*-го розряду у цілій частині числа, якщо $n < 0$;

trunc(x) – повертає цілу частину дійсного числа *x*.

mod(x, n) – повертає залишок цілочислового ділення цілих чисел *x* та *n*.

1.4.5. Математичні вирази

Для формування математичних виразів та обчислення їх значень за різних даних використовуються: константи, змінні, оператори, символьні операції, вмонтовані функції MathCad Prime, а також функції користувача.

Тепер обчислимо вираз з попереднього прикладу у пакеті MathCad.

Спочатку ініціалізуємо змінну *x*, а потім використовуємо її як аргумент функції.

Пам'ятаємо, *усі обчислення виконуються зверху вниз і зліва праворуч.*

Функції (поряд з операторами) можуть входити у математичні вирази, наприклад (рис. 1.9):

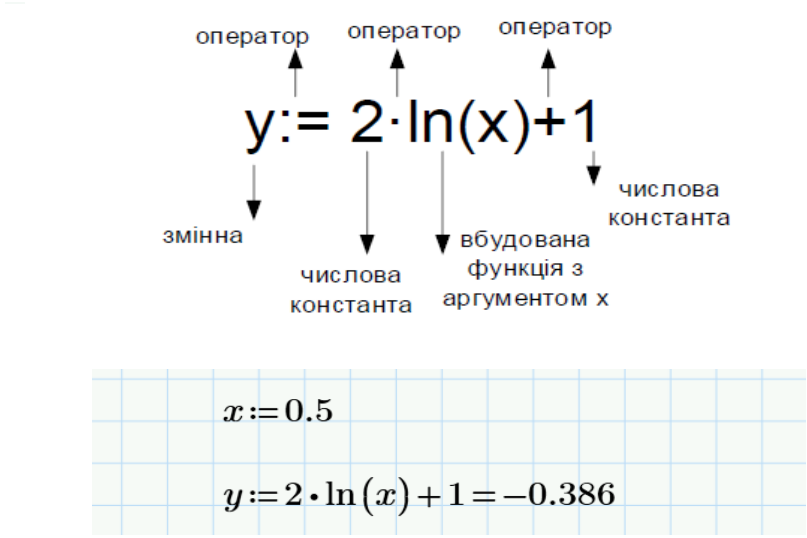


Рис. 1.9. Приклад математичного виразу

Функція умовного розгалуження if

У *Mathcad Prime* є функція умовного розгалуження. Функція *if* реалізовує логічну структуру «якщо – тоді».

Для формування функції умовного розгалуження *if* використовується така синтаксична конструкція:

if(L, vT, vF)

де: *L* – логічний вираз (умова);

vT – вираз, який обчислюється за умови, що *L*, у результаті аналізу, приймає значення *true*;

vF – вираз, який обчислюється за умови, що *L*, у результаті аналізу, приймає значення *false*.

Важливо! Функцію *if* потрібно набирати за допомогою клавіатури.

Напрклад, обчислити значення функції (рис. 1.10):

$$y = \begin{cases} x^2 + 1, & \text{якщо x} > 0 \\ x - 8, & \text{якщо } x \le 0 \end{cases}$$

$u(r) := \text{if}(r > 0, r^2 + 1, r - 8)$

$r := 3 \qquad u(r) = 10$

$r1 := -0.8 \qquad u(r1) = -8.8$

Рис. 1.10. Обчислення значення функції з використанням функції умовного розгалуження *if*

Індивідуальні завдання.

Обчислити значення виразу з використанням вмонтованих функцій пакету MathCad.

Варіант 1

Обчислити значення функції $y = \dfrac{e^{\sin\mu} + \sqrt[4]{a+\mu}}{\ln^3 b\mu}$; μ=10.6; a =17.3; b=0.36.

Варіант 2

Обчислити значення функції $y = \dfrac{\ln^4 b\xi + 0{,}85}{\sqrt[3]{a + b\xi^3}}$; ξ=0.4;. a = 46; b=1.85.

Варіант 3

Обчислити значення функції $y = \dfrac{\sqrt[4]{1 + \sqrt{a\omega + b}}}{\sin^2 b\omega + \omega}$; ω=4.3; a =1.35; b=8.4.

Варіант 4

Обчислити значення функції $y = \dfrac{tg^2(\gamma - a) + \sqrt{\ln\gamma}}{e^{-b\gamma}}$; γ=1.3; a =1.8; b=0.56.

Варіант 5

Обчислити значення функції $y = \dfrac{\sqrt[3]{a^3 + \lambda^3}}{tg^3 b\lambda + 1{,}6}$; λ=0.15; a =1.25; b=0.86.

Варіант 6

Обчислити значення функції $y = \dfrac{\sqrt[3]{e^{a\eta}} + b}{0{,}25\ln^2 a\eta}$; η=10.5; a = 0.3; b=9.5.

Варіант 7

Обчислити значення функції $y = \dfrac{\ln^2(a^3 + \chi^3)}{\sqrt{a^3 + \chi^3} + \sqrt[3]{b}}$; χ=8.2; a=43; b=205.

Варіант 8

Обчислити значення функції $y = \dfrac{1 + \cos^2(a^3 + \nu^3)}{\nu^2 + \sqrt[3]{tgb\nu}}$; ν=0.5; a =0.84; b=0.63.

Варіант 9

Обчислити значення функції $y = \dfrac{a^{\mu} + e^{-b\mu}}{\sin^2 b\mu + 1{,}24}$; μ =0.3; a =0.5; b=0.16.

Варіант 10

Обчислити значення функції $y = \sqrt[3]{\dfrac{|b\kappa|}{arctg\dfrac{b^2}{a^2 + \kappa^2}}}$; κ=-10; a = 2.8; b=1.5.

Варіант 11

Обчислити значення функції $y = \dfrac{(\upsilon^2 + 1)^{-\frac{1}{\sin b\upsilon}}}{\sqrt[3]{\dfrac{\upsilon}{a} - 0{,}39}}$; υ =0,2; a =0.36; b=0.74.

Варіант 12

Обчислити значення функції $y = \dfrac{e^{\nu^2 + 1}}{\sqrt[5]{\nu - a} + \ln^2 b\nu}$; ν =1.2; a =4.6; b=6.8.

Варіант 13

Обчислити значення функції $y = \dfrac{e^{\sin^2 a\eta} + arctgb\eta}{\sqrt[3]{(\eta + b)^2}}$; η =1.5; a =0.45; b=8.8.

Варіант 14

Обчислити значення функції $y = \dfrac{\sin^2 a\rho + \sqrt[3]{|\rho - b|}}{|\rho - b|^3}$; ρ=16; a =0.28; b=19.3.

Варіант 15

Обчислити значення функції $y = \dfrac{\omega^{\frac{a}{b}} - \sqrt[3]{\dfrac{\omega + b}{a}}}{1{,}1 + \cos^2 a\omega}$; ω =6.8; a =3.5; b=6.4.

Варіант 16

Обчислити значення функції $y = \sqrt[5]{\dfrac{\lambda^2}{4a^2 + 0{,}6} \ln^2 (b + \lambda)}$; λ =10; a=2.4; b=16.8.

Варіант 17

Обчислити значення функції $y = \dfrac{\cos^2 a\mu + e^{-a\mu}}{arctg(\sqrt{b} + \sqrt{\mu})}$; μ =0.6; a =1.3; b=0,75.

Варіант 18

Обчислити значення функції $y = \dfrac{a - e^{b\sigma}}{(\sigma - a)^2 + \ln^2(\sigma - a)}$; σ=2.1; a=4.38; b=0.24.

Варіант 19

Обчислити значення функції $y = \dfrac{\sqrt[5]{a + \delta} - b}{1{,}6\cos^{5,2} a\delta}$; δ=0.2; a =1.85; b=2.64.

Варіант 20

1. Обчислити значення функції $y = \dfrac{e^{a+\mu}}{\sqrt[4]{b\mu} + \ln^2 a\mu}$; μ =0.8; a =1.38; b=9.6.

Варіант 21

Обчислити значення функції $y = \dfrac{\sqrt[4]{a + \xi} + (\dfrac{\xi}{b})^{1,3}}{0{,}8tg^3 b\xi}$; ξ =8.5; a =17.6; b=0.14.

Варіант 22

Обчислити значення функції $y = \dfrac{\cos^2|\mu - a|}{\sqrt[3]{|b\mu|} + e^{\mu - a}}$; μ =-5; a =1.5; b=14.8.

Варіант 23

Обчислити значення функції $y = \dfrac{1+\cos^2(a+b\eta)}{0{,}5\sqrt[3]{a+b\eta}}$; η =2.5; a =33.6; b=6.46.

Варіант 24

Обчислити значення функції $y = \dfrac{e^{\sin(\lambda+b)}+a^3}{\ln^2(\lambda+b)}$; λ =0.25; a =0.86; b=2.8.

Варіант 25

Обчислити значення функції $y = \dfrac{\sqrt[3]{a+e^{b\chi}}}{a+\dfrac{\ln^2\chi}{b\chi}}$; χ =6.8; a =2.3; b=0.28.

Варіант 26

Обчислити значення функції $y = \dfrac{0{,}25+\cos^2 a\xi}{\sqrt[4]{b^3}+\xi^3}$; ξ =1.3; a =43; b=2.46.

Варіант 27

Обчислити значення функції $y = \dfrac{arctg\left(a^2+\omega^2\right)}{\ln^3 b\omega+\sqrt{b}}$; ω =1.1; a =1.38; b=2.46.

Варіант 28

1. Обчислити значення функції $y = \dfrac{e^{-b\rho}+a^2}{\sqrt[3]{\left(\dfrac{a}{\rho}\right)^3+\sqrt{b}}}$; ρ =0.1; a =0.15; b=2.3.

Варіант 29

Обчислити значення функції $y = \dfrac{\sqrt{\ln a+\sqrt[3]{b^2+13x}}}{\sin^2 ax+0{,}84}$; x=0.05; a =4.2; b=1.54.

Варіант 30

Обчислити значення функції $y = \dfrac{\ln^3(a+x)-10{,}5}{\sqrt[4]{0{,}6b^2+\left(\dfrac{x}{a}\right)^2}}$; x=58; a =3.2; b=8.4.

1.5. Ранжовані змінні. Функції користувача

Ранжовані змінні. *Ранжована змінна* – змінна, якій наданий діапазон зміни значень, що дозволяє здійснювати можливість проведення циклічних обчислень. Таку змінну можна вважати аналогом змінної параметричного циклу з постійним кроком в алгоритмічній мові.

Синтаксична конструкція для оголошення ранжованої (діапазонної) змінної має такий вигляд:

```
Var : = Var1, Var1 + h .. Var2,
```

де

`Var` – ім'я діапазонної змінної;

`Var1` – початкове значення діапазонної змінної; `h` – крок зміни діапазонної змінної; якщо крок не заданий, тоді його приймають таким, що дорівнює 1; якщо ж крок `h` від'ємний, необхідно забезпечити `Var1> Var2`;

`Var1,Var1 + h` – перше та друге значення діапазонної змінної відповідно;

`Var2` – кінцеве значення діапазонної змінної.

Друге значення дорівнює сумі першого значення та кроку зміни значення змінної. Якщо крок змінної дорівнює одиниці, його можна не задавати, наприклад:

x змінюється від 1 до 10 з кроком 0.1; x:=1,0.1..10

x змінюється від 1 до 10 з кроком 1; x:=1,2..10

x змінюється від 1 до 0 з кроком –0.1. x:=1,0.9..0

При визначенні діапазону всі числа мають бути дійсними. Вони можуть бути від'ємними, десятковими або виразами, які обчислюються як дійсні числа. У MathCad-документі варіанти формування ранжованих змінних будуть виглядати як на рис. 1.11. Увага! Останнє значення ранжованої змінної *y* дорівнює 3, а не π. Цей випадок може призвести до неочікуваних результатів під час побудови графіків періодичних функцій.

Особливістю діапазонної змінної є те, що у разі звертання до неї одразу використовуються всі її рівновіддалені значення. Немає змоги використати

лише одне або декілька її значень, як це можливо для доступу до значень окремих елементів векторів або матриць, вказавши їх індекс (індекси). У вказаному сенсі діапазонна змінна посідає проміжне місце між скалярними та векторними величинами.

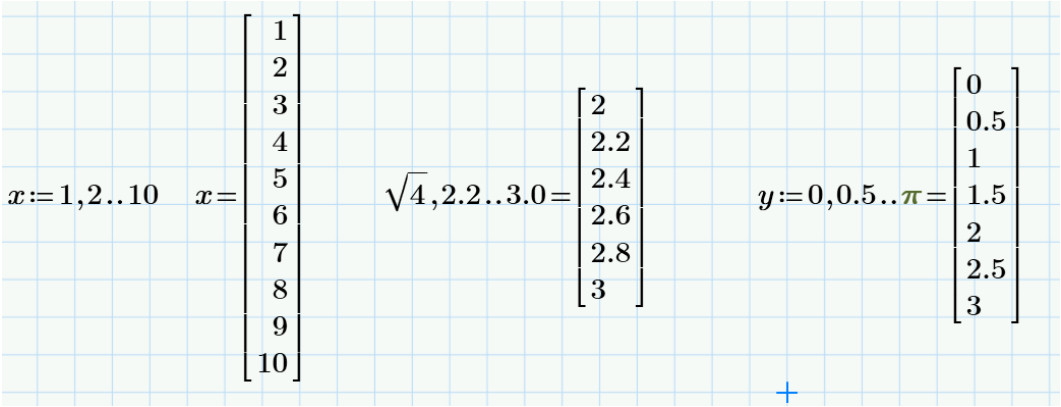


Рис. 1.11. Результат формування ранжованих змінних.

Тому, якщо у виразі використано ім’я діапазонної змінної, тоді одразу обчислюються і на екран виводяться всі значення, які відповідають кожному значенню діапазонної змінної.

Завдяки цій властивості діапазонні змінні зручно використовувати для задавання індексів векторів і матриць, для обчислення значень виразів, функцій, зокрема, для табулювання функцій під час побудови їх графіків або виведення їх значень у стовпець після символу "=", який задається після виразу функції, тощо.

Функції користувача. Незважаючи на велику колекцію вмонтованих функцій, часто виникає необхідність доповнити обчислення розробленими користувачем функціями. Такі функції називатимемо *функціями користувача*.

Функції користувача можуть бути значно складнішими, містити інші функції або елементи програмування

Використання функції користувача передбачає два моменти: оголошення функції та звернення до неї.

Ліворуч від оператора присвоєння – ім’я функції, а далі – перелік її аргументів у дужках (*у програмуванні вони називаються формальними аргументами/параметрами, оскільки використовуються лише для описування*

алгоритму обчислювання значення функції), а праворуч – вираз для її обчислення з використанням цих формальних аргументів.

Для формування функції користувача використовується така синтаксична конструкція:

<ім'я_функції> (<перелік_аргументів>) <вираз>

де

ім'я_функції – послідовність символів, яка узгоджується з тими ж правилам, як і правила формування ідентифікаторів.

перелік_аргументів – перелік використовуваних у виразі змінних, записаних через кому.

вираз – аналітична залежність, яка описує алгоритм обчислення значення функції з аргументами.

Змінні, які входять до переліку аргументів, є формальними параметрами, і їх значення не задаються до оголошення функції.

Наприклад:

F(x):= 1 + 5.2*sin(π + x) – функція користувача однієї змінної;

My_Function(x, y, t):= $\sin(2\cdot x)^2$ – cos(3·y)*z + tan(t) – функція користувача трьох змінних.

Під час виклику (звертання) до функції на місці формальних аргументів записуються реальні (фактичні) їхні значення, тобто означені змінні, константи, вирази. Функціонування механізму формальних/фактичних параметрів вам добре відомий з курсу «Алгоритмізація та програмування».

1.6. Побудова графіків

В інженерній та науковій практиці відтворення інформації за допомогою графіків має велике значення: інформація у графічному вигляді здебільшого є наочнішою (якісна характеристика інформації) та зрозумілішою від цифрової (кількісна характеристика), поданої у вигляді таблиць.

Побудову графіків у MathCad розглянемо на прикладах найчастіше вживаних графіків у декартових координатах, полярних і тривимірних (поверхневих) графіків.

За кількістю аргументів у функції графіки можна умовно розбити на дві групи: графіки функцій однієї змінної (або двовимірні графіки 2D) і графіки функцій двох змінних (або тривимірні графіки 3D).

У пакеті MathCad Prime вмонтовано кілька різних типів графіків, серед них найпоширеніші це *XY Plot* та *3D Plot.*

XY Plot. Служить для побудови графіка функції $y = f(x)$.

3D Plot. Служить для крапкового подання елементів матриці або візуалізування значень функції у заданих точках (x_i, y_i).

Вибір типу графіка здійснюється у розділі *Insert Plot* вкладки *Plots*. (рис. 1.12).

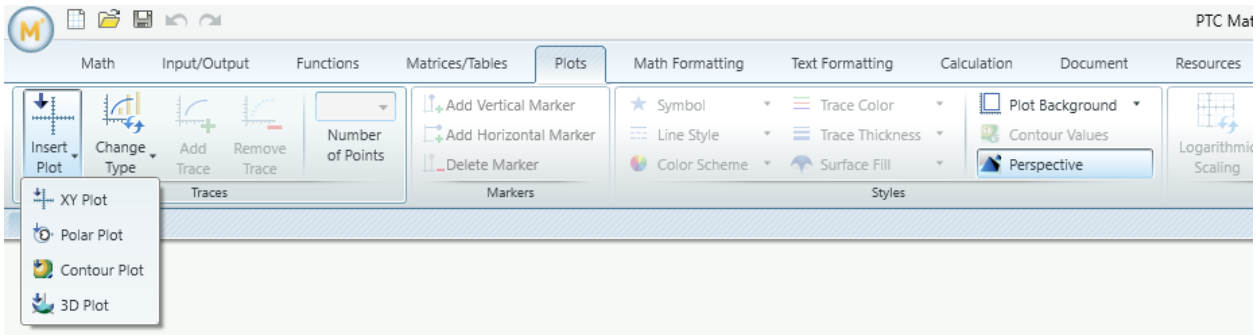


Рис. 1.12. Вибір типу графіка

Для побудови графіків у пакеті MathCad існує дві можливості:

1. Використання функцій.
2. Використання масивів (векторів і матриць).

Двовимірні графіки. Побудова графіка функції однієї змінної включає наступні етапи.

1. Для прикладу оберемо функцію *f(x) : = x*sin(x)*. Визначимо діапазон зміни значень аргументу, оголосивши змінну *x* ранжованою *x := -10,-9.9..10*.

2. Виберемо шаблон графіка *XY Plot* у розділі *Insert Plot* → *Plots* (рис. 1.12).

3. У полі функції (позиція 4) введіть функцію або її аналітичний вигляд. Якщо функція має розмірність (метри, кілограми тощо), тоді у поле введення одиниці виміру осі Y (позиція 3) вставте одиницю виміру.

1 – область побудови графіка;

2 – легенда;

3 – поле введення одиниці виміру осі Y;

4 – поле функції;

5 – поле аргументу функції.

4. Перетягніть легенду осі Y у ліву сторону.

5. У поле аргументу функції (позиція 5) введіть змінну функції. Якщо необхідно, вставте одиницю виміру.

6. Клацніть мишкою осторонь шаблону. Появиться графік функції (рис. 1.13).

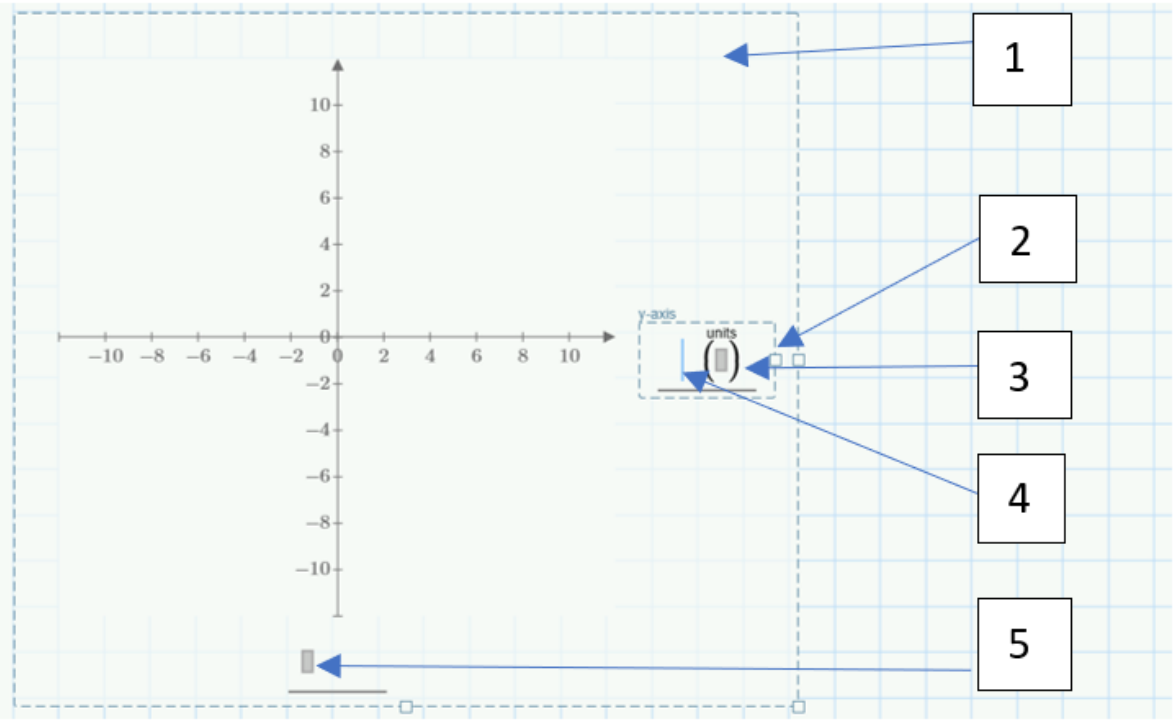


Рис. 1.13. Поля шаблону графіка функції однієї змінної

Зміна діапазонів осями X та Y дозволяє змінювати масштаб, щоб подати графік у зручному для користувача вигляді.

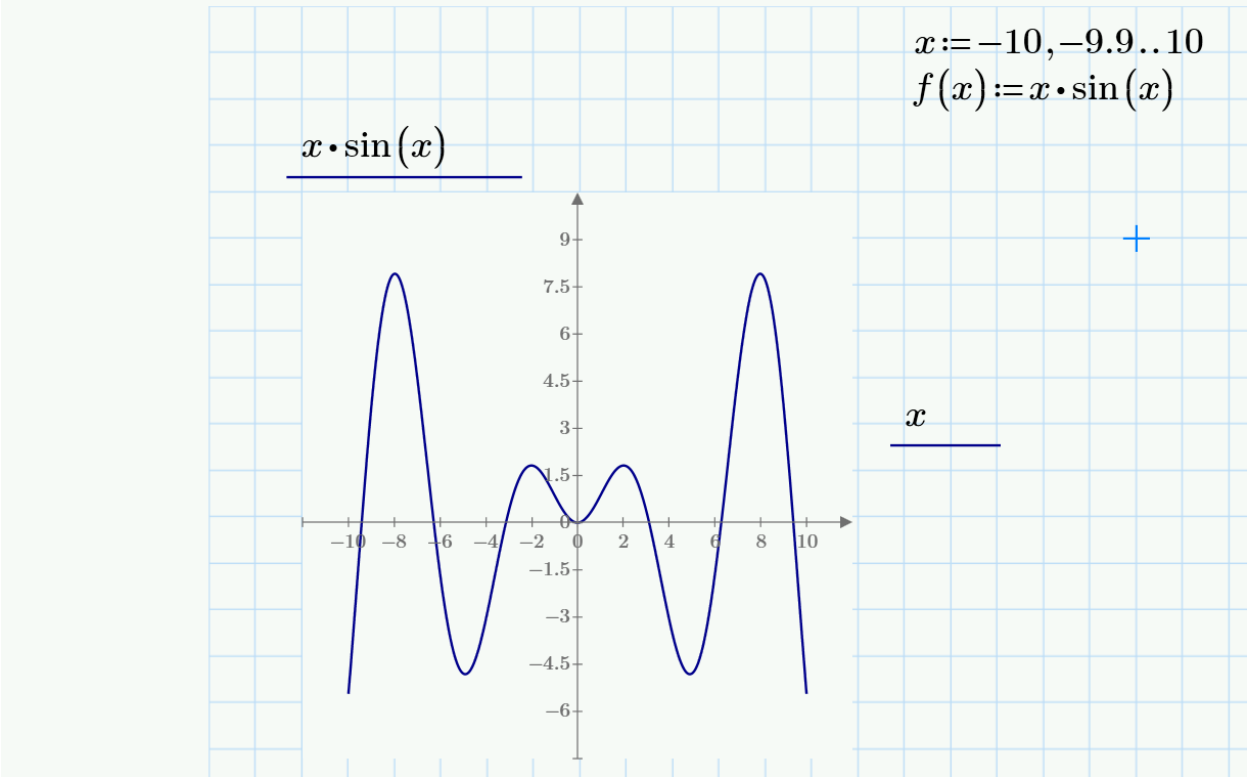


Рис. 1.13. Ілюстрація побудови графіка в прямокутних координатах

Форматування 2D-графіків

Оформлення графіка необхідним чином може бути виконано за допомогою інструментів розділу *Styles* вкладки *Plots*. Вікно *Styles* з цими інструментами показано на рис. 1.14.

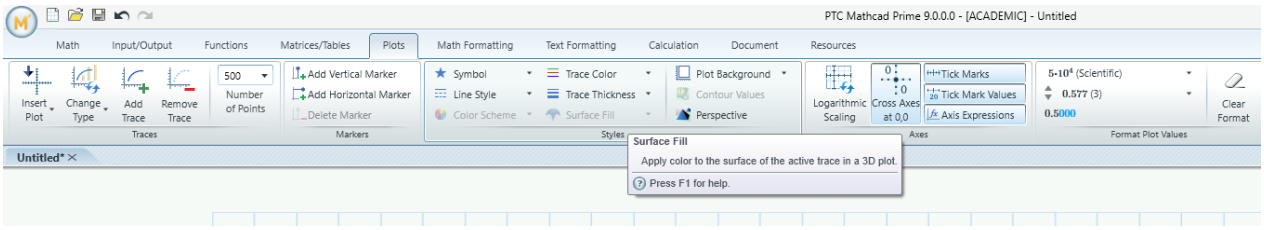


Рис. 1.14. Інструменти розділу *Styles* вкладки *Plots*

Вузлові точки (точки, для котрих обчислюються координати) графіків часто потрібно відзначити якимось символом. Перелік *Symbol* дозволяє вибрати позначки для вузлових точок графіка кожної з функцій.

Перелік *Line Style* дає змогу вибрати типи ліній: безперервна, пунктирна, штрихпунктирна та ін.

Список *Trace Color* дозволяє вибрати колір лінії або символу на графіку.

Список *Trace Thickness* дає змогу вибрати товщину лінії або символу графіка.

Список *Color Scheme* дозволяє встановити зображення контурного графіка у вигляді кольорових контурів.

Змінимо діапазон осі X з –10 та 10 на –5 та 5, відповідно. Для цього замініть на осі X крайні значення з –10 та 10 на –5 та 5. Клацніть мишкою осторонь графіка. Далі змініть колір, тип та товщину ліній шляхом використання інструментів розділу *Style* вкладки *Plots*.

Якщо значення аргументу не вказати, тоді MathCad створює графік функції у межах значень аргументу, за замовчуванням від –10 до 10.

У пакеті MathCad Prime у одній графічній області можна відтворити графіки кількох функцій. Якщо у всіх функцій однакові значення аргументу, достатньо ввести ці значення тільки один раз .

Для розміщення кількох графіків у одній графічній області необхідно виконати такі дії:

- помістити курсор поруч із іменем функції осі Y;
- клацнути *Add Trace* на вкладці *Plots*;
- з’явиться нове поле функції осі Y під поточним полем;
- у нове поле функції осі Y ввести додане ім’я функції;
- помістити курсор праворуч від імені змінної осі X;
- клацнути *Add Trace* на вкладці *Plots*;
- з’явиться нове поле аргументу функції осі X під поточним полем;
- у полі аргументу функції осі X ввести ім’я аргументу функції,

яка долучається до існуючого рисунку.

Для кращого розпізнавання графіків, побудованих в одній графічній області, в MathCad є змога подавати їх лініями різних кольорів, товщини та типів, а їхні точки можна зобразити різними символами з різною товщиною та кольором контурів тощо. На рис. 1.15 подано результат виведення графіків двох функцій у одній графічній області по виконанні поданих вище дій.

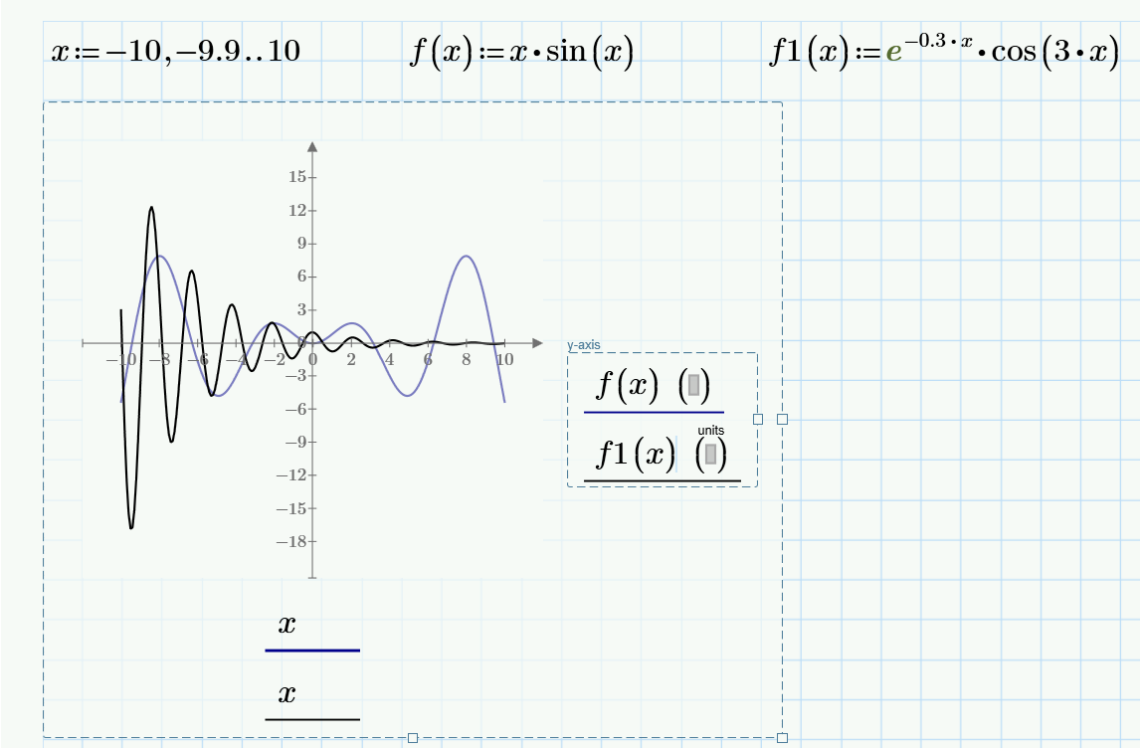


Рис. 1.15. Виведення графіків двох функцій у одній графічній області

Графіки у полярних координатах

Побудову графіка у полярних координатах розпочинають подібно до описаного вище графіка у декартовій системі координат: для цього потрібно вибрати шаблон графіка *Polar Plot* у вкладці *Plots* розділу *Insert Plot*.

За замовчуванням аргумент функції – полярний кут, який змінюється у межах від 0 до 360° (або від 0 до 2π – у радіанах) і необхідність у зміні цього діапазону виникає вкрай рідко.

Відмінності між полярними і декартовими графіками можна легко опанувати на конкретних прикладах побудови графіків у полярних координатах, а за необхідності – скористатися засобами допомоги MathCad.

Приклад графіка у полярних координатах ілюструється MathCad-документом, поданим на рис. 1.16.

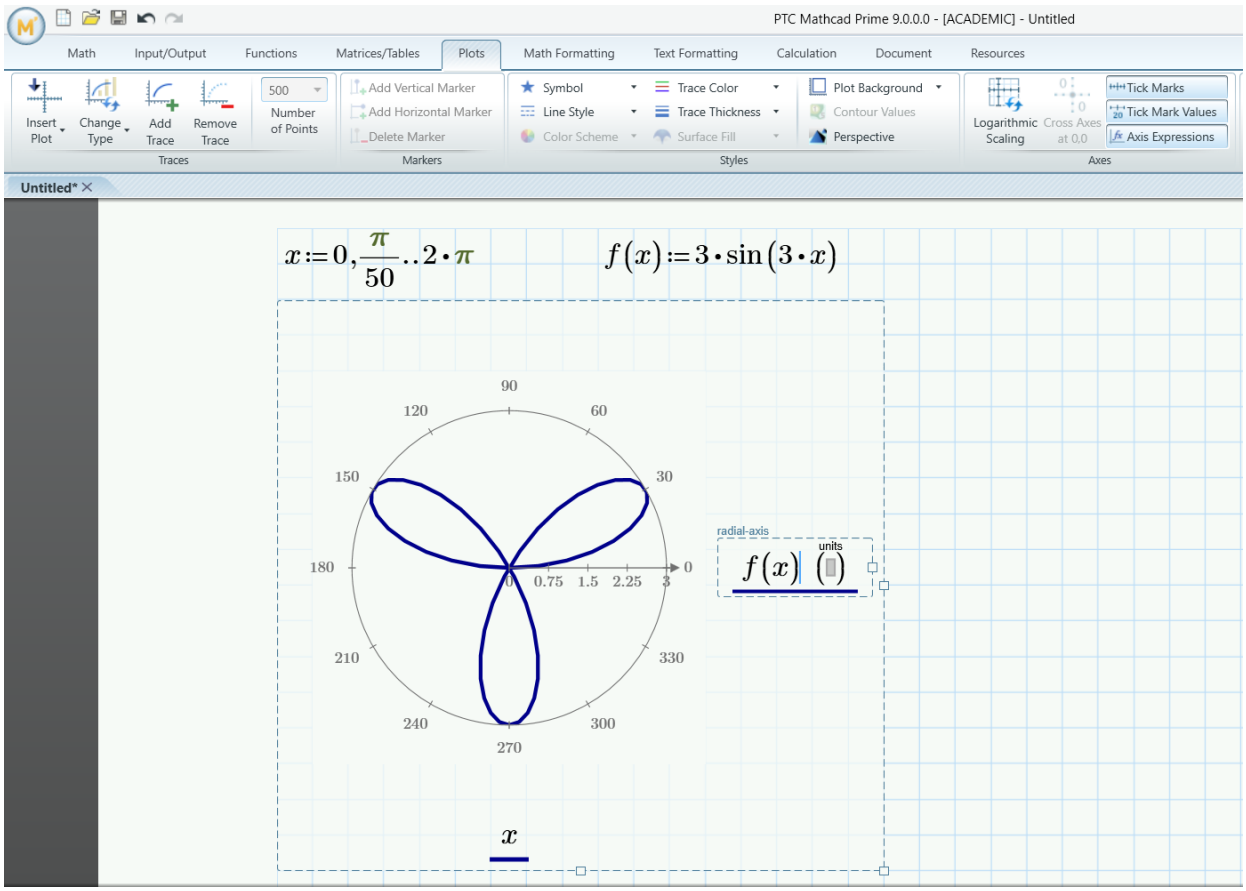


Рис. 1.16. Приклад графіка у полярних координатах

Тривимірні графіки

Тривимірні графіки використовуються для візуалізування функції двох змінних – функції, яка приймає значення у вигляді векторів, або набору 3D-даних. Початкові дані для 3D-графіків можуть мати такі типи даних:

- функція, яка набуває значення у вигляді векторів для одного або двох параметрів із трьома елементами, що відповідають координатам *x*, *y* та *z;*
- матриця з трьома стовпцями, які містять координати *x*, *y* та *z;*
- вектор з трьох векторів, що відповідають координатам *x*, *y* та *z* і є початковими даними функції *CreateSpace*;
- вектор із трьох вкладених матриць, яким відповідають координати *x*, *y* та *z* і є початковими даними функції *CreateMesh*.

Для створення графіка функції трьох змінних (3D-графіка) необхідно активувати піктограму *3D Plot* розділу *Insert Plot* вкладки *Plots*. У документі появиться шаблон графіка з трьома осями та порожнім полем. У це поле вводиться або ім'я масиву, або ім'я двох змінних.

У першому випадку необхідно попередньо сформувати матрицю із значень функції у вузлах прямокутної сітки. Цей спосіб дозволяє відтворити елементи матриці, які містять результати обчислень.

У другому випадку попередньо треба описати функцію від двох незалежних змінних.

Наприклад, побудувати тривимірний графік функції (рис. 1.17)

$S(x, y) = \sin(x^2 + y^2)$,

де x та y змінюються у інтервалі від -1.5 до 1.5.

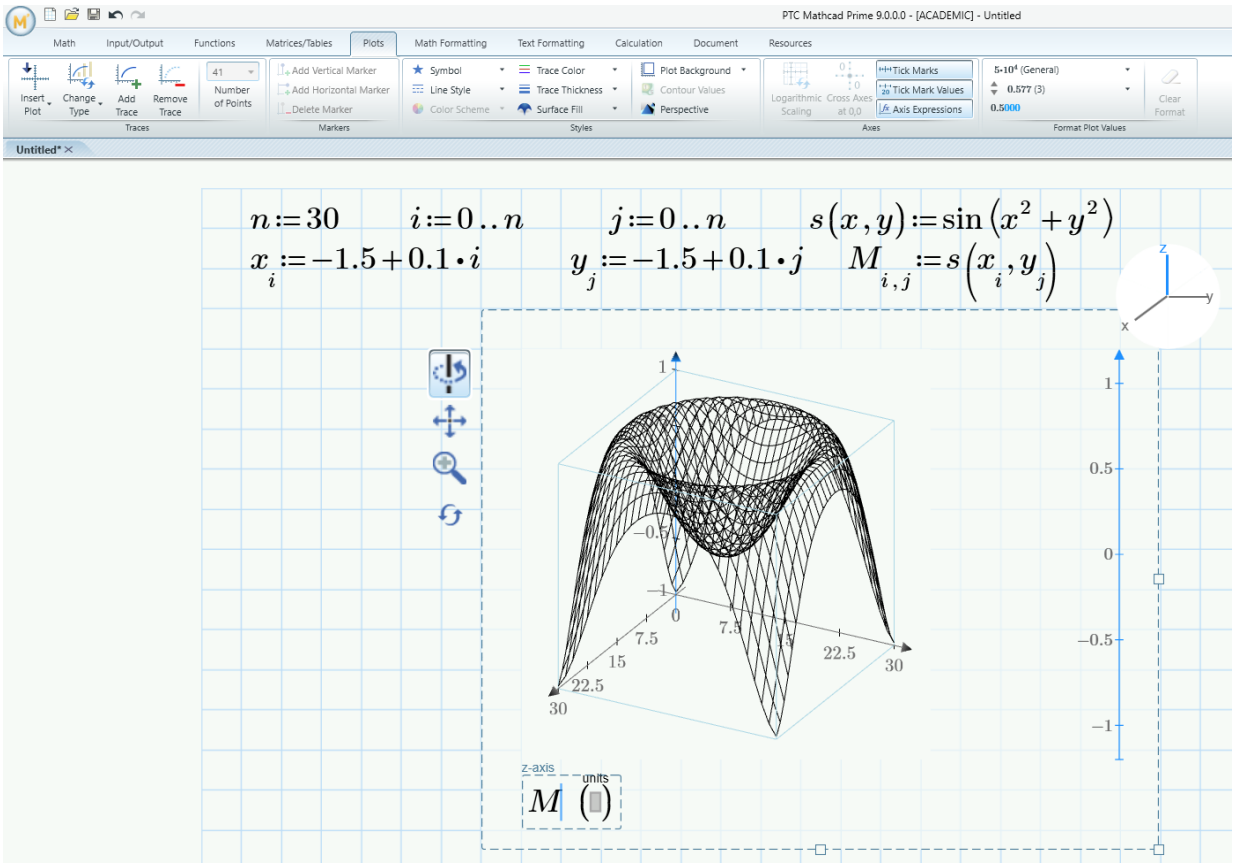


Рис. 1.17. Результати побудови тривимірного графіка

Індивідуальні завдання.

Побудувати графік функції двох незалежних змінних. Початкові дані подано у табл. 1.х.

№ з/п	Функція	Початкові дані
1.	$w = \sqrt{e^{xy} + 1}$	$-2 \le y \le 1, \quad \Delta y = 0.1$ $2.1 \le x \le 4.6, \quad \Delta x = 0.2$
2.	$w = \sqrt[3]{x^2 + 3y}$	$-2 \le y \le 10, \quad \Delta y = 1$ $2 \le x \le 6, \quad \Delta x = 0.2$
3.	$w = cos^2 \frac{x+3}{y+1}$	$2 \le y \le 10, \quad \Delta y = 1$ $2.1 \le x \le 4.6, \quad \Delta x = 0.2$
4.	$w = ln(cos(xy - 1) + 2)$	$0.5 \le y \le 1.5, \quad \Delta y = 0.2$ $1 \le x \le 4, \quad \Delta x = 0.3$
5.	$w = tg^2(x + y) + x^3$	$0.5 \le y \le 1.5, \quad \Delta y = 0.2$ $1 \le x \le 4, \quad \Delta x = 0.3$
6.	$w = 1.57 \cdot 10^3 \cdot x + \sqrt{y}$	$0.5 \le y \le 1.5, \quad \Delta y = 0.2$ $1 \le x \le 4, \quad \Delta x = 0.3$
7.	$w = arctg(3xy) + e^x$	$-2 \le y \le 2, \quad \Delta y = 0.4$ $0.2 \le x \le 2, \quad \Delta x = 0.2$
8.	$w = 5^x + \sqrt[3]{y^2 + 7}$	$-2 \le y \le 2, \quad \Delta y = 0.4$ $0.2 \le x \le 2, \quad \Delta x = 0.2$

9.	$w = \sqrt[5]{cos^2 3y + 2x^2}$	$-4 \le y \le 4, \quad \Delta y = 0.2$ $0 \le x \le 6.5, \quad \Delta x = 0.1$
10.	$w = log_3 \lvert x^2 - 6.5 - y \rvert$	$-4 \le y \le 4, \quad \Delta y = 0.2$ $0 \le x \le 6.5, \quad \Delta x = 0.1$
11.	$w = sin(x + y) - 0.42y$	$-4 \le y \le 4, \quad \Delta y = 0.2$ $0 \le x \le 6.5, \quad \Delta x = 0.1$
12.	$w = e^{cos(x)} + e^{sin(y)}$	$-1 \le y \le 1, \quad \Delta y = 0.1$ $-0.8 \le x \le 1, \quad \Delta x = 0.1$
13.	$w = arctg(xy)$	$-2 \le y \le 1.5, \quad \Delta y = 0.1$ $-0.8 \le x \le 0, \quad \Delta x = 0.1$

РОЗДІЛ 2

ЧИСЕЛЬНІ МЕТОДИ РОЗВ'ЯЗУВАННЯ НЕЛІНІЙНИХ РІВНЯНЬ

Задачі з обчислення розв'язків алгебричних рівнянь, нелінійних рівнянь, систем лінійних та нелінійних рівнянь і нерівностей застосовуються під час розв'язування багатьох прикладних задач із фізики, механіки, електрики тощо.

Для цього у пакеті MathCad передбачено декілька спеціалізованих функцій та блоків, які реалізовують як точні, так і наближені (числові) методи обчислення розв'язку алгебричних рівнянь. Деякі з цих засобів призначені для розв'язування певного типу рівнянь, тобто є вузькоспеціалізованими, а інші – універсальними.

Розв'язати рівняння $f(x) = 0$ означає обчислити множину всіх його коренів (тобто таких значень $x \in [a; b]$, за яких воно стає числовою тотожністю) або ж довести, що їх не існує. Розв'язок (корінь) рівняння $f(x) = 0$ називають ще нулем функції $f(x)$.

У числових методах рівняння вважається розв'язаним, якщо всі корені обчислені з заданою точністю, тобто з наперед заданою похибкою. Відтинком ізоляції кореня рівняння називають такий відтинок множини дійсних чисел, у якому рівняння має один і тільки один розв'язок.

Обчислення коренів числовими методами включає два етапи:

1. *Відділення кореня* – обчислення досить малого інтервалу, всередині якого знаходиться один і лише один корінь рівняння.

Враховуючи легкість побудови графіків функцій у пакеті MathCad, будемо використовувати графічний метод відділення коренів.

2. *Уточнення кореня до заданої точності* – обчислення кожного кореня із заданою точністю усередині окресленого інтервалу.

Для уточнення кореня використовуються спеціальні обчислювальні методи, такі як метод половинного поділу, метод хорд, метод дотичних (метод Ньютона).

У пакеті MathCad Prime для уточнення *кореня нелінійного рівняння* використовується вмонтована функція *root*, яку можна використовувати у двох форматах:

- *root*(*f*(*x*), *x*) – обчислює із заданою точністю значення змінної *x*, при якому вираз *f*(*x*) дорівнює нулю, функція реалізує обчислення ітераційним методом і перед її застосуванням необхідно задати початкове значення змінної *x*, яка належить інтервалу ізоляції кореня.
- *root*(*f*(*x*), *x*, *a*, *b*) – обчислює із заданою точністю значення змінної *x*, при якому вираз *f*(*x*) дорівнює нулю, *a* та *b* – межі інтервалу ізоляції кореня. Зрозуміло, що за такої форми запису функції немає необхідності задавати початкове значення *x*, оскільки воно означено інтервалом [*a*, *b*].

2.1. Обчислення ізольованого кореня нелінійного рівняння

Для обчислення ізольованого кореня (розв'язку) *f(x) = 0* з одним невідомим використовується внутрішня функція пакету MathCad *root(f(x), x)*. Аргументи цієї функції та результат її виконання повинні бути скалярними величинами. Ця функція реалізовує модифікований числовий метод січних або метод Мюллера (*Mueller*), тому обчислює перший найближчий корінь рівняння і для своєї роботи вимагає початкового (нульового) наближення кореня.

Для обчислення решти коренів (якщо вони існують) потрібно змінити початкове наближення або скористатися методом відокремлення коренів згідно з теоремою Безу, як показано нижче у прикладі.

Функція *root* для *дійсного* початкового наближення повертає *дійсний* корінь, а для *комплексного* початкового наближення – *комплексний* корінь.

Використовуючи розширений варіант функції *root,* можна не означувати початкове наближення для обчислення кореня, а натомість виокремити область (інтервал) його розміщення.

Наприклад: обчислити корені рівняння $2.1x-5sin(x)^2+1=0$ (рис. 2.1).

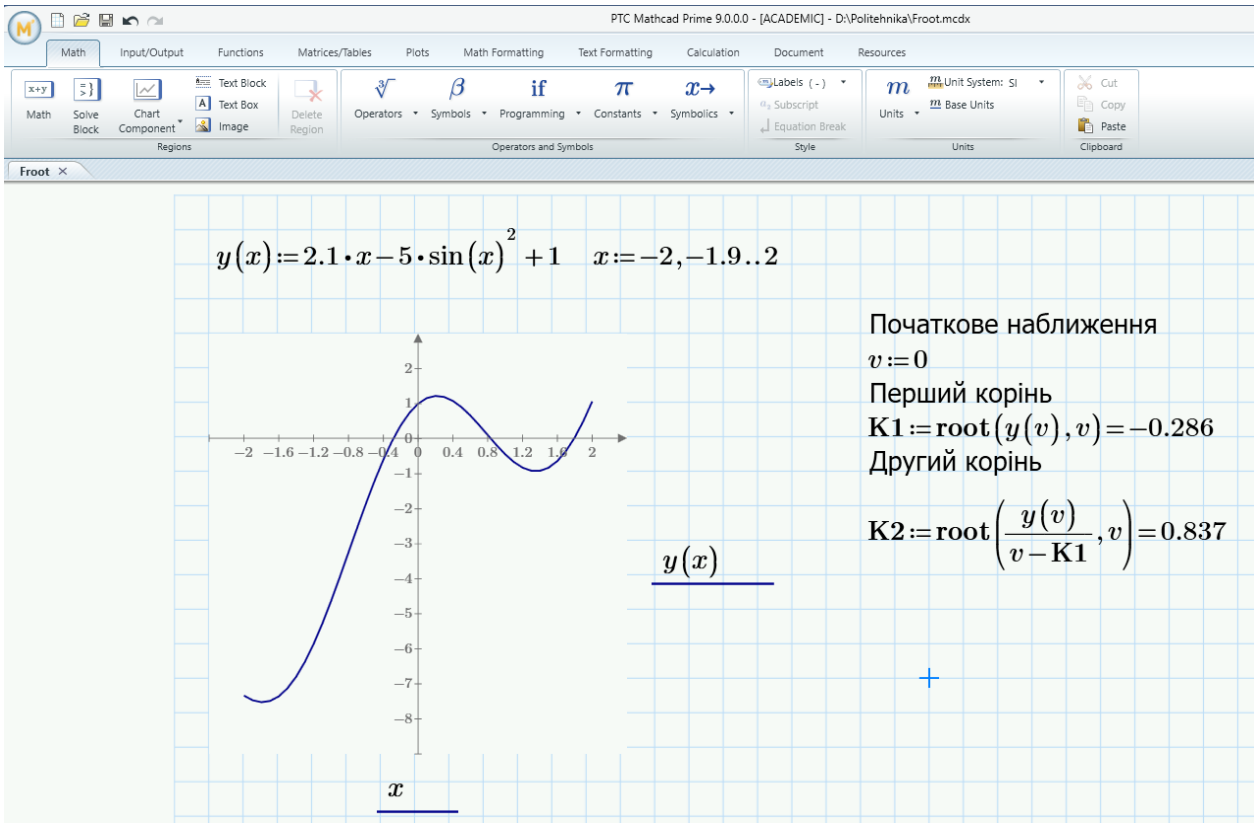


Рис. 2.1. Вигляд MathCad-документа, який відтворює послідовність дій для обчислення коренів рівняння $2.1x-5sin(x)^2+1=0$

У такому разі для обчислення кореня використовується метод Ріддера (*Ridder*) або Брента (*Brent*), а значення функції на кінцях інтервалу *f(a)* та *f(b)* повинні мати різні знаки. Викликають функцію так:

root(f(x), x, a, b),

де *a* – ліва межа області розміщення кореня;

b – права межа.

Обчислити корені рівняння *sinx + cosx = 0* в інтервалі $-\pi \le x \le \pi$.

Порядок виконання розрахунків:

- запишіть функцію користувача *f(s) := sins + coss*
- побудуйте графік цієї функції (графік перетинає вісь абсцис у двох точках, отже рівняння має два корені);
- визначте межі інтервалу ізоляції коренів [–2; 0] і [0, 2] та використайте їх у функції *root*.

Порядок виконання розрахунків відтворено на рис. 2.2.

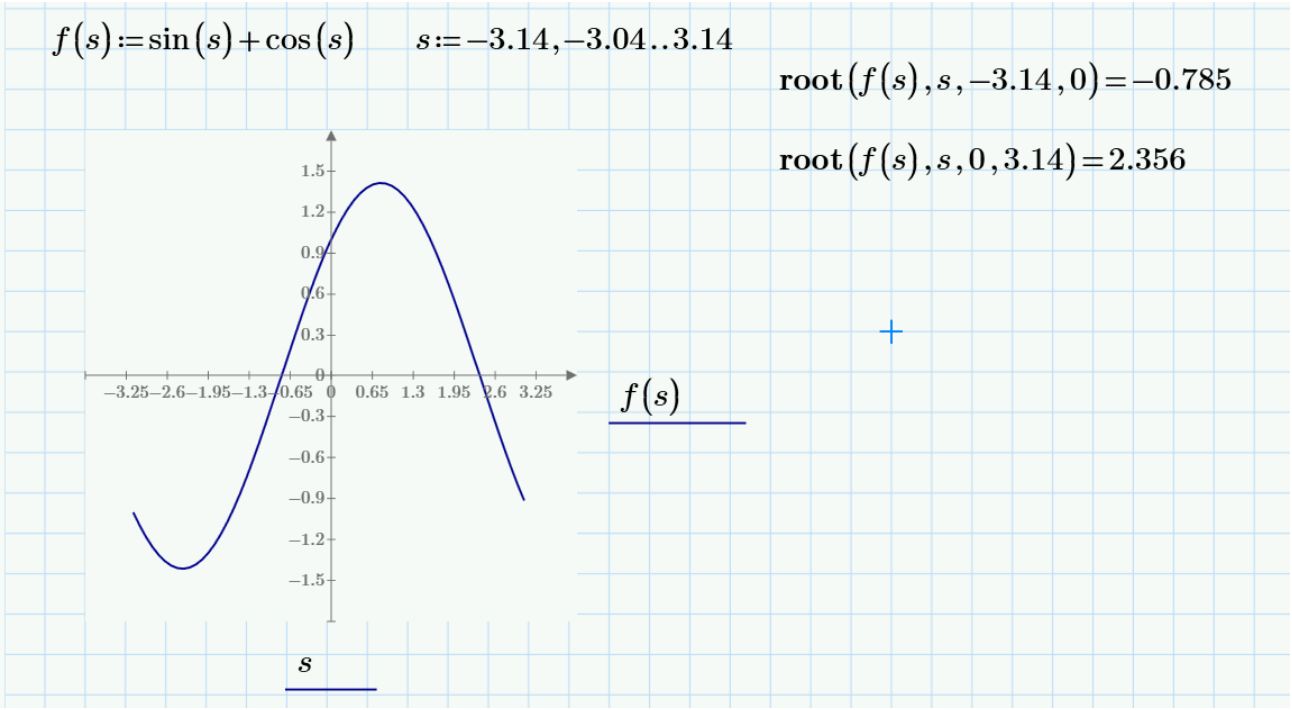


Рис. 2.2. Використання вмонтованої функції *root* для обчислення коренів рівняння *sinx + cosx = 0* у інтервалі -π ≤ x ≤ π

Розглянемо ще один приклад. Обчислити корені рівняння (рис. 2.3) $x^3 + 11x^2 + 3x + 135 = 0$.

Порядок виконання розрахунків:

- запишіть функцію користувача $f(x) := x^3 + 11x^2 + 3x + 135$;
- побудуйте графік цієї функції (графік перетинає вісь абсцис у трьох точках, отже, рівняння має три корені);
- визначте межі інтервалу ізоляції коренів [–10; -8], [-6, -4], [2, 4] та використайте їх у функції *root*.

Нагадаємо, перед використанням обох варіантів функції *root* доцільно наперед побудувати графік досліджуваної функції *f(x)* для оцінювання кількості коренів та їх положення, а потім, для покращення збіжності обчислення кореня, відповідно до графіка, вибрати точніше початкове наближення або область (інтервал) розміщення бажаного кореня.

Нижче перераховані типові причини, внаслідок яких функція *root* не може обчислити корінь рівняння або обчислює не той корінь, що потрібно:

- функція не має коренів;
- велика відстань між коренем і заданим його початковим наближенням;
- наявність локального екстремуму між початковим наближенням та коренем рівняння;
- наявність розриву у функції *f(x)* між початковим наближенням та коренем рівняння;
- корінь комплексний, а початкове значення наближення – дійсне (або навпаки);
- наявність великої кількості зосереджених у вузькому діапазоні коренів;
- плоска функція – крива функції дуже повільно прямує до осі.

Швидка зміна функції *f(x)* в околі її дійсного кореня може бути причиною отримання, як результату комплексного значення кореня з малою уявною частиною.

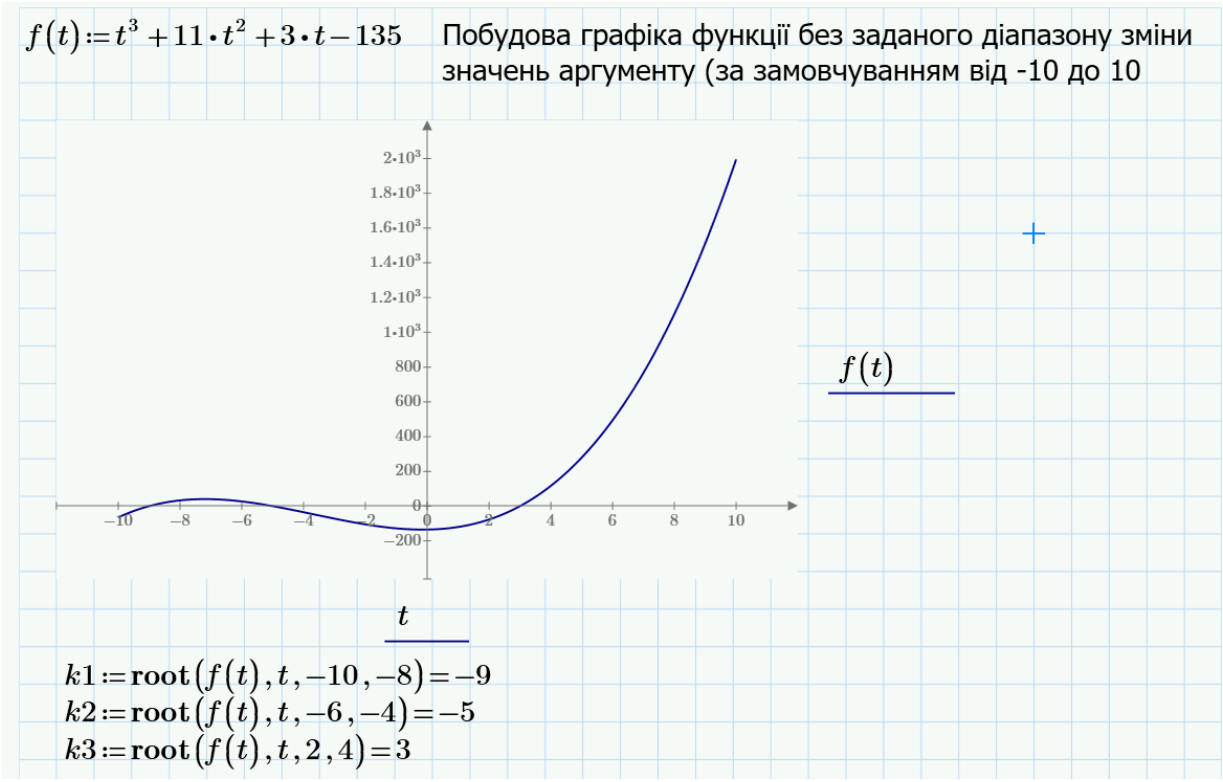


Рис. 2.3. Використання розширеного варіанту функції *root* (діапазон зміни значень аргументу функції заданий за замовчуванням)

Функцію *root* можна використовувати і у символьних обчисленнях для обчислення коренів рівнянь у загальному вигляді, що буде відображено у наступних розділах.

2.2. Розв'язання нелінійних рівнянь за допомогою функції *polyroots*

Якщо функція у рівнянні є поліномом *n*-го степеня, тобто

$$f(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x^1 + a_0,$$

тоді для розв'язання можна використовувати вмонтовану функцію

polyroots(v),

де *v* – вектор коефіцієнтів полінома, в якому всі компоненти розташовуються за порядком зростання степеня.

Усі корені полінома обчислюють з використанням функції *polyroots(v)*, де *v* – вектор-стовпець коефіцієнтів полінома. Як результат своєї роботи функція повертає вектор розв'язку і не потребує для своєї роботи початкових наближень.

У формуванні вектора-стовпця коефіцієнтів полінома не можна ігнорувати нульові коефіцієнти (тобто якщо поліном третього степеня і немає елемента з другим степенем *x*, необхідно вказати 0).

Наприклад, обчислити корені полінома $2x^3 - 11x + 5 = 0$. Процес обчислення коренів полінома подано на рис. 2.4.

$$2 \cdot x^3 - 11 \cdot x + 5 = 0$$

$$V := \begin{bmatrix} 5 \\ -11 \\ 0 \\ 2 \end{bmatrix} \qquad \text{polyroots}(V) = \begin{bmatrix} -2.546 \\ 0.474 \\ 2.072 \end{bmatrix}$$

Рис. 2.4. Обчислення коренів полінома $2x^3 - 11x + 5 = 0$

2.3. Розв'язання нелінійного рівняння методом половинного поділу

Для розв'язання нелінійного рівняння методом половинного поділу скористаємося вашими знаннями з курсу вищої математики та програмування

мовою С++. Це дасть вам змогу не тільки пригадати основні засоби програмування, а і виконати порівняння результатів розрахунків, отриманих у пакеті MathCad та програмних кодів мовою С++.

Подамо деякі теоретичні пояснення для полегшення процесу розв'язання задачі.

Метод половинного поділу (або метод бісекції) – це числовий метод для обчислення коренів нелінійних рівнянь вигляду $f(x) = 0$ шляхом ітеративного звуження інтервалу, на якому відомо, що функція змінює знак.

Метод ґрунтується на теоремі про проміжні значення і гарантує обчислення кореня за певних умов (наприклад, неперервність функції на відтинку).

Задано рівняння $f(x) = 0$, яке на відтинку $[a; b]$ має єдиний розв'язок, при чому функція $f(x)$ на цьому відтинку є неперервною. Геометричне інтерпретування методу половинного поділу подано на рис. 2.5.

Для обчислення розв'язку рівняння поділимо відтинок $[a; b]$ навпіл точкою $c = (a + b)/2$. Якщо значення функції у цій точці відмінне від нуля ($f(c) \neq 0$), тоді можливі два випадки:

1. Функція $f(x)$ змінює знак на відтинку $[a; c]$, тобто $f(a) \cdot f(c) < 0$.
2. Функція $f(x)$ змінює знак на відтинку $[c; b]$, тобто $f(c) \cdot f(b) < 0$.

Вибираючи той відтинок, на якому функція змінює знак, і продовжуючи процес половинного поділу далі, отримаємо такий собі малий відтинок, який буде містити корінь рівняння $f(x) = 0$.

Алгоритм розв'язання задачі буде полягати у виконанні таких кроків:

1. Визначте початковий інтервал $[a, b]$:

Визначте відтинок $[a, b]$, на кінцях якого функція має протилежні знаки, тобто $f(a) \cdot f(b) < 0$. Це гарантує, що на цьому відтинку існує хоча б один корінь.

2. Обчисліть середину поточного інтервалу: $c = (a+b)/2$.
3. Перевірка кореня:

- якщо $f(c) = 0$, тоді c є точним коренем;

- якщо $f(a)\cdot f(c) < 0$, тоді корінь знаходиться на лівому інтервалі $[a, c]$;
- якщо $f(\text{c})\cdot f(\text{b}) < 0$, тоді корінь знаходиться на правому інтервалі $[c, b]$.

4. Звуження інтервалу:

- якщо корінь на $[a, c]$, тоді новий інтервал стає $[a, c]$ (тобто b замінюється на c);
- якщо корінь на $[c, b]$, тоді новий інтервал стає $[c, b]$ (тобто a замінюється на c).

5. Повторення: Повторюйте кроки 2-4, поки довжина інтервалу не стане меншою за задану точність $f(c) < \varepsilon$, або поки не буде досягнуто іншої умови припинення обчислень.

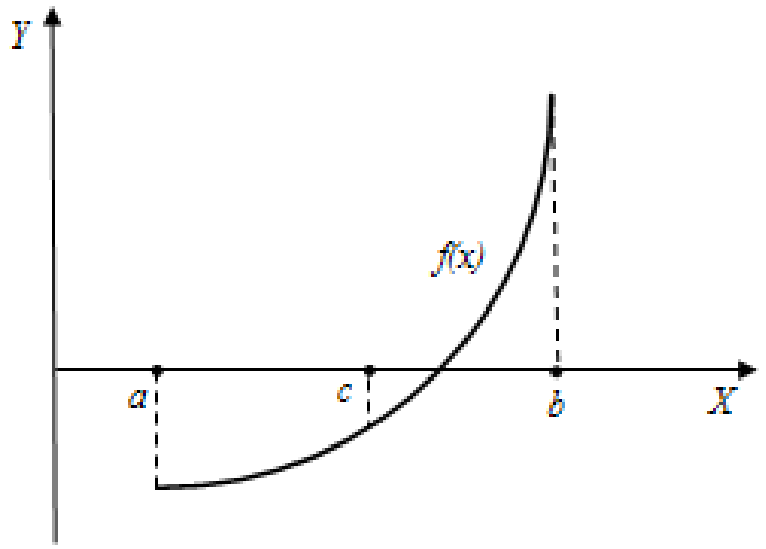


Рис. 2.5. Геометричне інтерпретування методу половинного поділу

Приклад проєкту програмного коду мовою C++ для розв'язання рівняння

$$x^3 - 18x - 83 = 0$$

методом половинного поділу (дихотомії):

```
// Метод дихотомії (половинного поділу)
#include <iostream>
#include <cmath>
using namespace std;
double Fx(double x)
```

```
{
// My function.
    return x * x * x - 18 * x - 83.0;
}

double Dyhotomy(double a, double b, double e, double
(*Fx)(double))
{
// Dichotomy method
    double c;
    while (b - a > e)
    {
        c = (a + b) / 2;
        if (Fx(b) * Fx(c) < 0) a = c;
        else b = c;
    }
    return (a + b) / 2;
}

int main()
{
    double a, b, e, res;
    cout << "\nBegin of interval a = ";
    cin >> a;
    cout << "\nEnd of interval b = ";
    cin >> b;
    cout << "\nPrecision e = ";
    cin >> e;
    res = Dyhotomy(a, b, e, Fx);
```

```
    cout << "\nCalculation result Dichotomy method = " <<
res;
    return 0;
}
```

Результат виконання програмного коду:

```
Begin of interval a = 2.0

End of interval b = 10.0

Precision e = 0.0001

Calculation result Dichotomy method = 5.70511
```

Приклад програмного коду мовою C++ для розв’язання рівняння

$$x^3 - 18x - 83 = 0$$

методом хорд:

```
//Метод хорд
#include <iostream>
#include <cmath>
using namespace std;
double Fx(double x)
{
// My function.
    return x * x * x - 18 * x - 83.0;
}

double Meth_Hord(double a, double b, double e, double
(*Fx)(double))
{
// Chord method
```

```
    double next = 0.0;
    double tmp;

    do
    {
        tmp = next;
        next = b - Fx(b) * (a - b) / (Fx(a) - Fx(b));
        a = b;
        b = tmp;
    } while (fabs(next - b) > e);
    return next;
}
int main()
{
    double a, b, e, res;
    cout << "\nBegin of interval a = ";
    cin >> a;
    cout << "\nEnd of interval b = ";
    cin >> b;
    cout << "\nPrecision e = ";
    cin >> e;
    res = Meth_Hord(a, b, e, Fx);
    cout << "\nCalculation result Chord method = " <<
res;
    return 0;
}
```

```
Begin of interval a = 2.0

End of interval b = 10.0
```

```
Precision e = 0.0001

Calculation result Chord method = 5.70512
```

2.4. Розв'язування системи нелінійних рівнянь

Для розв'язування системи лінійних і нелінійних рівнянь, нерівностей та оптимізаційних задач використовується блок розв'язку *Solve Block*.

Solve Block здійснює пошук розв'язку шляхом ітерацій, починаючи з заданих значень початкових наближень. За допомогою ітераційного процесу, який використовується *Solve Block*, можна отримати розв'язок системи нелінійних рівнянь. Розв'язати таку систему за допомогою матричних обчислень надзвичайно складно, якщо не неможливо.

Крім функції у *Solve Block,* необхідно задати початкові наближення, а також початкові або граничні умови. Початкове наближення для коренів системи рівнянь можна визначити графічним способом.

Solve Block – це спеціальна обчислювальна область, яка використовується для завдання початкових умов, рівнянь і обмежень розв'язуваної задачі. Введення *Solve Block* у MathCad-документ здійснюється на вкладці *Math* активуванням *Solve Block* (рис. 2.6).

Кожен *Solve Block* може мати лише одну функцію *find* або *minerr*. Однак ви можете визначити функцію, таку як *f(a):= find(x)* у кінці одного *Solve Block*, а потім використовувати цю саму функцію в іншому блоці розв'язання. Перший *Solve Block* називається параметризованим блоком розв'язання.

Solve Block може працювати з системою, яка містить до 400 рівнянь і до 200 додаткових умов для системи нелінійних рівнянь. Використання блока розв'язку просте (як і, до речі, інших конструкцій MathCad) і зрозуміле з поданого нижче прикладу. Зазначимо, що у рівняннях усередині *Solve Block* необхідно вказувати не звичайний, а потовщений знак символьного дорівнює ═ (набирається у блоці *Operators* → *Comparison*).

Обчислення кореня здійснюється за допомогою двох функцій *find* та *minerr*, звернення до яких розташовуються у *Solve Block.* Функції *find* та *minerr* можуть використовуватися *тільки* у *Solve Block*.

Якщо під час розв'язування деяких типів задач (апроксимаційні задачі, оптимізаційні задачі) система не може обчислити точний розв'язок або якщо для цієї задачі він не існує взагалі, тоді компілятор MathCad генерує і виводить на екран відповідне повідомлення. У такому разі доцільно наприкінці *Solve Block* застосовувати замість функції *find* функцію *minerr*.

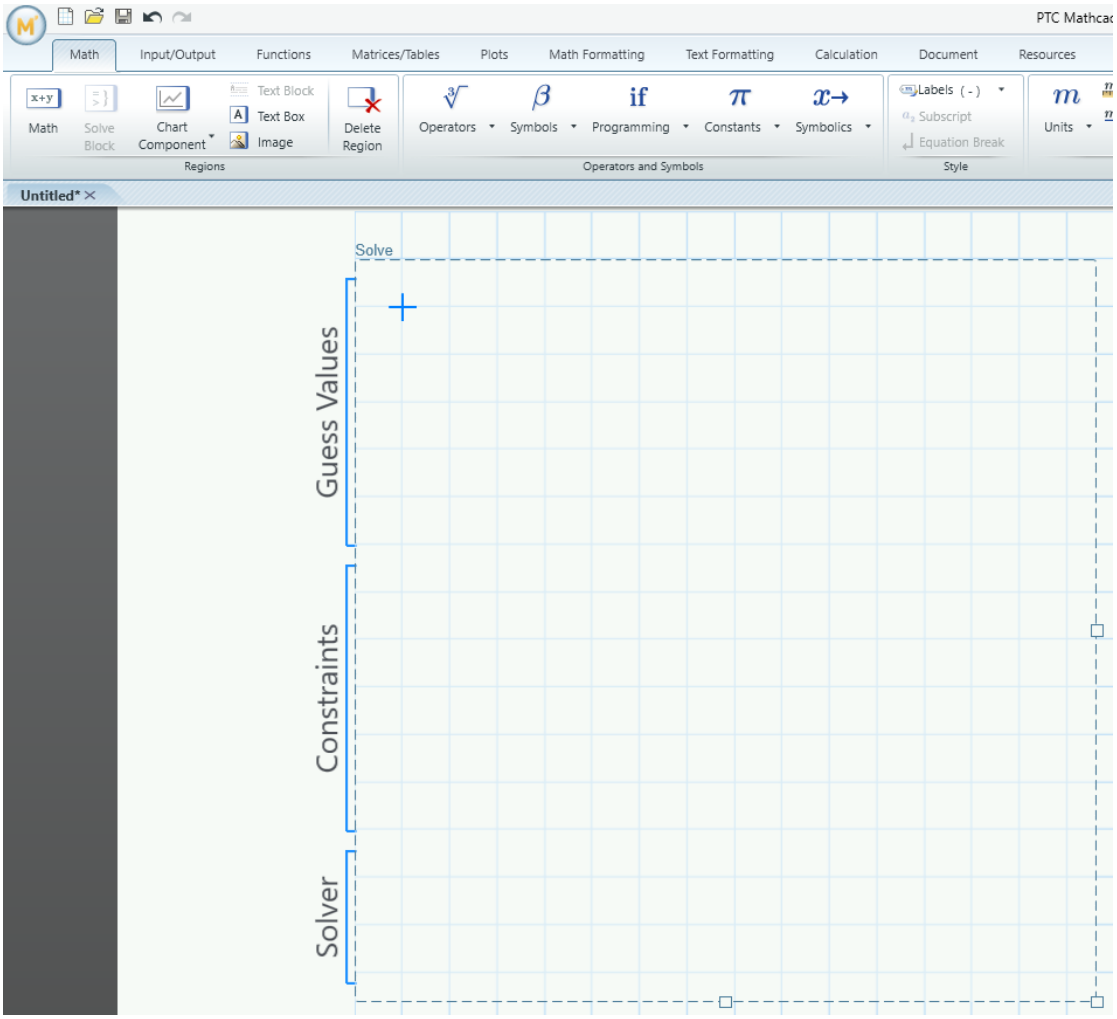


Рис. 2.6. Введення *Solve Block* у MathCad-документ

Відмінність між ними полягає у тому, що функція *minerr* обчислює наближений розв'язок системи, для якої сума квадратів нев'язок рівнянь системи є найменшою. Якщо ця система рівнянь має точний розв'язок, тоді результати виконання обох функцій *find* і *minerr* збігаються.

Функція *find* уточнює корінь рівняння, виклик цієї функції має вигляд *find(x),* де *x* – змінна, за якою уточнюється корінь. Якщо кореня рівняння на заданому інтервалі не існує, тоді слід скористатися функцією *minerr(x)*, яка повертає *наближене значення кореня*.

Наприклад, розв’язати систему нелінійних рівнянь:

$$\begin{cases} 3ab + c\sqrt{b} = 5 \\ 0.5a + 5.4b^3 - 6c = 10 \\ 2a^2 - log_3 + sin(c) = 3 \end{cases}$$

Процес розв’язання задачі подано на рис. 2.7.

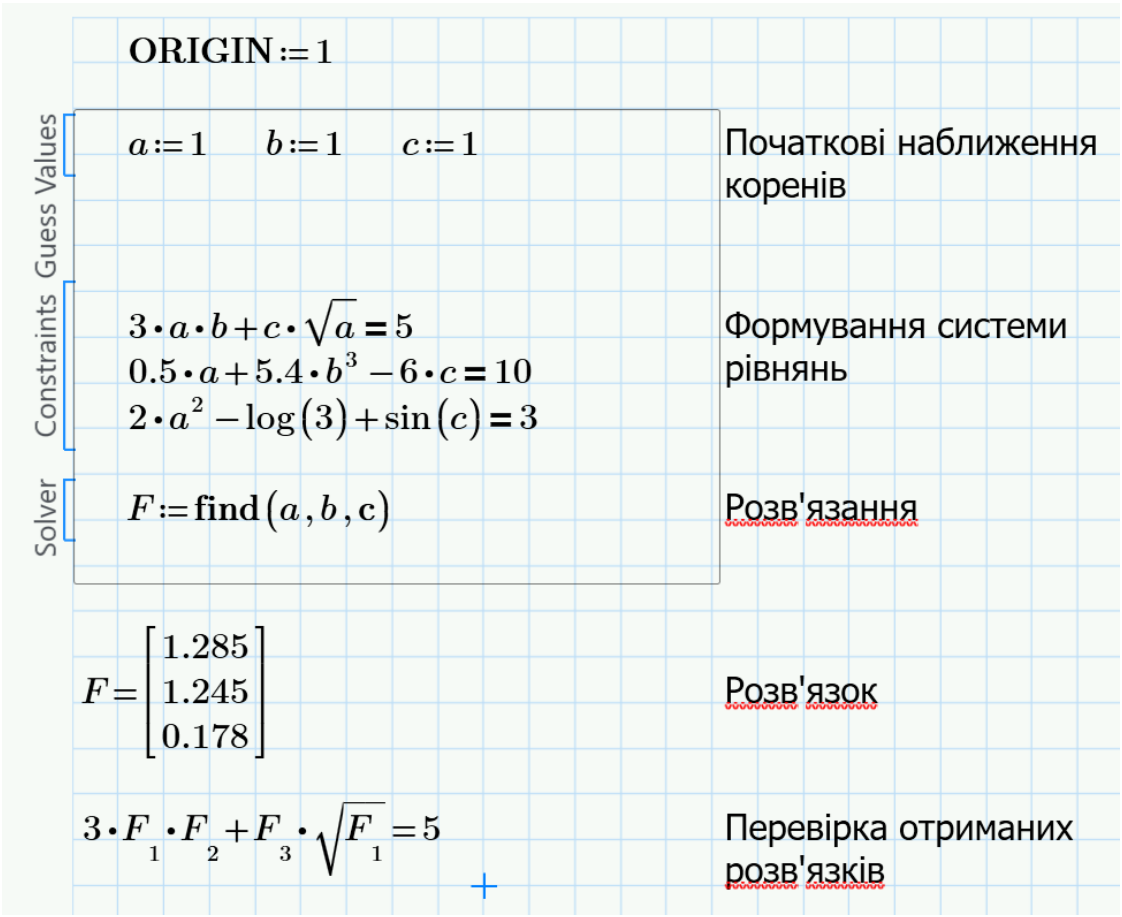


Рис. 2.7. Процес розв’язання задачі у MathCad-документі

Розглянемо ще один приклад. Обчислити корені системи рівнянь:

$$\begin{cases} y = -0.5 - 0.5x \\ y = x^2 - 1.75x + 0.25 \end{cases}$$

Щоб перевірити спільність рівнянь системи (тобто перевірити існування розв’язків), побудуємо графіки функцій, отриманих з рівнянь системи, рис. 2.8.

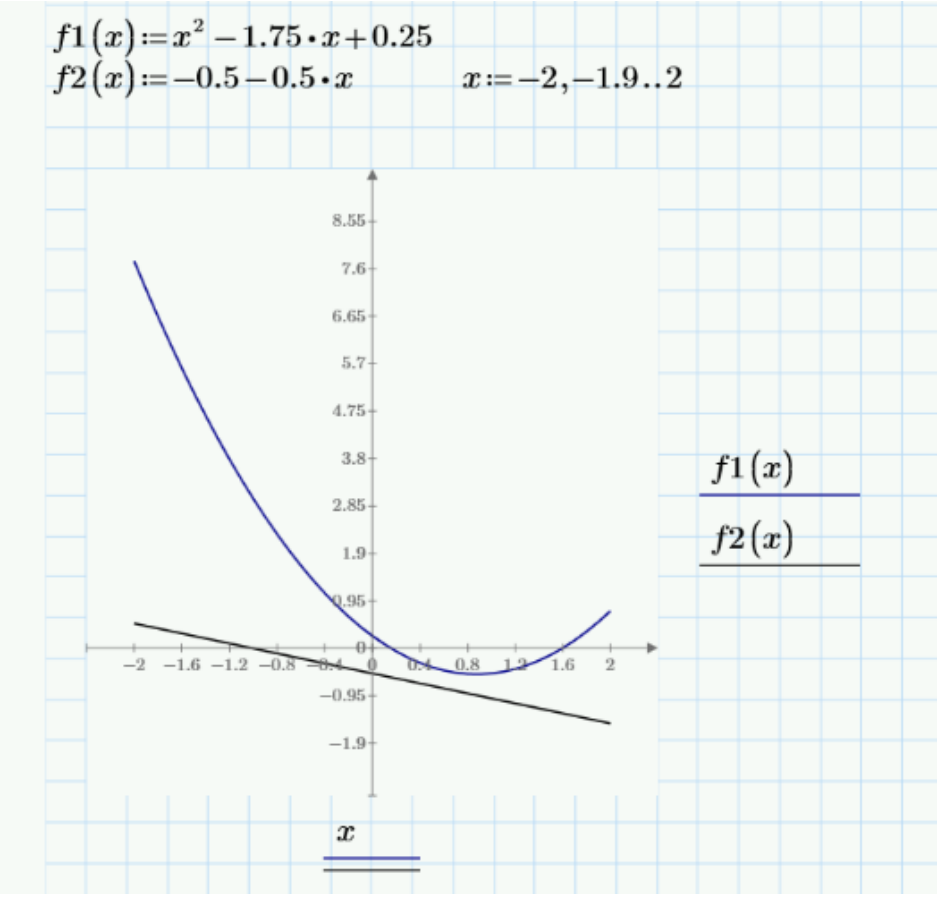


Рис. 2.8. Графіки функцій, отриманих з рівнянь системи

Графіки не перетинаються, отже система рівнянь не є спільною, і для обчислення наближеного розв'язку скористаємося функцією *minerr*.

Для цього вставимо два *Solve Block* у MathCad-документ. У першому (лівому) блоці використовується функція *find*, у другому – *minerr*. За спроби обчислити розв'язки за допомогою функції *find* компілятором генерується повідомлення про помилку.

При використанні функції *minerr* отримаємо наближені значення розв'язків системи нелінійних рівнянь – вектор *F1*. Перше значення якого – це значення *x*, друге значення – значення *y*.

Хоча обчислені значення не перетворюють рівняння системи на тотожності, тобто не належать одночасно кривим *f1(x)* та *f2(x)*, вони дають координати точки, яка знаходиться на мінімальній відстані, як від кривої *f1(x),* так і від кривої *f2(x)*.

Якщо виникають проблеми з розв'язанням системи нелінійних рівнянь з використанням *Solve Block* можна:

- змінити початкові умови;
- задати додаткову логічну умову для звуження області пошуку розв'язків;
- використати *minerr* замість *find* для обчислення наближених розв'язків.

Процес розв'язання задачі подано на рис. 2.9.

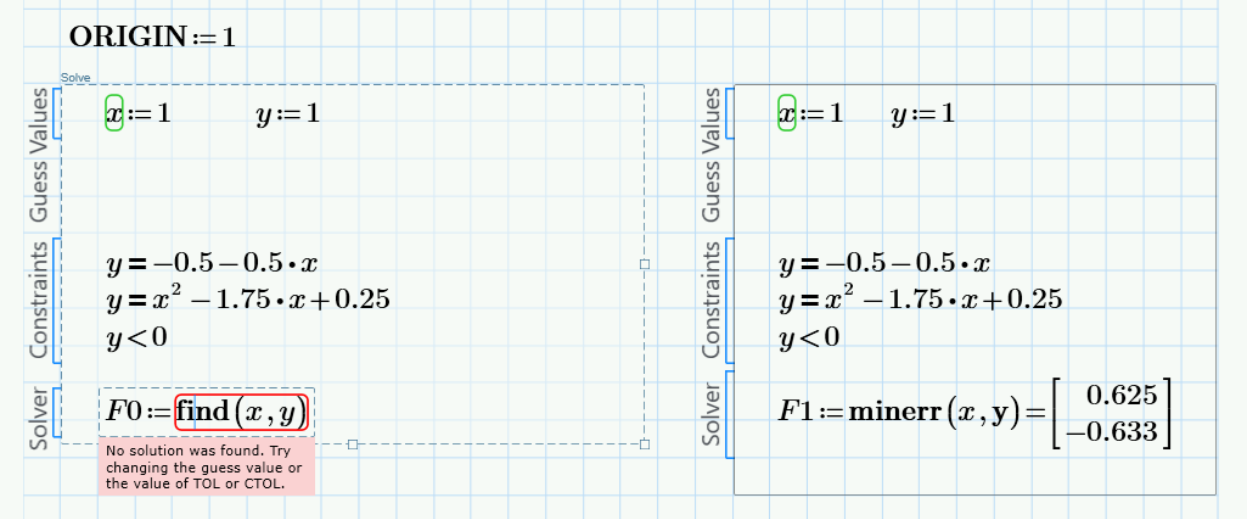


Рис. 2.9. Процес розв'язання задачі

Solve Block отримують розв'язки, використовуючи початкові наближення, а потім ітераційно рухаються до їх обчислення. Розв'язки часто є наближенням, яке потрапляє у межі допуску збіжності *TOL* та допуску обмежень *CTOL* реальних розв'язків.

Ви можете вказати початкові наближення, які обмежують бажаний розв'язок певним діапазоном простору розв'язків. Функції *Solve Block* автоматично вибирають відповідний алгоритм для розв'язання вашої задачі.

2.5. Робота з початковими наближеннями

Розглянемо приклад отримання розв'язків з використанням *Solve Block* з кількома варіантами початкових наближень. Для цього виконаємо таку послідовність дій:

1. Вставте *Solve Block* у *Engineering Notebook* та скористайтеся функцією find, щоб розв'язати систему рівнянь з двома змінними (рис. 2.10).

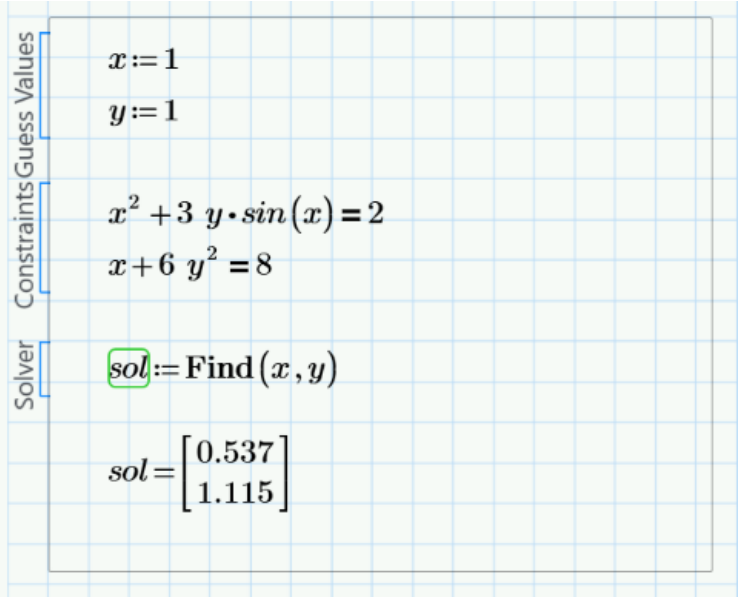


Рис. 2.10. Розв'язання системи рівнянь з двома змінними

Варіант початкових наближень дає відповідний варіант розв'язку системи рівнянь.

2. Вставте наступний *Solve Block*. Вкажіть обмеження та визначте функцію двох невідомих x та y, але не визначайте початкові наближення (рис. 2.11).

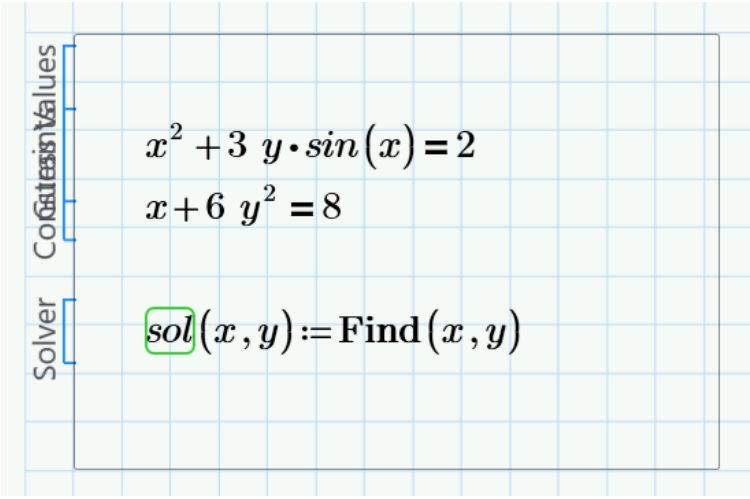


Рис. 2.11. Розв'язання системи рівнянь з двома змінними визначайте без початкових наближень

3. Переконайтеся, що ви отримаєте той же розв'язок, що й перед цим, передаючи ті ж значення початкових наближень (рис. 2.12).

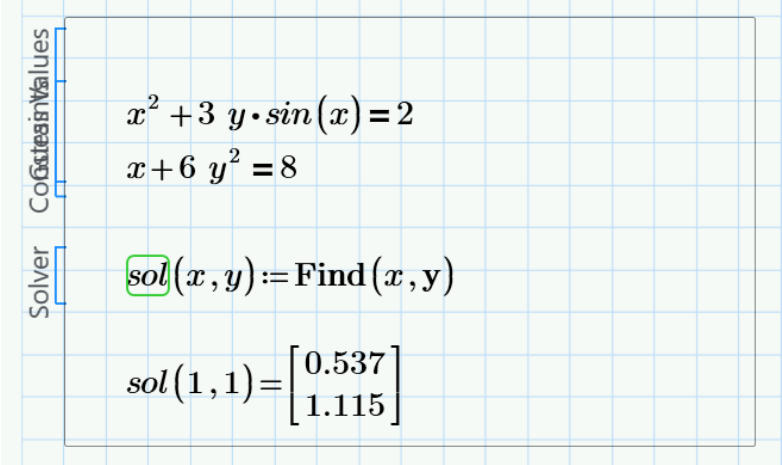


Рис. 2.12. Розв'язання системи рівнянь з двома змінними з визначенням початкових наближень

4. Передайте у функцію кілька варіантів початкових наближень та виконайте аналіз, як зміниться розв'язок (рис. 2.13).

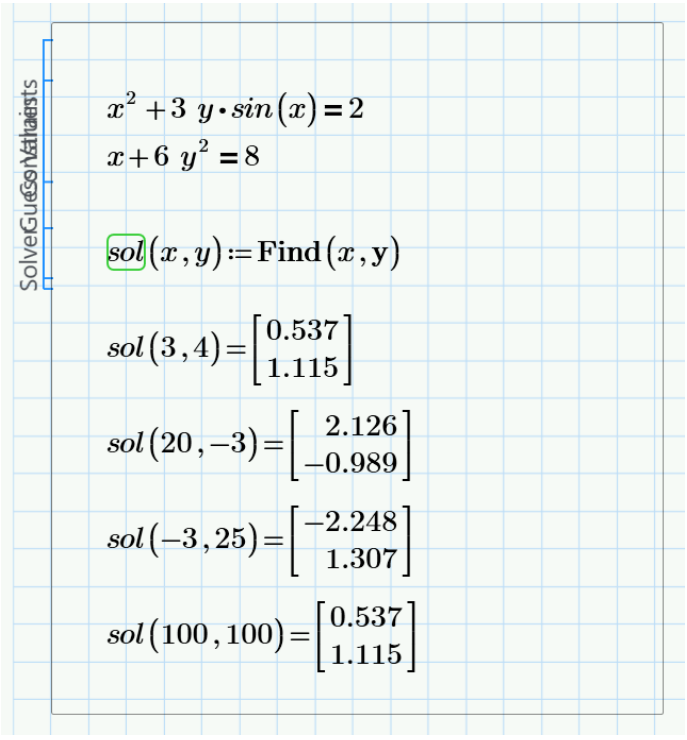


Рис. 2.13. Розв'язання системи рівнянь з двома змінними з визначенням кількох варіантів початкових наближень

Цей метод дозволяє отримати кілька результатів в одному блоці розв'язання та вибирати раціональні значення початкових наближень для вашої задачі.

Початкові наближення для всіх невідомих змінних потрібні під час розв'язання систем лінійних або нелінійних рівнянь. Для лінійних систем початкові наближення використовуються лише для визначення розміру результату розв'язку, їхні значення не мають значення.

Подані на рис. 2.14 блоки розв'язання показують, як різні початкові наближення значень *x* не впливають на результат розв'язання.

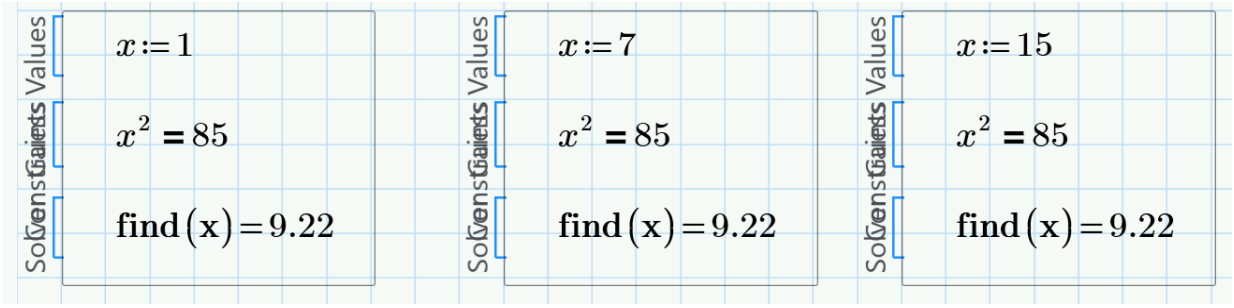


Рис. 2.14. Вплив початкових наближень значення *x* на результат розв'язання

Для нелінійних рівнянь розв'язок дуже чутливий до початкових наближень. Для *find* та *minerr* необхідно визначити невідомі змінні під час виклику функції *Solve Block*. Значення початкових наближень повинні мати такі ж ідентифікатори, як і невідомі змінні (рисю 2.15).

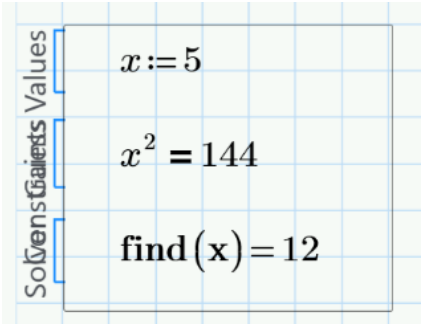


Рис. 2.15. Вибір ідентифікаторів та значення початкових наближень

Для minimize та maximize невідомі змінні зрозумілі, оскільки вони є аргументами цільової функції (функції для оптимізації). Однак ви повинні визначити початкове наближення для кожної з невідомих змінних у функції *Solve Block*. Порядок, у якому ви передаєте початкові наближення для minimize, має бути таким самим, як і порядок аргументів. Тут *a* – це початкове наближення для θ, а *b* – початкове наближення для φ (рис. 2.16).

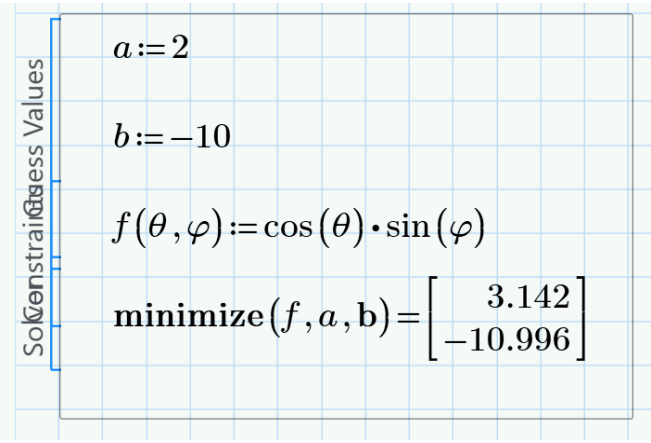


Рис. 2.16. Порядок передавання початкових наближень для minimize

2.6. Параметризація блоків розв'язання

Кожен *Solve Block* повертає один розв'язок, і ви можете використовувати цей розв'язок нижче або праворуч від області *Solve Block*. Ви можете додати параметри до *Solve Block*. Виведення блоку розв'язання тоді є функцією параметрів.

Приміром, використаємо функцію find, щоб обчислити розв'язок. Коли ви вказуєте нове значення параметра, функція викликає *Solve Block* та повертає розв'язок для цього значення. Це особливо корисно для побудови графіків розв'язків *Solve Block* (рис. 2.17).

Крім того, ви можете працювати з розв'язком *Solve Block* всередині іншого *Solve Block*. Обмеження (*2 - 0.1x*) застосовується до функції *f(x)*, яка визначена в першому блоці розв'язання.

Розв'язок, обчислений функцією *find*, є перетином лінії обмеження з *f(x)* (рис. 2.18).

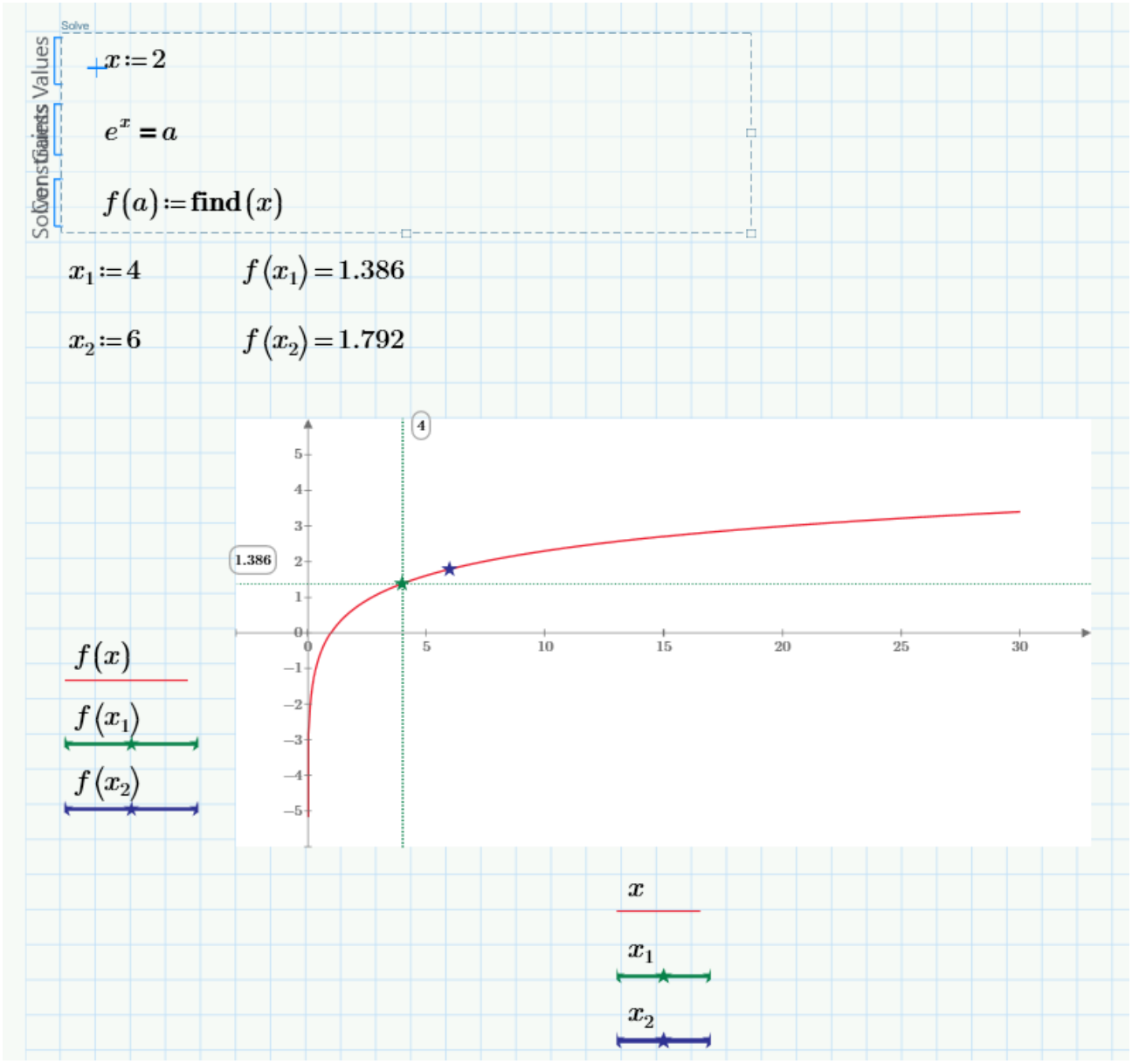


Рис. 2.17. Побудова графіків розв’язків *Solve Block*

Результат розв’язання може бути призначений одній змінній, яку можна використовувати у всьому робочому аркуші або всередині інших блоків розв’язання.

Тут функція *find* не може обчислити розв’язок, оскільки обмеження є функцією, яка не перетинається з функцією *f(x)* (рис. 2.19).

Обчисліть функцію *f(x)* за початковим наближенням та призначте результат змінній.

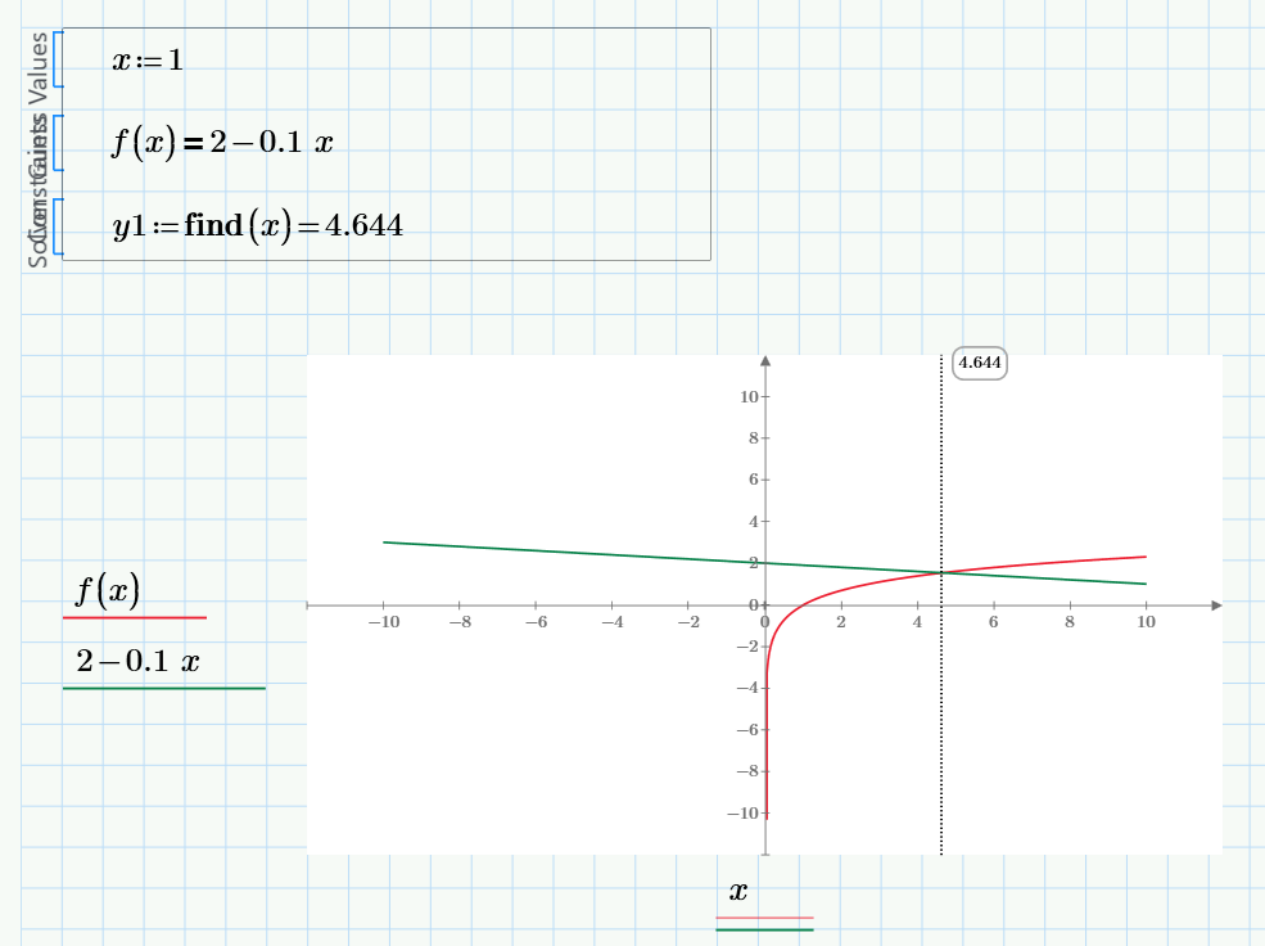


Рис. 2.18. Розв’язок обчислень функцією *find*

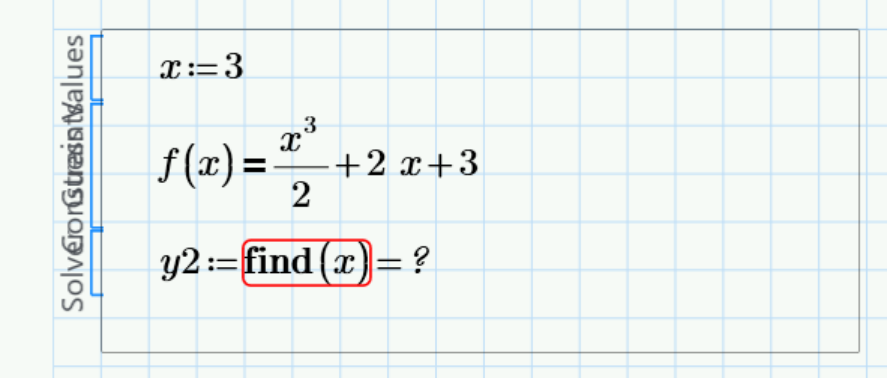


Рис. 2.19. Випадок, коли обмеження є функцією, яка не перетинається з функцією *f(x)*

Обмеження тепер застосовується до константи *sol* або *f(3)*, а розв’язок, обчислений функцією *find*, є перетином *sol* та функції обмеження (рис. 2.20).

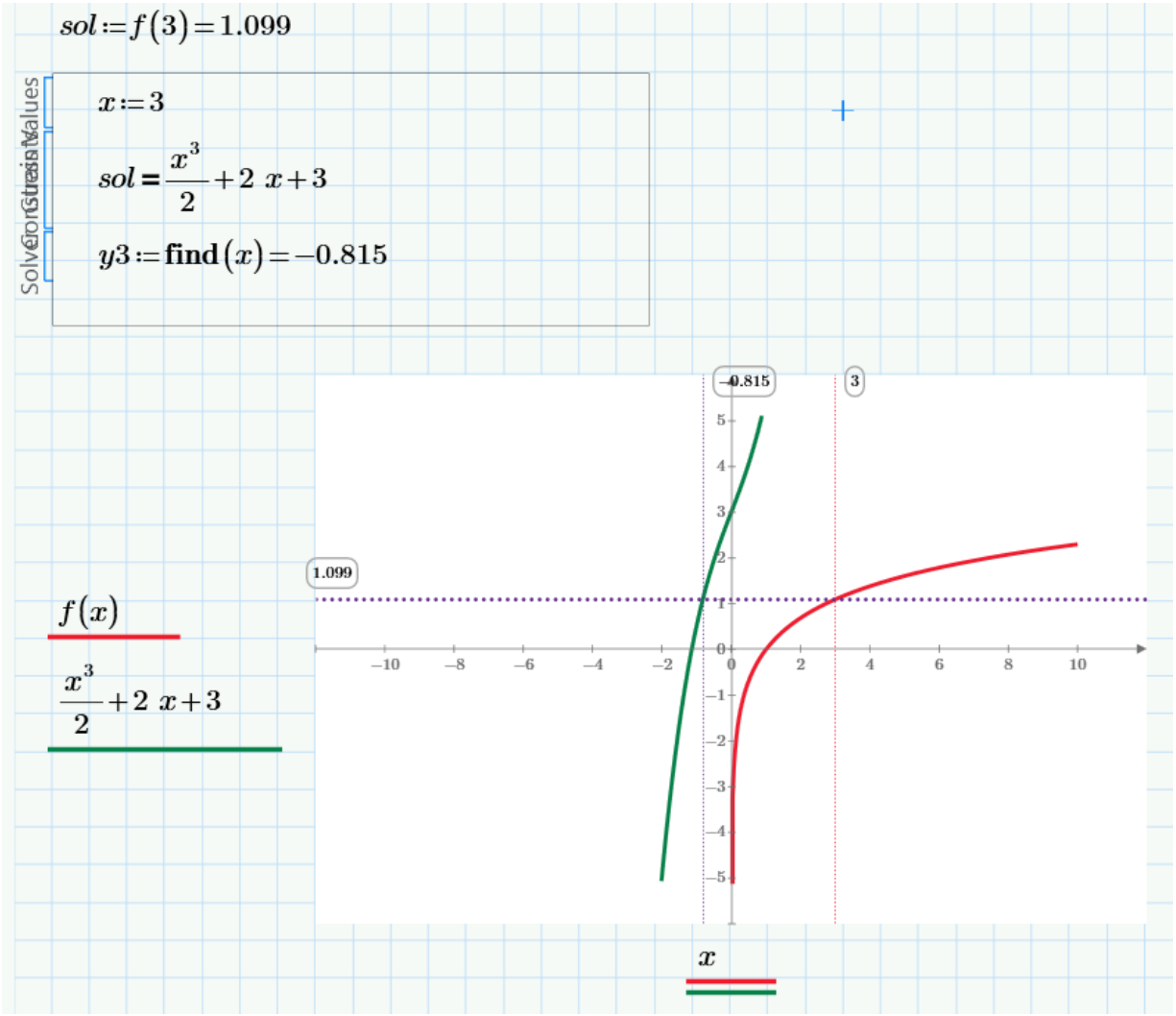


Рис. 2.20. Випадок, коли розв’язок, обчислений функцією *find*, є перетином *sol* та функції обмеження

Індивідуальні завдання

Завдання 1. Розв’язати рівняння, використовуючи засоби пакету MathCad, та створити проєкт програмного коду розв’язання задачі (методом половинного поділу та методом хорд) мовою C++.

Для обчислення ізольованого кореня рівняння *f(x) = 0* з одним невідомим використати вмонтовану функцію пакету MathCad *root(f(x), x)*.

Усі корені полінома обчислити з використанням функції *polyroots(v)*, де *v* – вектор-стовпець коефіцієнтів полінома. Варіанти індивідуальних завдань подані у табл. 2.1.

Таблиця 2.1

Варіанти індивідуальних завдань для розв’язання нелінійного рівняння

Варіант	Рівняння
1	$9x^4 + 8x^3 + 1.5x^2 + 2x - 10 = 0$
2	$3x^4 + 4x^3 - 12x^2 - 5 = 0$
3	$6x^4 + 4x^3 + x^2 + x - 10 = 0$
4	$x^4 + 4x^3 - 8x^2 - 17 = 0$
5	$4.5x^4 - 4x^3 + 1.5x^2 - 2x - 7 = 0$
6	$6x^4 + 8x^3 - 24x^2 - 7 = 0$
7	$x^4 + 2x^3 + 2x^2 + 6x - 5 = 0$
8	$x^4 - 12x^3 - 9 = 0$
9	$6x^4 + 4x^3 - x^2 - x - 10 = 0$
10	$x^4 - 108x + 7 = 0$
11	$3x^4 + 2x^3 + x^2 + x - 5 = 0$
12	$9x^4 + 12x^3 - 36x^2 - 2 = 0$
13	$3x^4 - 4x^3 + x^2 - 2x - 5 = 0$
14	$3x^4 - 4x^3 + 2x^2 - 4x - 1 = 0$
15	$3x^4 - 10x^3 + x^2 - 5x - 3 = 0$
16	$2x^4 - 2x^3 - 4x^2 + 6x + 5 = 0$
17	$4x^4 - 4x^3 + 2x^2 - 3x - 9 = 0$
18	$2x^4 - 8x^3 - 16x^2 - 1 = 0$
19	$3x^4 - x^3 + 10x^2 - 5x - 3 = 0$
20	$4x^4 + 4x^3 - 6x^2 - 9x - 1 = 0$
21	$3x^4 - 4x^3 + 2x^2 - 2x - 3 = 0$
22	$x^4 + 2x^3 + 2x^2 + 6x - 3 = 0$
23	$2x^4 + 4x^3 + x^2 + 3x - 6 = 0$
24	$3x^4 - 2x^3 - x^2 + x - 4 = 0$

25	$6x^4 - 4x^3 - x^2 + x - 7 = 0$
26	$3x^4 + 4x^3 - 12x^2 - 1 = 0$
27	$x^4 - 2x^3 + 2x^2 - 6x + 1 = 0$
28	$3x^4 - 4x^3 + x^2 - 2x - 1 = 0$
29	$2x^4 + 4x^3 - x^2 - 3x - 1 = 0$
30	$x^4 - 4x - 1 = 0$

Завдання 2. Розв'язати систему нелінійних рівнянь, використовуючи засоби пакету MathCad. Обчислення коренів виконати за допомогою двох функцій – *find* та *minerr*, звернення до яких розташувати у *Solve Block.* Варіанти індивідуальних завдань подані у табл. 2.2.

Таблиця 2.2

Варіанти індивідуальних завдань для розв'язання системи нелінійних рівнянь

№ 1 $\begin{cases}\sin(x+1) - y = 1.2 \\ 2x + \cos y = 2\end{cases}$	№ 2 $\begin{cases}\sin(y+1) - x = 1.2 \\ 2y + \cos x = 2\end{cases}$
№ 3 $\begin{cases}\sin(x+1) - y = 1 \\ 2x + \cos y = 2\end{cases}$	№ 4 $\begin{cases}\cos(x-1) + y = 0.5 \\ x - \cos y = 3\end{cases}$
№ 5 $\begin{cases}\cos(y-1) + x = 0.5 \\ y - \cos x = 3\end{cases}$	№ 6 $\begin{cases}\cos(x-1) + y = 0.8 \\ x - \cos y = 2\end{cases}$
№ 7 $\begin{cases}\sin x + 2y = 2 \\ \cos(y-1) + x = 0.7\end{cases}$	№ 8 $\begin{cases}\sin y + 2x = 2 \\ \cos(x-1) + y = 0.7\end{cases}$
№ 9 $\begin{cases}\sin x + y = 1.2 \\ \cos(y-1) + x = 1\end{cases}$	№ 10 $\begin{cases}\cos x + y = 1.5 \\ 2x - \sin(y-0.5) = 1\end{cases}$
№11 $\begin{cases}\cos y + x = 1.5 \\ 2y - \sin(x-0.5) = 1\end{cases}$	№12 $\begin{cases}cos\, x + y = 1.2 \\ 2x - \sin(y-0.5) = 2\end{cases}$

№13 $\begin{cases}\sin(x+0.5)-y=1\\ \cos(y-2)+x=0\end{cases}$	№14 $\begin{cases}\sin(y+0.5)-x=1\\ \cos(x-2)+y=0\end{cases}$
№15 $\begin{cases}\sin(x+0.5)-y=1.2\\ \cos(y-2)+x=0\end{cases}$	№16 $\begin{cases}\cos(x+0.5)+y=0.8\\ \sin y-2x=1.6\end{cases}$
№17 $\begin{cases}\cos(y+0.5)+x=0.8\\ \sin x-2y=1.6\end{cases}$	№18 $\begin{cases}\cos(x+0.5)+y=1\\ \sin y-2x=0\end{cases}$
№19 $\begin{cases}\sin(x-1)=1.3-y\\ x-\sin(y+1)=1.8\end{cases}$	№20 $\begin{cases}\sin(y-1)+x=1.3\\ y-\sin(x+1)=0.8\end{cases}$
№21 $\begin{cases}\sin(x-1)+y=1.5\\ x-\sin(y+1)=1\end{cases}$	№22 $\begin{cases}2y-\cos(x+1)=0\\ x+\sin y=-0.4\end{cases}$
№23 $\begin{cases}2x-\cos(y+1)=0\\ y+\sin x=-0.4\end{cases}$	№24 $\begin{cases}\sin(y+1)-x=1\\ 2y+\cos x=2\end{cases}$
№25 $\begin{cases}\cos(x+0.5)-y=2\\ \sin y-2x=1\end{cases}$	№26 $\begin{cases}\cos(y+0.5)-x=2\\ \sin x-2=1\end{cases}$
№27 $\begin{cases}\cos(y-1)+x=0.8\\ y-\cos x=2\end{cases}$	№28 $\begin{cases}\sin(x+2)-y=1.5\\ x+\cos(y-2)=0.5\end{cases}$
№29 $\begin{cases}\sin(y+2)-x=1.5\\ y+\cos(x+2)=0.5\end{cases}$	№30 $\begin{cases}\cos(x-1)+y=1\\ \sin y+2x=1.6\end{cases}$

РОЗДІЛ 3
ЕЛЕМЕНТИ ВЕКТОРНОЇ І МАТРИЧНОЇ АЛГЕБРИ

Переваги пакету MathCad особливо відчутні у роботі з масивами (векторами і матрицями). Здебільшого ця робота полягає у багаторазовому повторенні однотипних розрахунків зі всіма елементами масивів.

У пакеті MathCad використовуються масиви двох типів: одновимірні (вектори) та двовимірні (матриці).

Вектором називається стовпець чисел. Доступ до довільного елементу масиву можливий за його індексом, тобто порядкового номеру у послідовності значень.

Системна константа ORIGIN визначає номер початкового індексу масивів. За замовчуванням ORIGIN:=0. Це означає, що нумерація елементів масивів починається з нуля.

Щоб перший елемент масиву мав номер (індекс) 1, необхідно змінити значення системної змінної ORIGIN:=1.

Для оброблення одновимірних масивів у MathCad необхідно ввести дискретну змінну, яка символізуватиме *порядковий номер (індекс)* елемента у векторі $i = 1..n$, де n – кількість елементів.

Для звернення до елемента вектора використовується індексна змінна. Щоб ввести індексну змінну, натисніть кнопку M_i розділу *Vector/Matrix Operators* на панелі інструментів *Matrices/Tables*.

Матриця – прямокутна таблиця чисел. Кожен елемент має два індекси – номер рядка та номер стовпця.

Для оброблення двовимірних масивів у MathCad необхідно ввести дві дискретні змінні, перша з яких відповідає номерові рядка, а друга – номерові стовпця елемента матриці:

$i = 1..n$;

$j = 1..m$;

де i – кількість рядків n;

j – кількість стовпців m.

Звернення до значень елементів матриці здійснюється за іменем матриці та індексами, наприклад, A_{ij}.

Для роботи з векторами та матрицями у пакеті MathCad використовується вкладка *Matrices/Tables*. Щоб ввести індексну змінну A_{ij}, натисніть кнопку M_i розділу *Vector/Matrix Operators* на панелі інструментів *Matrices/Tables*.

Оператори роботи з векторами та матрицями вибираються при активуванні кнопки *Vector/Matrix Operator,* як це подано на рис. 3.1.

У відзначеному рамкою об’єкті на місці курсору необхідно ввести ім’я масиву та кількість елементів у індексі.

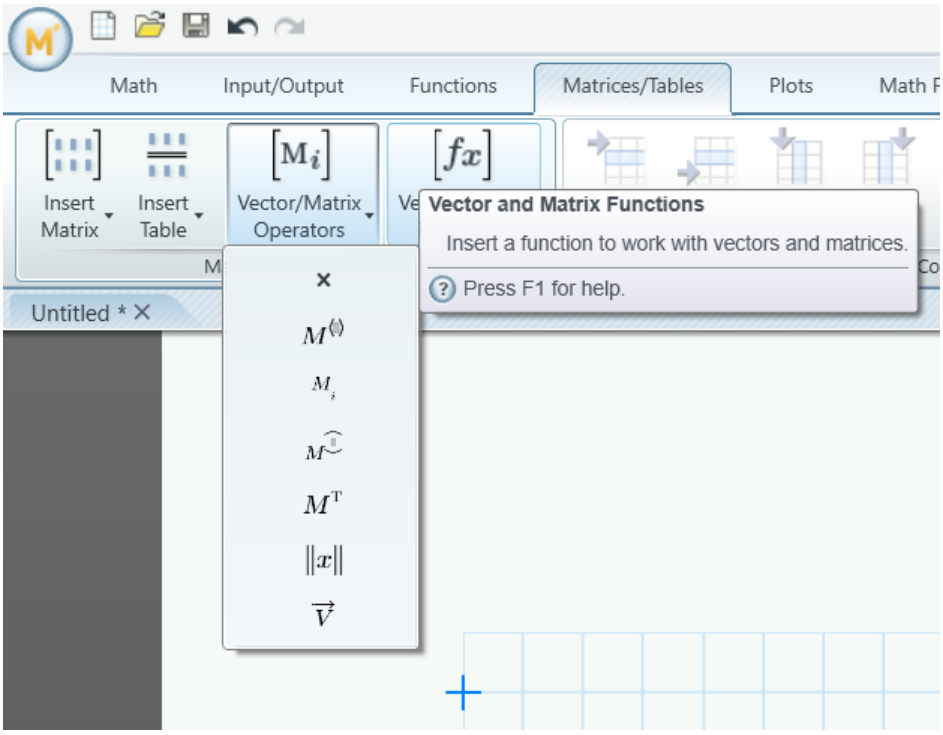


Рис. 3.1. Активування кнопки *Vector/Matrix Operator*

3.1. Створення вектора

Спосіб 1. З використанням шаблону.

Вкладка *Matrices/Tables*, кнопка *Insert Matrix*

У шаблоні, який з’явиться, відзначаємо мишкою необхідну конфігурацію (кількість стовпців) масиву (рис. 3.2).

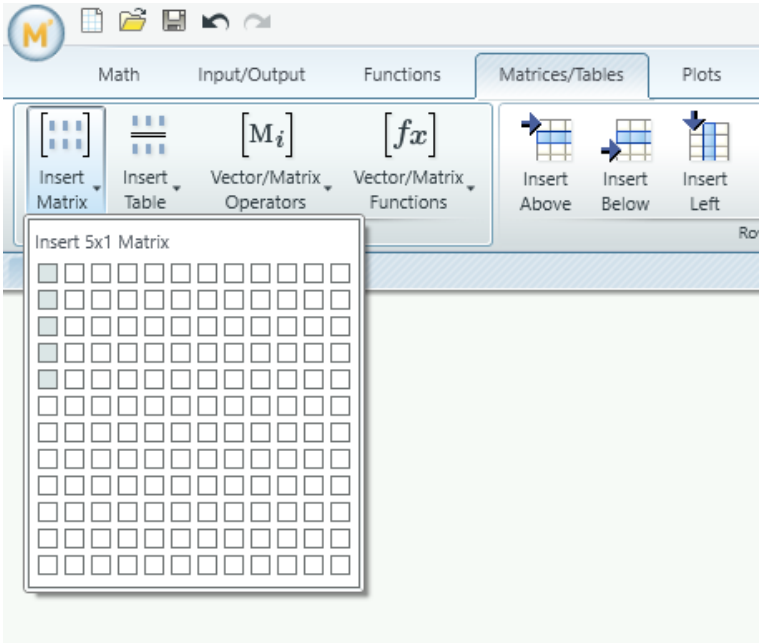


Рис. 3.2. Створення масиву з використанням шаблону

Системна змінна ORIGIN. Індекси елементів векторів та матриць у пакеті MathCad за замовчуванням починають свої значення з нуля. Для вектора першим буде нульовий елемент (елемент з індексом нуль), для матриці – рядок із номером «нуль» та стовпець із номером «нуль». Початкове значення нумерації елементів масивів керується значенням системної змінної ORIGIN, яка за замовчанням дорівнює нулю.

Однак часом виникає необхідність нумерувати елементи масиву з 1. Для цього треба змінити значення змінної ORIGIN за допомогою оператора присвоєння або звернутися до вкладки *Calculation* стрічкового меню, відкрити перелік значень змінної ORIGIN (рис. 3.3).

У цьому переліку вибирається одне з двох значень, 0 або 1, що визначатиме значення ORIGIN для цілого MathCad-документа.

Треба відзначити, що перший спосіб має вищий пріоритет: за допомогою оператора присвоєння можна змінити значення ORIGIN у довільному місці MathCad-документа.

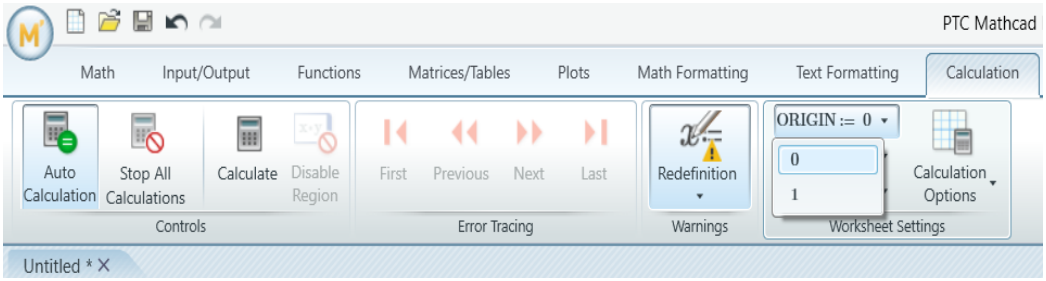


Рис. 3.3. Зміна значення системної змінної ORIGIN за допомогою вкладки *Calculation*

Наприклад, сформуємо два вектори X та Y (рис. 3.4).

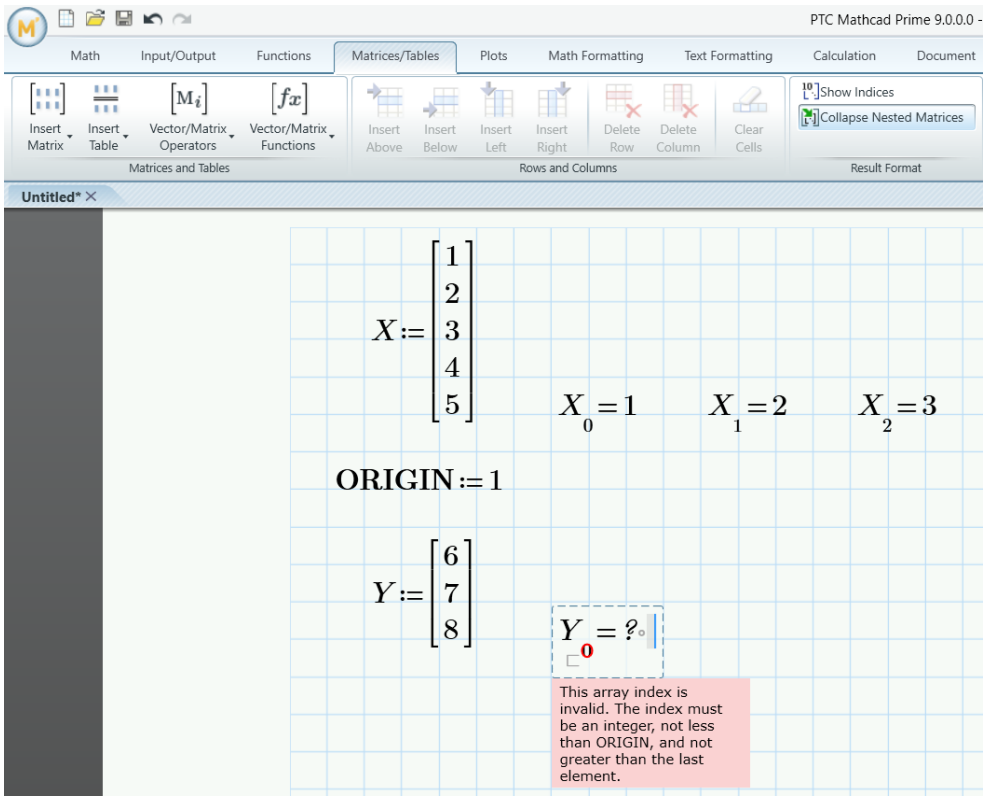


Рис. 3.4. Зміна значення системної змінної ORIGIN

Як видно з рисунку, звертання до елементу масиву Y_0 при зміні значення системної змінної ORIGIN створює ситуацію, коли компілятор генерує помилку. У цьому випадку нумерація елементів масиву Y починається з одиниці.

Спосіб 2. Створення масиву за допомогою ранжованої змінної та функції генерування випадкових чисел *rnd*.

Функція MathCad *rnd(x)* генерує випадкове число з рівномірним розподілом в інтервалі від 0 до *x*. При кожному новому обчисленні

або оновленні MathCad-документа функція *rnd(x)* генеруватиме нову послідовність випадкових чисел, *x* – додатне число, яке визначає верхню границю діапазону випадкових чисел. Нижня границя завжди дорівнює 0.

Рівномірний розподіл означає, що довільне число у вказаному діапазоні має однакову ймовірність бути обраним.

Щоб отримати випадкове число в іншому діапазоні, наприклад, від *a* до *b*, можна використовувати таку залежність: *a + rnd(b - a)*. Наприклад, щоб отримати випадкове число в діапазоні від 5 до 15: *5 + rnd(10)*.

Наприклад, створити вектор *X* із 7 елементів (випадкових чисел). Для формування вектора використати функцію *rnd(x)*. Створити вектор *Y* із елементів вектора *X*, заокруглених до цілого значення. Для формування векторів *X* та *Y* використано ранжовану змінну *i*. Процес розв'язання задачі подано на рис. 3.5.

$$i := 1,2..7 \qquad n := 10$$

$$X_i := \mathrm{rnd}(n) \qquad Y_i := \mathrm{round}(X_i)$$

$$X = \begin{bmatrix} 0 \\ 0.013 \\ 1.933 \\ 5.85 \\ 3.503 \\ 8.228 \\ 1.741 \\ 7.105 \end{bmatrix} \qquad Y = \begin{bmatrix} 0 \\ 0 \\ 2 \\ 6 \\ 4 \\ 8 \\ 2 \\ 7 \end{bmatrix}$$

Рис. 3.5. Використання ранжованої змінної та функції *rnd()* для формування вектора

Спосіб 3 – за допомогою ранжованої змінної за заданим законом формування значень.

Наприклад, створити вектор $X = (X_1, X_2, X_3, X_4)$. Елементи вектора сформувати заданим законом: $X_i = i^2 + 3i - 5$. Процес розв'язання задачі подано на рис. 3.6.

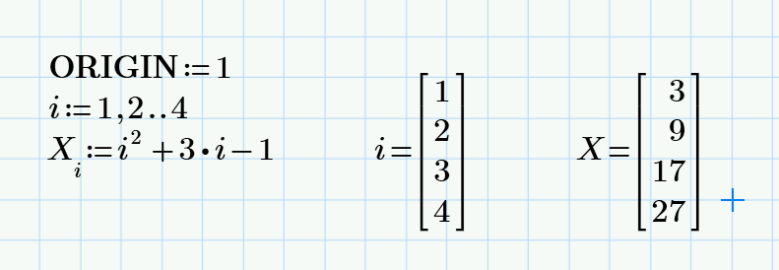


Рис. 3.6. Створення масиву за допомогою ранжованої змінної за заданим законом формування значень

Формувати вектори можна, означуючи значення окремих елементів, наприклад (рис. 3.7):

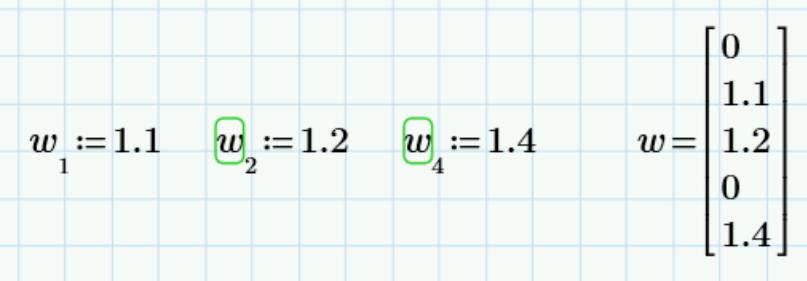


Рис. 3.7. Приклад формування вектора

3.2. Формування матриць

У пакеті MathCad передбачено чотири способи формування матриць:

1. Заповнення шаблону матриці, яка містить порожні місця введення чисел, що підходить для введення невеликих масивів (не більше 600 елементів).

2. Використання дискретної змінної. Цей метод ефективний, коли є закон для обчислення елементів матриці або коли у матриці більше, ніж 600 елементів.

3. Формування нових матриць із наявних. За допомогою вмонтованих функцій можна виокремлювати з матриць фрагменти, їх об’єднувати між собою та з іншими матрицями.

4. Читання даних з файлів – ще одна дуже важлива можливість MathCad: запис числових даних у файл і зчитування їх з файлу.

Спосіб 1. З використанням шаблону. Вкладка *Matrices/Tables*, кнопка *Insert Matrix*. У шаблоні, який появиться, відзначаємо мишкою необхідну конфігурацію (кількість рядків та кількість стовпців) масиву (рис. 3.8).

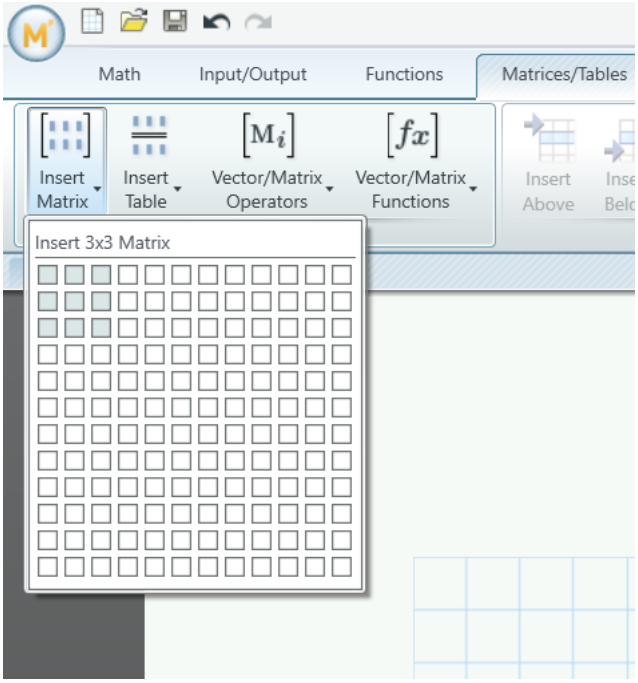


Рис. 3.8. Створення матриці з використанням шаблону вкладки *Matrices/Tables*

Зауважимо, що, використовуючи шаблон *Matrices/Tables*, кнопкою *Insert Matrix* можна створити матрицю з одного числа (розмірністю 1x1).

Нумерація елементів матриць за замовчуванням починається з 0 (нуля)!

Доступ до довільного елементу матриці можна отримати, задавши ім’я (ідентифікатор) матриці з двома індексами. Перший індекс позначає номер рядка, другий – номер стовпця.

Приклади створених матриць і доступу до їх елементів подано на рис. 3.9.

$$A1 := \begin{bmatrix} 3.5 & 2.87 & -0.2 \\ 2.87 & 0.8 & -0.3 \\ -7.9 & -7.73 & 5.8 \\ 0.166 & 0.213 & 0 \end{bmatrix} \quad B1 := \begin{bmatrix} 5.5 & 5.3 & -0.02 \\ 4.8 & 0 & 1.36 \\ 4.1 & -4.1 & 4.14 \\ -1.9 & -7.73 & -0.2 \end{bmatrix} \quad A1_{1,2} = -0.3 \quad B1_{0,1} = 5.3$$

Рис. 3.9. Приклади створення матриць за шаблоном та звертання до елементів матриць

Спосіб 2. Формування матриць за допомогою дискретних змінних (рис. 3.10), а також функції *matrix(M, N, f)* – створення матриці розміру $N \times M$ (рис. 3.11). У першому випадку для цього необхідно:

- визначити кількість рядків і стовпців (на рис. 3.10, а це, відповідно, *n:=500* і *m:=250);*
- задати зміну індексів матриці (*i:=0..n*, *j:=0..m*);
- записати зміну аргументів від індексів та функції з індексами (імені матриці), записати закон формування значень елементів;
- візуалізувати отриману матрицю. Для цього записати ім'я матриці без індексів та знак = (дорівнює).

ORIGIN := 1

$f =$

	1	2	3	4	5	6	7	8	9	10	11	12	… 250
1	0.435	0.644	0.813	0.932	0.993	0.992	0.929	0.808	0.638	0.427	0.19	−0.058	
2	0.605	0.783	0.913	0.985	0.997	0.946	0.837	0.675	0.472	0.239	−0.008	−0.256	
3	0.751	0.891	0.976	1	0.961	0.863	0.711	0.516	0.287	0.042	−0.207	−0.443	
4	0.867	0.964	1	0.974	0.887	0.746	0.558	0.335	0.091	−0.158	−0.397	−0.612	
5	0.949	0.997	0.984	0.909	0.778	0.598	0.382	0.141	−0.108	−0.351	−0.572	−0.757	
6	0.993	0.992	0.929	0.808	0.638	0.427	0.19	−0.058	−0.304	−0.53	−0.723	−0.872	
7	0.997	0.946	0.837	0.675	0.472	0.239	−0.008	−0.256	−0.487	−0.688	−0.846	−0.952	
8	0.961	0.863	0.711	0.516	0.287	0.042	−0.207	−0.443	−0.651	−0.818	−0.935	−0.994	
9	0.887	0.746	0.558	0.335	0.091	−0.158	−0.397	−0.612	−0.789	−0.916	−0.987	−0.996	
10	0.778	0.598	0.382	0.141	−0.108	−0.351	−0.572	−0.757	−0.895	−0.978	−0.999	−0.959	
11	0.638	0.427	0.19	−0.058	−0.304	−0.53	−0.723	−0.872	−0.966	−1	−0.972	−0.883	
12	0.472	0.239	−0.008	−0.256	−0.487	−0.688	−0.846	−0.952	−0.998	−0.982	−0.906	−0.773	
⋮ 500													⋱

Рис. 3.10. Створення матриці за допомогою дискретної змінної

У другому випадку з використанням функції *matrix(M, N, f)* кожний елемент, який розташований на перетині *i*-го рядка та *j*-го стовпця є *f1(i, j)*, *M* – кількість рядків, *N* – кількість стовпців, *f1(i, j)* – функція користувача.

$$f1(i,j) := 3 \cdot i + \sqrt{3} \cdot j \qquad A := \text{matrix}(2,3,f1) = \begin{bmatrix} 0 & 1.732 & 3.464 \\ 3 & 4.732 & 6.464 \end{bmatrix}$$

Рис. 3.11. Формування матриці за допомогою функції *matrix*

Можна змінювати розмір матриці, вставляючи і видаляючи рядки та стовпці (рис. 3.12). Для вставляння або вилучення рядків/стовпців потрібно:

- вказати мишкою елемент матриці, від якого праворуч та (або) нижче треба вставити стовпці та (або) рядки, відповідно;

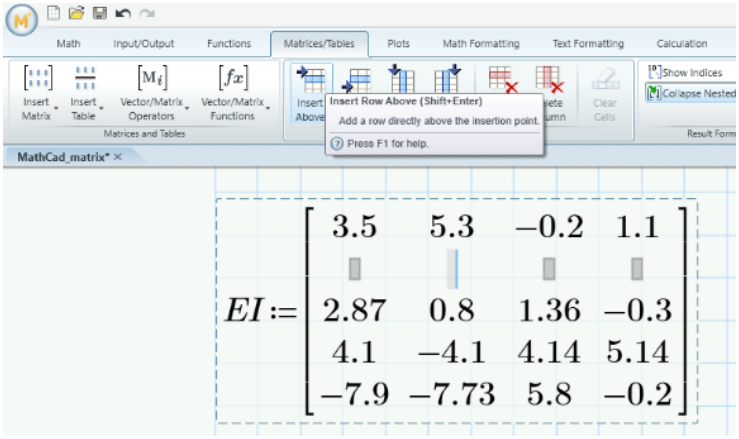


Рис. 3.12. Зміна розміру матриці (вставлення рядка вище від виокремленого елементу)

- використовуючи шаблон *Matrices/Tables → Insert Above/(Insert Below)* вставити рядок вище/(нижче) від виокремленого елементу або *Matrices/Tablesм → Insert Left/(Insert Right)* вставити стовпець ліворуч/(праворуч) від виокремленого елементу, відповідно. Також на цій панелі можна скасувати обрану функцію (рис. 3.12).

Створити нову матрицю можна також об'єднанням існуючих матриць. На відміну від функції APPENDPRN, яка дозволяє об'єднувати (дописувати) матриці у файлі, функції *stack* і *augment* (рис. 3.13) об'єднують дві матриці без створення файлу. Формат функцій:

- *stack* (*A*, *B*) об'єднує матриці одну над одною. Матриці *A* і *B* повинні мати однакову кількість стовпців;
- *augment* (*A*, *B*) об'єднує матриці *A* і *B* пліч-о-пліч. Матриці *A* і *B* повинні мати однакову кількість рядків.

Ще один зі способів створення нової матриці – копіювання частини існуючої матриці у нову. Функція *submatrix* (*A, irows*, *jrows*, *icols*, *jcols*) створює

матрицю, вирізану з матриці *A*. Нова матриця містить елементи матриці *A*, вирізані від ряду *irows* до ряду *jrows*, від стовпця *icols* до стовпця *jcols*.

Для формування спеціальних матриць застосовують функції *identity* та *diag*: *identity (N)* – створення одиничної матриці розміру *N*×*N*, *diag (V)* – створення діагональної матриці, на діагоналі якої розташовані елементи заданого вектора *V* (рис. 3.14).

Приклади використання функцій *stack*, *augment* і *submatrix* подано на рис. 3.13.

$$\text{submatrix}(C,1,3,1,2)=\begin{bmatrix}1.2 & 2.3\\ 0.28 & 0\\ 1.1 & -4.1\end{bmatrix}$$

$$A:=\begin{bmatrix}1.5 & 5.3 & -3.2 & 1.5\\ 0.87 & 0 & 0.36 & -0.2\\ 3.1 & -4.1 & 4.14 & 5.14\\ -1.98 & -47.73 & 0.8 & -0.25\end{bmatrix}\qquad \text{stack}(C,A)=\begin{bmatrix}1.2 & 2.3 & -3.2 & 1.5\\ 0.28 & 0 & 0.36 & -4.2\\ 1.1 & -4.1 & 0.14 & 3.14\\ 0.98 & -2.73 & 0.1 & 0.25\\ 1.5 & 5.3 & -3.2 & 1.5\\ 0.87 & 0 & 0.36 & -0.2\\ 3.1 & -4.1 & 4.14 & 5.14\\ -1.98 & -47.73 & 0.8 & -0.25\end{bmatrix}$$

$$B:=\text{augment}(C,A)=\begin{bmatrix}1.2 & 2.3 & -3.2 & 1.5 & 1.5 & 5.3 & -3.2 & 1.5\\ 0.28 & 0 & 0.36 & -4.2 & 0.87 & 0 & 0.36 & -0.2\\ 1.1 & -4.1 & 0.14 & 3.14 & 3.1 & -4.1 & 4.14 & 5.14\\ 0.98 & -2.73 & 0.1 & 0.25 & -1.98 & -47.73 & 0.8 & -0.25\end{bmatrix}$$

Рис. 3.13. Використання функцій *stack*, *augment* і *submatrix*

$$C:=\text{identity}(4)=\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{bmatrix}\qquad S:=\begin{bmatrix}4\\5\\6\\7\end{bmatrix}\qquad T:=\text{diag}(S)\quad T=\begin{bmatrix}4&0&0&0\\0&5&0&0\\0&0&6&0\\0&0&0&7\end{bmatrix}$$

Рис. 3.14. Формування матриць з використанням функцій *identity* та *diag*

3.3. Визначення параметрів матриць

У MathCad є вмонтовані функції для визначення параметрів матриці у вкладці *Matrices/Tables→Vector/Matrix Functions* (або набрати з клавіатури):

- *rows* (*M*) – кількість рядків у матриці або векторі;
- *cols* (*M*) – кількість стовпців у матриці;

- *max*(*M*)/*min* (*M*) – максимальне і мінімальне значення елементів у матриці;
- сума елементів вектора, обчислюється натисненням кнопки $\sum V$;
- *tr* (*M*) – сума діагональних елементів квадратної матриці, яка називається слідом матриці, де *M* – ім'я матриці.
- *last* (*M*) – індекс останнього елемента у векторі.

Приклад використання функцій визначення параметрів матриць подано на рис 3.15.

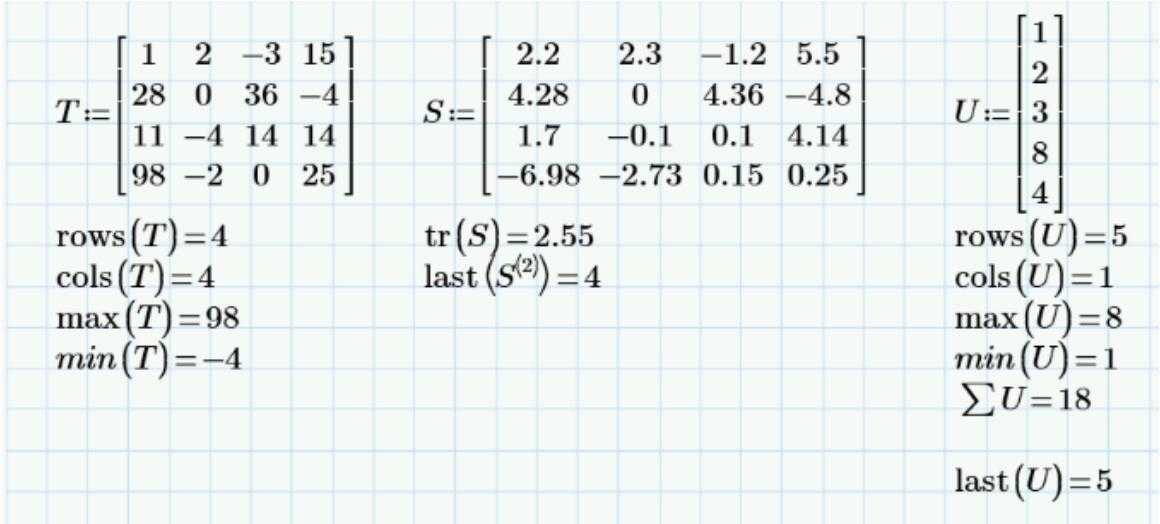


Рис. 3.15. Функції визначення параметрів матриць

3.4. Сортування елементів масивів

У MathCad є декілька вмонтованих функцій для сортування елементів матриці у порядку зростання або спадання значень елементів:

- *sort(v)* – сортування елементів вектора за зростанням;
- *reverse(v)* – перестановка елементів вектора у зворотному порядку;
- *csort(M, j)* – перестановка рядків матриці *M* за зростанням елементів *j*-го стовпця;
- *rsort(M, i)* – перестановка стовпців матриці *M* за зростанням елементів *i*-го рядка;
- *mean(M)* – обчислює середнє значення елементів матриці (вектора) *M*;

- *median(M)* – обчислює медіану матриці (вектора) *M*.

Приклад використання функцій сортування елементів матриць подано на рис 3.16.

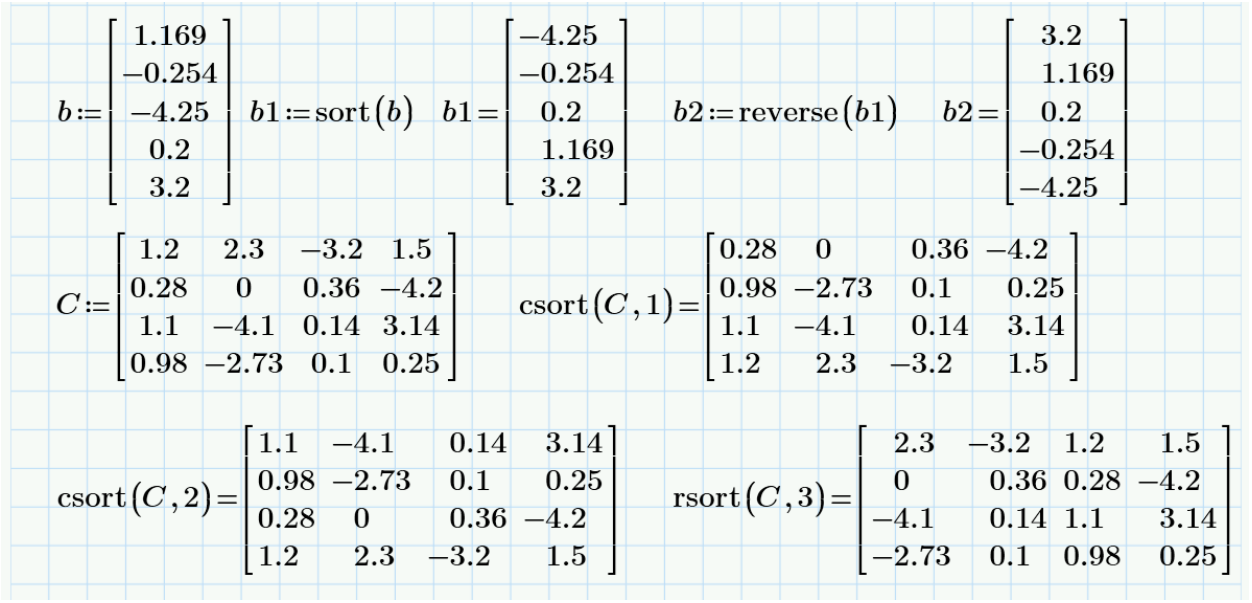


Рис. 3.16. Використання функцій сортування елементів матриць

3.5. Математичні перетворення над матрицями

Транспонування матриці. Транспонуванням називається операція, в результаті якої стовпці початкової матриці стають рядками, а рядки – стовпцями.

Для отримання транспонованої матриці необхідно у вкладці *Matrices/Tables* → *Vector/Matrix Functions* вибрати оператор M^T і у шаблоні ввести ім’я матриці.

При наборі символу *T* у верхньому індексі після імені матриці з клавіатури компілятор згенерує код помилки (рис. 3.17).

При введенні великих векторів з міркувань економії місця зручно вводити їх у вигляді рядка з подальшим транспонуванням.

Всі матричні оператори і матричні функції працюють з векторами тільки у вигляді стовпця, але не з рядками, тому рядки спочатку доводиться транспонувати у стовпець, а після виконання потрібної операції знов транспонувати у рядок.

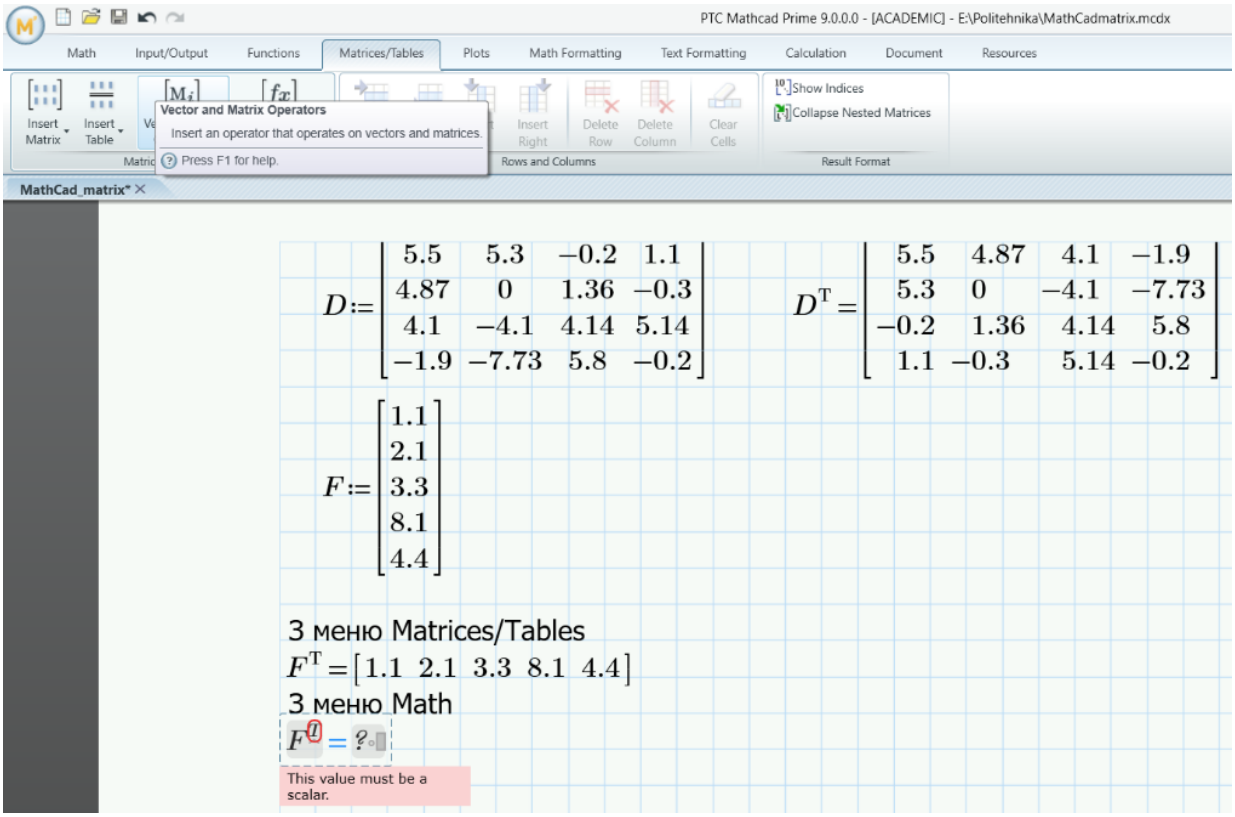


Рис. 3.17. Транспонування матриці з використанням вкладки *Matrices/Tables*

Обчислення визначника матриці та оберненої матриці. У MathCad функція обчислення визначника матриці розташована у вкладці *Matrices/Tables* → *Vector/Matrix Functions* функція ||x||.

Для обчислення визначника заданої матриці у вкладці *Matrices/Tables* → *Vector/Matrix Functions* обрати шаблон функції ||x|| та у шаблоні ввести ім'я матриці (рис. 3.18).

Отримати обернену матрицю можна через вкладку *Math* → *Operaqtors* і обрати шаблон функції x^{y}. Задана матриця обов'язково повинна бути квадратною!

Цей процес продемонстровано на рис. 3.18.

Добутком прямої матриці на обернену є одинична матриця.

Іноді одинична матриця необхідна для розв'язання матричних рівнянь. Для створення одиничної матриці у пакеті MathCad є вмонтована функція *identity(n)*, де *n* – порядок квадратної матриці.

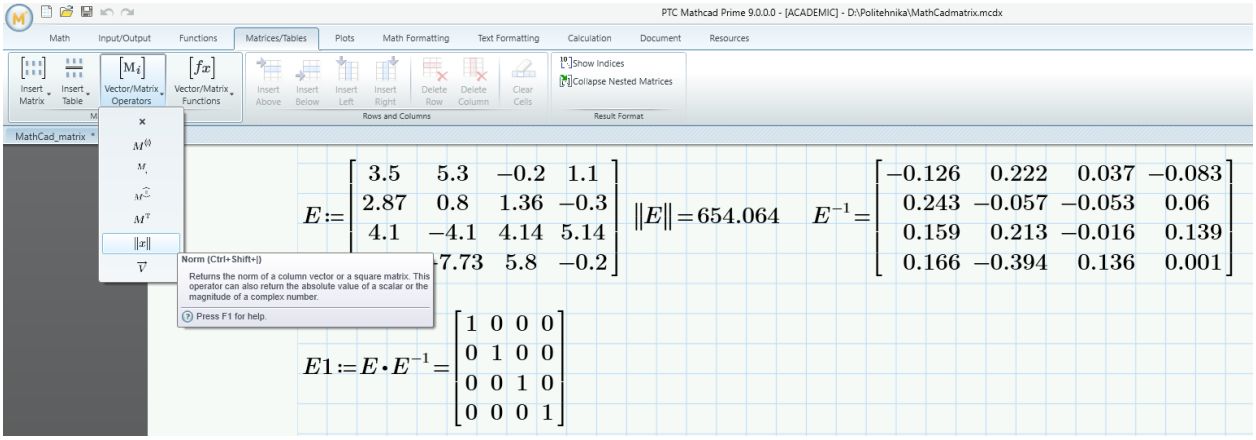


Рис. 3.18. Обчислення визначника та оберненої матриці

Арифметичні дії над матрицями. Додавання, віднімання і множення матриць виконується за допомогою традиційних операторів плюс "+", мінус "–" і знаку множення "*" (рис. 3.19).

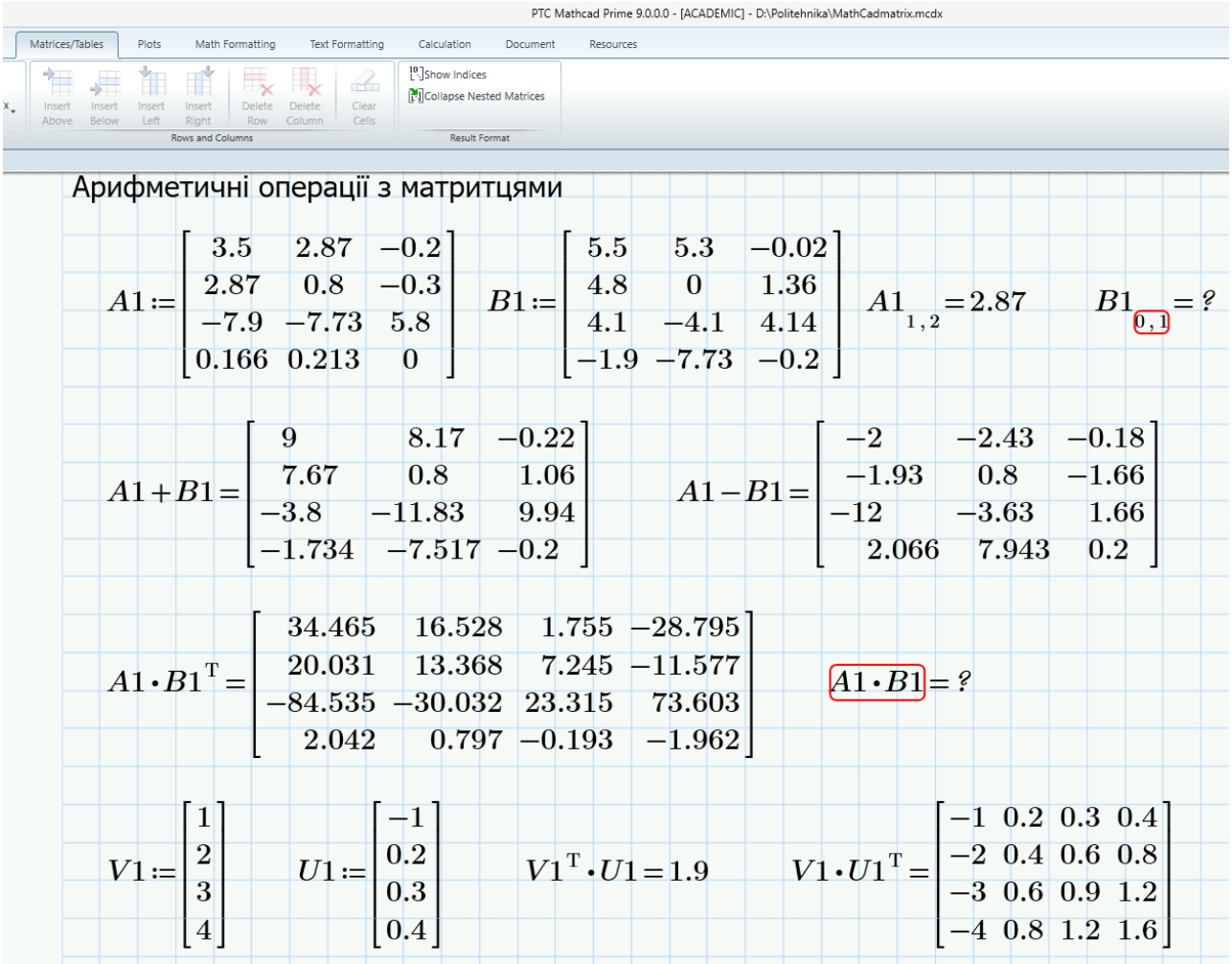


Рис. 3.19. Арифметичні операції з матрицями

У поданому прикладі звертання до елементу $B1_{0,1}$ матриці *B1* приводить до генерування помилки компілятором, оскільки значення системної змінної ORIGIN = 1.

Ще одна помилка генерується у результаті спроби множення матриць *A1* та *B1*, оскільки порушено правило множення матриць.

За правилом множення матриць, кількість стовпців першої матриці повинна бути рівною кількості рядків другої матриці:

$A(m \times n) \bullet B(n \times k) = C(m \times k)$ (у дужках вказані розміри матриць).

При додаванні і відніманні матриць ці дії здійснюються поелементно (додаються/віднімаються відповідні елементи обох матриць). Це вимагає, щоб матриці були однакового розміру (кількість рядків і кількість стовпців у матрицях були однакові).

У результаті множення отримаємо нову матрицю, яка матиме таку кількість рядків, як у першої матриці і таку кількість стовпців, як у другої матриці.

УВАГА! Від перестановки матриць місцями добуток ЗМІНЮЄТЬСЯ!

ПРИМІТКА. Корисно запам’ятати, що добуток рядка на стовпець дає число, а добуток стовпця на рядок дає квадратну матрицю (рис. 3.19).

Знак множення у MathCad-документі за замовчуванням позначається крапкою.

Обчислення матричного виразу. Розглянемо такий приклад.

Обчислити матричний вираз:

$$(2A+B)-B\cdot(A^T- B^T)-2A\cdot E$$

де A, B і E матриці 4-го порядку;

E – одинична матриця,;

A^T і B^T – транспоновані матриці.

Матриці A і B задаються у такому вигляді:

$$A = \begin{pmatrix} 5.16 & -1.25 & 0.52 & -1.24 \\ 0.35 & 6.72 & 4.82 & -0.65 \\ -1.22 & 2.35 & -2.54 & 2.63 \\ 3.71 & 6.12 & 2.35 & -0.76 \end{pmatrix}, \quad B = \begin{pmatrix} 4.51 & 0.00 & 7.05 & -8.25 \\ 3.25 & -0.26 & 1.25 & 2.15 \\ 0.35 & 0.17 & 0.00 & 7.35 \\ 2.55 & 1.71 & 0.23 & -3.56 \end{pmatrix}.$$

На рис. 3.20 подано вигляд MathCad-документу процесу розв'язання задачі. Матриці *A* та *B* задаються традиційним способом, а одинична матриця *E* формується з використанням функції *identity*. Очевидно, що використовуючи інші програмні застосунки нам би довелося витратити набагато більше часу.

$$(2A+B)-B\cdot(A^T-B^T)-2A\cdot E$$

$$A := \begin{bmatrix} 5.16 & -1.25 & 0.52 & -1.24 \\ 0.35 & 6.723 & 4.82 & -0.65 \\ -1.22 & 2.35 & -2.54 & 2.63 \\ 3.71 & 6.12 & 2.35 & -0.76 \end{bmatrix} \quad B := \begin{bmatrix} 4.51 & 0.0 & 7.05 & -8.25 \\ 3.25 & -0.26 & 1.25 & 2.15 \\ 0.35 & 0.17 & 0.0 & 7.35 \\ 2.55 & 1.71 & 0.23 & -3.56 \end{bmatrix}$$

$$E := \text{identity}(4) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T := (2\cdot A + B) - B\cdot(A^{\mathrm{T}} - B^{\mathrm{T}}) - 2\cdot A\cdot E = \begin{bmatrix} 105.448 & -35.19 & -6.902 & -5.328 \\ -6.097 & 12.538 & 20.242 & -9.143 \\ -51.189 & 20.578 & 34.871 & -14.386 \\ 29.488 & -13.625 & -15.713 & -4.579 \end{bmatrix}$$

Рис. 3.20. Вигляд MathCad-документу обчислення матричного виразу

Для прикладу подамо проєкт програмного коду обчислення матричного виразу $S = 2AB + 3(AB^T - EA)$ мовою програмування С++.

A, *B* – цілочислові матриці п'ятого порядку. Для формування матриць *A* та *B* використано стандартну функцію *rand()*. *E* – одинична матриця п'ятого порядку. Обчислення суми, різниці, добутку, транспонування матриць оформити у вигляді відповідних функцій.

Виконання розрахунків передбачає формування проміжних матриць, яким присвоєно такі символічні імена:

*A2 – 2*A;*

*AB2 – 2*A*B;*

BT – B^T;

*ABT – A*B^T;*

*EA – E*A;*

ABTEA – (AB^T - EA);

ABTEA3 – 3(AB^T - EA).

Приклад проєкту програмного коду реалізування задачі.

```
#include<iostream>
using namespace std;
int const m = 5;
void DobMatr(int X[][m], int Y[][m], int Z[][m]);
void kMatr(int k, int X[][m], int Y[][m]);
void TranspMatr(int X[][m], int XT[][m]);
void SumaMatr(int X[][m], int Y[][m], int Z[][m]);
void RiznMatr(int X[][m], int Y[][m], int Z[][m]);
int main()
{
    int A[m][m], B[m][m], E[m][m] = {};
    int A2[m][m], BT[m][m], EA[m][m],
        ABT[m][m], AB2[m][m], ABTEA[m][m],
        S[m][m], ABTEA3[m][m];
    cout << "*****************" << endl;
    cout << "Matrix A : " << endl;
    for (int i = 0; i < m; i++)
    {
        for (int j = 0; j < m; j++)
        {
            A[i][j] = rand() % 5 - 2;
            cout << A[i][j] << " ";
        }
        cout << endl;
    }
```

```
cout << "*****************" << endl;
cout << "Matrix B : " << endl;
for (int i = 0; i < m; i++)
{
    for (int j = 0; j < m; j++)
    {
        B[i][j] = rand() % 6 - 3;
        cout << B[i][j] << " ";
    }
    cout << endl;
}
for (int i = 0; i < m; i++)
    E[i][i] = 1;
cout << "*****************" << endl;
cout << "Matrix E : " << endl;
for (int i = 0; i < m; i++)
{
    for (int j = 0; j < m; j++)
        cout << E[i][j] << "\t";
    cout << endl;
}
kMatr(2, A, A2);
DobMatr(A2, B, AB2);
TranspMatr(B, BT);
DobMatr(A, BT, ABT);
DobMatr(A, E, EA);
RiznMatr(ABT, EA, ABTEA);
kMatr(3, ABTEA, ABTEA3);
SumaMatr(AB2, ABTEA3, S);
cout << "*****************" << endl;
```

```
    cout << "\nMatrix S:" << endl;
    for (int i = 0; i < m; i++)
    {
        for (int j = 0; j < m; j++)
            cout << S[i][j] << "\t";
        cout << endl;
    }
    system("pause");
    return 0;
}

void DobMatr(int X[][m], int Y[][m], int Z[][m])
{
    for (int i = 0; i < m; i++)
        for (int j = 0; j < m; j++)
        {
            Z[i][j] = 0;
            for (int k = 0; k < m; k++)
                Z[i][j] += Y[i][k] * X[k][j];
        }
}

void kMatr(int k, int X[][m], int Y[][m])
{
    for (int i = 0; i < m; i++)
        for (int j = 0; j < m; j++)
            Y[i][j] = k * X[i][j];
}

void SumaMatr(int X[][m], int Y[][m], int Z[][m])
```

```
{
    for (int i = 0; i < m; i++)
        for (int j = 0; j < m; j++)
            Z[i][j] = X[i][j] + Y[i][j];
}

void RiznMatr(int X[][m], int Y[][m], int Z[][m])
{
    for (int i = 0; i < m; i++)
        for (int j = 0; j < m; j++)
            Z[i][j] = X[i][j] - Y[i][j];
}

void TranspMatr(int X[][m], int XT[][m])
{
    for (int i = 0; i < m; i++)
        for (int j = 0; j < m; j++)
            XT[i][j] = X[j][i];
}
```

Результат виконання програмного коду:

```
*****************
Matrix A :
-1 0 2 -2 2
2 1 1 0 2
-2 -2 -1 0 -1
-1 -2 0 0 -1
-1 2 0 1 0
*****************
Matrix B :
1 0 -1 -1 2
```

```
2 -3 2 -3 0
1 2 -2 -2 -3
2 0 -1 0 0
-1 0 -2 2 1
*****************
Matrix E :
1       0       0       0       0
0       1       0       0       0
0       0       1       0       0
0       0       0       1       0
0       0       0       0       1
*****************

Matrix S:
3       -2      9       -3      11
-54     -26     -20     -8      -32
66      26      21      -10     38
-6      25      1       4       -5
16      22      21      -6      17
```

Ми свідомо не подали результати проміжних розрахунків, оскільки це потребує багато місця. Понад те, розв'язуючи реальні задачі, рекомендуємо на етапі відлагодження і тестування програмного коду використовувати проміжне виведення для контролю достовірності результатів обчислень.

Індивідуальні завдання

Обчислити матричний вираз з використанням пакету MathCad Prime та розробити проєкт програмного коду мовою C++:

1. $(2A^T+B)\cdot A+(2B-A)-E\cdot A$
2. $(3A-2B)+(A+3B)\cdot B-B\cdot E$
3. $2(A-B)\cdot(A^2+B)-2A^T\cdot E$
11. $(2A-B)\cdot(3A-2B)-2A\cdot B+A^T\cdot E$
12. $A\cdot(A^T-B^2)-2(B^T+A)\cdot B+A\cdot E$
13. $(2A+B)-B\cdot(A^T-B)-2A\cdot E$

4. $(A-3B)\cdot 2A+(3A-B)\cdot B+A\cdot E$

5. $(A-B^2)+(2A+B)-(A+3B)-A^T\cdot E$

6. $(B^T+A)\cdot A-(3A^T+B)-A\cdot E$

7. $(A^T+B^T)\cdot A+(2B+3B)\cdot E-A\cdot B$

8. $2A^T-A\cdot B\cdot(B-3A)+E$

9. $(2A\cdot B)^T-(A-B)\cdot(B-3A)+E$

10. $B\cdot(A+2B)-3A^T\cdot B+(A\cdot E-B)$

14. $(A+3B)\cdot A^T+B\cdot(2A+5B)-3B\cdot E$

15. $A\cdot(A-3B)+B\cdot(B^T-A)+2A\cdot B$

16. $E\cdot(A+2B)-B\cdot(A+4B)+A\cdot B$

17. $(3A+B)\cdot E-(A-B)\cdot B+B\cdot A$

18. $(A^T+B)\cdot B+3B\cdot A-A\cdot E$

19. $A^2-(A+2B)+E\cdot A+2A\cdot B$

20. $3(A+2B)-(2A-3B)+2A^T\cdot B-B$,

21. $4A^T-A\cdot 3B\cdot(B-3A)+E$

22 $(A^T+3B)\cdot B+3B\cdot A-2A\cdot E$

23. $(2A^T+B^T)\cdot 2A+(2B+3B)\cdot E-A\cdot 2B$

24. $(3A+2B)\cdot E-(2A-B)\cdot B+B\cdot A$

25. $3A^T-2A\cdot B\cdot(3B-3A)+E$

26. $(5A^T+B)\cdot 2B+3B\cdot A-A\cdot E$

де A, B і E матриці 4-го порядку, причому E – одинична матриця, A^T і B^T – транспоновані матриці A і B. Матриці A і B задаються у такому вигляді:

$$A=\begin{pmatrix} 5.16 & -1.25 & 0.52 & -1.24 \\ 0.35 & 6.72 & 4.82 & -0.65 \\ -1.22 & 2.35 & -2.54 & 2.63 \\ 3.71 & 6.12 & 2.35 & -0.76 \end{pmatrix}, \quad B=\begin{pmatrix} 4.51 & 0.00 & 7.05 & -8.25 \\ 3.25 & -0.26 & 1.25 & 2.15 \\ 0.35 & 0.17 & 0.00 & 7.35 \\ 2.55 & 1.71 & 0.23 & -3.56 \end{pmatrix}.$$

Для обчислення матричного виразу розробити підпрограми суми, різниці та добутку матриць.

РОЗДІЛ 4
РОЗВ'ЯЗУВАННЯ СИСТЕМ ЛІНІЙНИХ АЛГЕБРИЧНИХ РІВНЯНЬ (СЛАР)

Існує декілька основних методів розв'язування СЛАР (система лінійних алгебричних рівнянь), які поділяються на прямі та ітераційні.

Основні методи розв'язування СЛАР.

Прямі методи, які дають точний розв'язок за кінцеву кількість кроків (якщо він існує). До них віднесемо:

- метод Гауса (або метод послідовного виключення невідомих) – найпоширеніший алгоритм, який приводить матрицю коефіцієнтів системи до трикутного вигляду за допомогою елементарних перетворень рядків, а потім знаходить розв'язок зворотним ходом;
- метод Крамера (або метод визначників) – підходить для систем, де кількість рівнянь дорівнює кількості невідомих, і визначник основної матриці не дорівнює нулю. Розв'язок знаходиться через обчислення визначників матриць;
- метод оберненої матриці (матричний метод) – використовується, коли систему можна подати у вигляді $A \cdot X = B$, де A – квадратна невироджена матриця коефіцієнтів.

Ітераційні методи, які застосовуються, як правило, для розв'язування СЛАР великих порядків з розрідженими матрицями. Вони полягають у побудові послідовності наближень до точного розв'язку.

4.1. Метод оберненої матриці

Метод оберненої матриці (також відомий як матричний метод) – це числовий метод розв'язання систем лінійних алгебричних рівнянь шляхом використання оберненої матриці.

Розглянемо систему *n* лінійних рівнянь з *n* невідомими

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1; \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2; \\ a_{31}x_1 + a_{32}x_2 + \cdots + a_{3n}x_n = b_3;, \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n = b_n. \end{cases}$$

і визначник основної матриці не дорівнює нулю. Запишемо систему лінійних рівнянь у матричному вигляді $A \cdot X = B$,

де A – матриця коефіцієнтів, X – матриця-стовпець з невідомих змінних, B – матриця-стовпець з вільних членів.

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix} \quad X = \begin{pmatrix} x_1 \\ x_2 \\ \cdots \\ x_n \end{pmatrix} \quad B = \begin{pmatrix} b_1 \\ b_2 \\ \cdots \\ b_n \end{pmatrix}.$$

Якщо матриця A має обернену матрицю A^{-1}, то

$$A^{-1} \cdot A \cdot \mathrm{X} = A^{-1} \cdot B.$$

Оскільки $A^{-1} \cdot A = E$, *тому розв'язок системи визначається рівністю.*

$$\mathrm{X} = A^{-1} \cdot B.$$

Отже, щоб розв'язати систему рівнянь, достатньо обчислити матрицю, обернену до матриці системи A, і помножити її на матрицю з вільних членів справа.

Алгоритм розв'язування СЛАР методом оберненої матриці:

1. *Підготовка системи рівнянь*: записуємо систему лінійних рівнянь у матричному вигляді $A \cdot X = B$, де A – матриця коефіцієнтів, X – матриця-стовпець з невідомих змінних, B – матриця-стовпець з вільних членів.

2. *Перевірка визначеності матриці*: переконуємося, що матриця A має ненульовий визначник. Якщо визначник дорівнює нулю, обернена матриця не існує, і метод не може бути застосований.

3. *Обчислення оберненої матриці*: якщо визначник ненульовий, обчислюємо обернену матрицю A^{-1} для матриці A.

4. *Обчислення розв'язку*: розв'язок системи обчислюється як добуток оберненої матриці A^{-1} та стовпця вільних членів B.

4.2. Метод Гауса

Метод Гауса (також відомий як метод послідовного виключення невідомих) – найбільш потужний та універсальний інструмент для обчислення розв'язку довільної системи лінійних алгебричних рівнянь (СЛАР). Суть методу Гауса полягає у тому, щоб за допомогою елементарних перетворень привести задану систему до системи «сходинкового» трикутного вигляду.

Розглянемо систему *m* лінійних рівнянь з *n* невідомими

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1; \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2; \\ a_{31}x_1 + a_{32}x_2 + \cdots + a_{3n}x_n = b_3; \\ \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n = b_m. \end{cases},$$

Очевидно, що серед коефіцієнтів a_{i1} хоча б один відмінний від нуля. Якщо ж $a_{11} = 0$, тоді першим в системі запишемо те рівняння, в якому коефіцієнт при x_1 відмінний від нуля. Позначимо цей коефіцієнт через a'_{11}. Перетворимо систему, виключаючи x_1 в усіх рівняннях, крім першого. Для цього помножимо перше рівняння на $-\frac{a_{21}}{a'_{11}}$ і додаємо до другого, потім помножимо перше рівняння на $-\frac{a_{31}}{a'_{11}}$ і додаємо до третього і т.д. При цьому може статися так, що друге невідоме x_2також не входить в усі рівняння з номером $i > 1$. Якщо x_k невідоме з найменшим номером, яке входить у довільне рівняння, не рахуючи першого. Одержимо систему

$$\begin{cases} a'_{11}x_1 + \dots + a'_{1n}x_n = b'_1; \\ a'_{2k}x_k + \dots + a'_{2n}x_n = b'_2; \\ \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots \\ a'_{mk}x_k + \dots + a'_{mn}x_n = b'_n \end{cases},$$

$k > 1, a'_{11} \neq 0.$

Застосувавши до всіх рівнянь, окрім першого, таку саму процедуру і виконавши низку елементарних перетворень, одержимо систему

$$\begin{cases} a''_{11}x_1 + \ldots + a''_{1n}x_n = b''_1; \\ a''_{2k}x_k + \ldots + a''_{2n}x_n = b''_2; \\ a''_{3l}x_l + \ldots + a''_{3l}x_l = b''_3; \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ a''_{lk}x_l + \ldots + a''_{ln}x_n = b''_n \end{cases},$$

$a''_{11} \neq 0, a''_{2k} \neq 0.$

Якщо продовжити цей процес, тоді отримаємо систему

$$\begin{cases} \bar{a}_{11}x_1 + \ldots + \bar{a}_{1n}x_n = \bar{b}_1; \\ \bar{a}_{2k}x_k + \ldots + \bar{a}_{2n}x_n = \bar{b}_2; \\ a''_{3l}x_l + \ldots + a''_{3l}x_l = \bar{b}_3; \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ \bar{a}_{rs}x_l + \ldots + \bar{a}_{rs}x_n = \bar{b}_r; \\ 0 = \bar{b}_{r+1}; \\ \ldots \\ 0 = \bar{b}_m \end{cases},$$

Таку систему рівнянь називають *східчастою або трапецієподібною*.

Система східчастого вигляду вказує на два можливі випадки:

1. Якщо система містить рівняння вигляду $0 = b_t$ і $b_t \neq 0$, тоді вона очевидно несумісна.

2. Якщо система не містить рівнянь вигляду $0 = b_t$ (коли ліві частини відповідних рівнянь дорівнюють нулю), маємо тотожності, і ці нульові рядки можна відкинути.

Тоді матимемо один з двох випадків:

а) кількість рівнянь дорівнює кількості невідомих – система має єдиний розв’язок;

б) кількість рівнянь менша за кількості невідомих – система має безліч розв’язків.

4.3. Алгоритм розв’язування системи *n* лінійних рівнянь з *n* невідомими методом Гауса

Процес розв’язування СЛАР методом Гауса складається з двох етапів. На *першому етапі* (прямий хід) система рівнянь, за допомогою елементарних перетворень, зводиться до рівнозначної їй системи трикутної форми

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 \ +\ldots+a_{1n}x_n = b_1; \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 \ +\cdots+ a_{2n}x_n = b_{2:}, \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 \ +\cdots+ a_{3n}x_n = b_3; \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ a_{n1}x_1 + a_{n2}x_2 + a_{n3}x_3 +\ldots+a_{nn}x_n = b_n. \end{cases}$$

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 +\ldots+a_{1n}x_n = b_1; \\ \qquad c_{22}x_2 + c_{23}x_3 \ +\cdots+ c_{2n}x_n = d_2; \\ \qquad\qquad c_{33}x_3 +\cdots+ a_{3n}x_n = d_3; \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ \qquad\qquad\qquad\qquad a_{nn}x_n = d_n. \end{cases}$$

На *другому етапі* (обернений хід), з отриманої «сходинкової» системи, виконується послідовне обчислення значень невідомих величин.

Якщо кількість невідомих дорівнює кількості рівнянь, тоді система має єдиний розв'язок, якщо система не містить рівнянь вигляду $0 = d_n$, тоді вона має безліч розв'язків.

4.4. Розв'язування системи *n* лінійних алгебричних рівнянь з *n* невідомими

Обчислення розв'язку СЛАР виконаємо різними програмними засобами, з використанням пакету MathCad:

- з використанням блоку розв'язку *Solve Block*;
- з використання матричних операцій;
- з використанням внутрішньої функції *lsolve(A, B)*.

та розробленням проєкту програмного коду мовою C++.

$$\begin{cases} -0.87x_1 + 0.27x_2 - 0.22x_3 - 0.18x_4 = -1.21 \\ -0.21x_1 - x_2 - 0.45x_3 + 0.18x_4 = 0.33 \\ 0.12x_1 + 0.13x_2 - 1.33x_3 - 0.18x_4 = 0.48 \\ 0.33x_1 - 0.05x_2 + 0.06x_3 - 1.28x_4 = 0.17 \end{cases}$$

На рис 4.1 продано процес розв'язання СЛАР з використанням блоку розв'язку *Solve Block*.

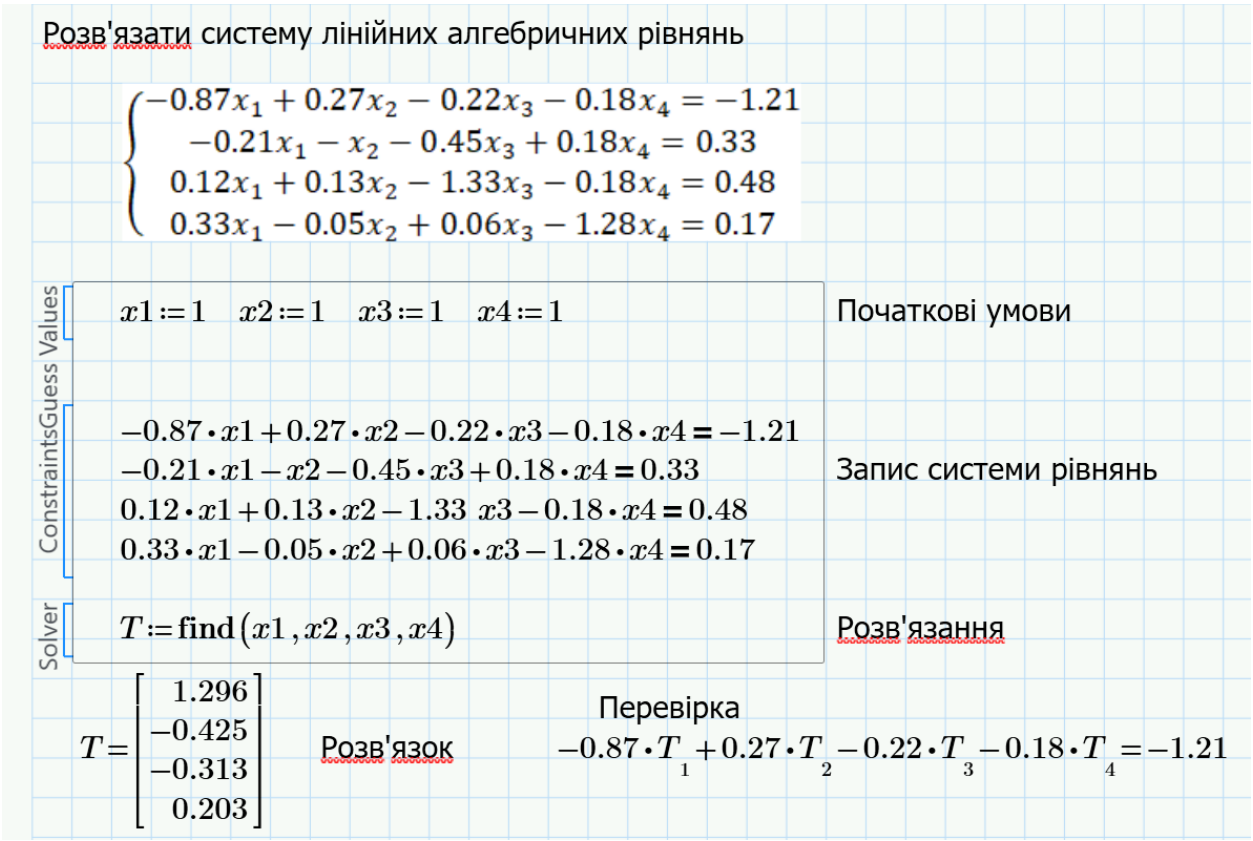


Рис. 4.1. Процес розв’язання СЛАР з використанням блоку розв’язку *Solve Block*.

Процес розв’язання СЛАР (умова завдання запозичена з попереднього прикладу) з використанням вмонтованої функції *lsolve(A, B)* та у матричній нотації подано на рис. 4.2 з відповідними поясненнями.

У поданих прикладах використані такі позначення:

- *A* – матриця коефіцієнтів при невідомих СЛАР;
- *B* – вектор вільних членів;
- *T* – вектор розв’язків СЛАР з використанням *Solve Block*;
- S – вектор розв’язків СЛАР з використанням вмонтованої функції *lsolve(A, B)*;
- *X* – вектор розв’язків СЛАР з використанням оберненої матриці.

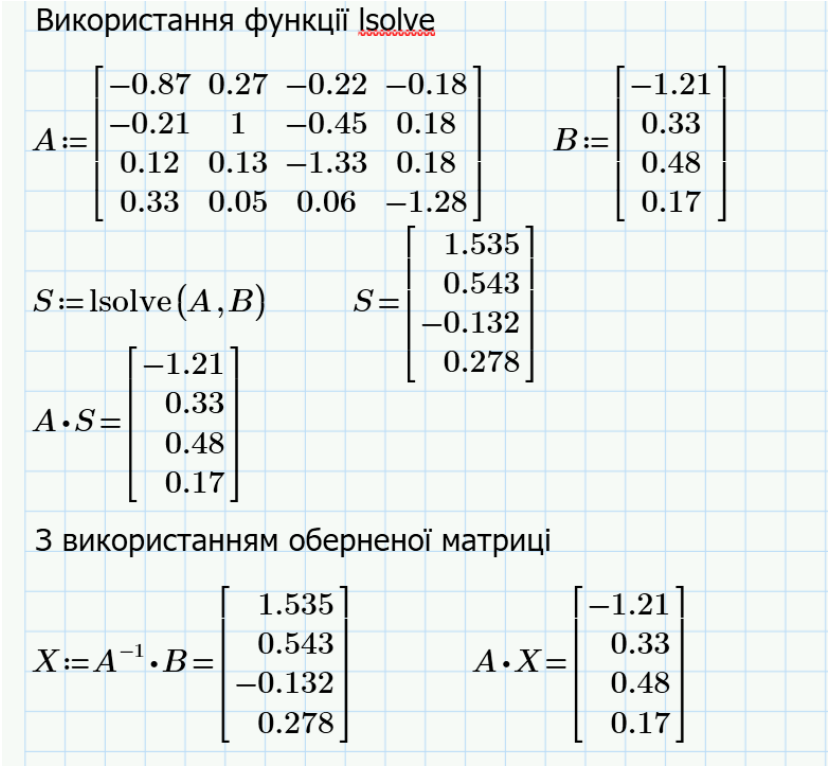


Рис. 4.2. Процес розв’язання СЛАР з використанням вмонтованої функції *lsolve(A, B)* та у матричній нотації

Далі подамо приклади проєктів програмних кодів для розв’язання СЛАР матричним способом та методом Гауса мовою С++.

Аналізуючи наступний проєкт програмного коду спробуйте пояснити наявність двовимірних масивів matrix та OBRM для перевірки коректності отриманих розв’язків.

Проєкт програмного коду для розв’язання СЛАР матричним способом:

```
#include <iostream>
#include <cmath>
using namespace std;
void Inverse(double **A, int N)
{
    double temp;
    double **E = new double* [N];
    for (int i = 0; i < N; i++)
```

```
    E[i] = new double[N];
for (int i = 0; i < N; i++)
    for (int j = 0; j < N; j++)
    {
        E[i][j] = 0.0;
        if (i == j)
            E[i][j] = 1.0;
    }
for (int k = 0; k < N; k++)
{
    temp = A[k][k];
    for (int j = 0; j < N; j++)
    {
        A[k][j] /= temp;
        E[k][j] /= temp;
    }
    for (int i = k + 1; i < N; i++)
    {
        temp = A[i][k];
        for (int j = 0; j < N; j++)
        {
            A[i][j] -= A[k][j] * temp;
            E[i][j] -= E[k][j] * temp;
        }
    }
}
for (int k = N - 1; k > 0; k--)
{
    for (int i = k - 1; i >= 0; i--)
    {
```

```
            temp = A[i][k];
            for (int j = 0; j < N; j++)
            {
                A[i][j] -= A[k][j] * temp;
                E[i][j] -= E[k][j] * temp;
            }
        }
    }
    for (int i = 0; i < N; i++)
        for (int j = 0; j < N; j++)
            A[i][j] = E[i][j];
    for (int i = 0; i < N; i++)
        delete[] E[i];
    delete[] E;
}
int main()
{
    int nr;
cout << "The order of the system of equations nr =: ";
    cin >> nr;
    double **matrix = new double *[nr];
    for (int i = 0; i < nr; i++)
        matrix[i] = new double [nr];
    double **OBRM = new double *[nr];
    for (int i = 0; i < nr; i++)
        OBRM[i] = new double [nr];
    double *B = new double [nr];
    double *X = new double [nr];
cout << "Enter matrix of coefficients and the vector of
free members: " << endl;
```

```
    for (int i = 0; i < nr; i++)
    {
        for (int j = 0; j < nr; j++)
        {
cout << "matrix[" << i+1 << "][" << j+1 << "] = ";
cin >> matrix[i][j];
            OBRM[i][j] = matrix[i][j];
        }
cout << "B[" << i << "]= ";
cin >> B[i];
    }
    cout << "The matrix of coefficients: " << endl;
    for (int i = 0; i < nr; i++)
    {
        for (int j = 0; j < nr; j++)
cout << matrix[i][j] << "\t";
cout << endl;
    }
cout << "The vector of free members: " << endl;
    for (int i = 0; i < nr; i++)
cout << "B[" << i+1 << "]=" << B[i] << " " << endl;
Inverse(OBRM, nr);
    cout << "Inverse matrix: " << endl;
    for (int i = 0; i < nr; i++)
    {
        for (int j = 0; j < nr; j++)
cout << OBRM[i][j] << "   \t";
cout << endl;
    }
    for (int i = 0; i < nr; i++)
```

```
    {
        X[i] = 0.0;
        for (int j = 0; j < nr; j++)
            X[i] += OBRM[i][j] * B[j];
    }
cout << "The solution vector: " << endl;
    for (int i = 0; i < nr; i++)
cout << "X[" << i + 1 << "]=" << X[i] << " " << endl;
//checking the correctness of the solutions
    for (int i = 0; i < nr; i++)
    {
        B[i] = 0.0;
        for (int j = 0; j < nr; j++)
            B[i] += matrix[i][j] * X[j];
    }
cout << "Vector of free members according to the result
of the correctness check: " << endl;
    for (int i = 0; i < nr; i++)
cout << "B[" << i + 1 << "]=" << B[i] << " " << endl;
//release of dynamic arrays:
    for (int i = 0; i < nr; i++)
    {
        delete[]matrix[i];
        delete[]OBRM[i];
    }
    delete []matrix;
    delete []OBRM;
    delete[]B;
    delete[]X;
    system("pause");
```

```
    return 0;
}
```

Результат виконання програмного коду:

```
The order of the system of equations nr =: 3
Enter matrix of coefficients and the vector of free
members:
matrix[1][1] = 1.7
matrix[1][2] = 10.
matrix[1][3] = 1.3
B[0]= 3.1
matrix[2][1] = 3.1
matrix[2][2] = 1.7
matrix[2][3] = 2.1
B[1]= 2.1
matrix[3][1] = 3.3
matrix[3][2] = 7.7
matrix[3][3] = 4.4
B[2]= 1.9
The matrix of coefficients:
1.7     10      1.3
3.1     1.7     2.1
3.3     7.7     4.4
The vector of free members:
B[1]=3.1
B[2]=2.1
B[3]=1.9
Inverse matrix:
0.14948         0.584674        -0.323213
0.115421        -0.0548723      -0.00791262
```

```
-0.314096       -0.342479       0.48353
The solution vector:
X[1]=1.0771
X[2]=0.227539
X[3]=-0.774198
Vector of free members according to the result of the
correctness check:
B[1]=3.1
B[2]=2.1
B[3]=1.9
```

Проєкт програмного коду для розв’язання СЛАР метод Гауса

```
#include <iostream>
#include <cmath>
using namespace std;
void SUSVAR(double** a, double* b, double* X, int n)
{
    int i, j, k;
    double c, s;
    for (k = 0; k < n-1; k++)
        for (i = k + 1; i < n; i++)
        {
c = a[i][k] / a[k][k];
a[i][k] = 0;
for (j = k + 1; j < n; j++)
    a[i][j] -= c * a[k][j];
b[i] -= c * b[k];
}
X[n - 1] = b[n - 1] / a[n - 1][n - 1];
for (i = n - 1; i >= 0; i--)
```

```
{
    s = 0;
    for (j = i + 1; j < n; j++)
    {
        s += a[i][j] * X[j];
        X[i] = (b[i] - s) / a[i][i];
    }
}
}
int main()
{
    int nr;
cout << "The order of the system of equations nr =: ";
cin >> nr;
    double** matrix = new double* [nr];
    for (int i = 0; i < nr; i++)
        matrix[i] = new double[nr];
    double** A = new double* [nr];
    for (int i = 0; i < nr; i++)
        A[i] = new double[nr];
    double* B = new double[nr];
    double* B1 = new double[nr];
    double* X = new double[nr];
cout << "Enter matrix of coefficients and the vector of
free members: " << endl;
    for (int i = 0; i < nr; i++)
    {
        for (int j = 0; j < nr; j++)
        {
cout << "matrix[" << i + 1 << "][" << j + 1 << "] = ";
```

```
cin >> matrix[i][j];
            A[i][j] = matrix[i][j];
        }
cout << "B[" << i+1 << "]= ";
cin >> B[i];
        B1[i] = B[i];
    }
cout << "The matrix of coefficients: " << endl;
    for (int i = 0; i < nr; i++)
    {
        for (int j = 0; j < nr; j++)
cout << matrix[i][j] << "\t";
cout << endl;
    }
    cout << "The vector of free members: " << endl;
    for (int i = 0; i < nr; i++)
cout << "B[" << i + 1 << "]=" << B[i] << " " << endl;
    SUSVAR(A, B1, X, nr);
cout << "The solution vector: " << endl;
    for (int i = 0; i < nr; i++)
cout << "X[" << i + 1 << "]=" << X[i] << " " << endl;
//checking the correctness of the solutions
    for (int i = 0; i < nr; i++)
    {
        B[i] = 0.0;
        for (int j = 0; j < nr; j++)
            B[i] += matrix[i][j] * X[j];
    }
cout << "Vector of free members according to the result
of the correctness check: " << endl;
```

```
    for (int i = 0; i < nr; i++)
cout << "B[" << i + 1 << "]=" << B[i] << " " << endl;
//release of dynamic arrays:
    for (int i = 0; i < nr; i++)
    {
        delete[]matrix[i];
        delete[]A[i];
    }
    delete[]matrix;
    delete[]A;
    delete[]B;
    delete[]B1;
    delete[]X;
    system("pause");
    return 0;
}
```

```
Enter matrix of coefficients and the vector of free
members:
matrix[1][1] = 1.7
matrix[1][2] = 10.
matrix[1][3] = 1.3
B[1]= 3.1
matrix[2][1] = 3.1
matrix[2][2] = 1.7
matrix[2][3] = 2.1
B[2]= 2.1
matrix[3][1] = 3.3
matrix[3][2] = 7.7
matrix[3][3] = 4.4
B[3]= 1.9
```

```
The matrix of coefficients:
1.7     10      1.3
3.1     1.7     2.1
3.3     7.7     4.4
The vector of free members:
B[1]=3.1
B[2]=2.1
B[3]=1.9
The solution vector:
X[1]=1.0771
X[2]=0.227539
X[3]=-0.774198
Vector of free members according to the result of the correctness check:
B[1]=3.1
B[2]=2.1
B[3]=1.9
```

Проєкт програмного коду для розв’язання СЛАР з використанням шаблонного класу (метод Гауса) мовою С++.

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

template <typename T>
class Darray
{
private:
    T **Matrix; // Матриця коефіцієнтів
    T *Vr;      // Вектор вільних членів
```

```
    T *Vn;      // Вектор невідомих
    int n;
public:
    Darray(int _n) : n(_n)
{
        Matrix = new T * [n];
        for (int i = 0; i < n; i++)
            Matrix[i] = new T[n]();
        Vr = new T[n]();
        Vn = new T[n]();
}
    void setDarray(T **inputMatrix)
{
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
                Matrix[i][j] = inputMatrix[i][j];
}
   T **getDarray() const
{
        return Matrix;
}
    void setOarray(T *inputVector)
{
        for (int i = 0; i < n; i++)
            Vr[i] = inputVector[i];
}
    T *getOarray() const
{
        return Vr;
}
```

```
    void print_Darray() const
{
cout << "Matrix: " << endl;
        for (int i = 0; i < n; i++)
{
            for (int j = 0; j < n; j++)
cout << Matrix[i][j] << "\t";
cout << endl;
}
        cout << "Vector Vr:" << endl;
        for (int i = 0; i < n; i++)
cout << Vr[i] << endl;
}
// Метод Гауса
    void Gelg()
{
        T **A = new T * [n];
        for (int i = 0; i < n; i++)
{
            A[i] = new T[n];
            for (int j = 0; j < n; j++)
                A[i][j] = Matrix[i][j];
}
        T *B = new T[n];
        for (int i = 0; i < n; i++)
            B[i] = Vr[i];
        for (int k = 0; k < n - 1; k++) {
            for (int i = k + 1; i < n; i++) {
                T c = A[i][k] / A[k][k];
                A[i][k] = 0;
```

```
                for (int j = k + 1; j < n; j++)
                    A[i][j] -= c * A[k][j];
                B[i] -= c * B[k];
            }
        }
        for (int i = n - 1; i >= 0; i--)
{
            T s = 0;
            for (int j = i + 1; j < n; j++)
                s += A[i][j] * Vn[j];
            Vn[i] = (B[i] - s) / A[i][i];
}
        for (int i = 0; i < n; i++)
            delete[] A[i];
        delete[] A;
        delete[] B;
}
   T *getGsolution() const
{
        return Vn;
}
    ~Darray() {
        for (int i = 0; i < n; i++)
            delete[] Matrix[i];
        delete[] Matrix;
        delete[] Vr;
        delete[] Vn;
}
};
```

```
int main()
{
    int n;
cout << "Enter the order of the system (n): ";
cin >> n;
    double **matrix = new double *[n];
    for (int i = 0; i < n; i++)
        matrix[i] = new double[n];
    double *vectorR = new double[n];
cout << "Enter the matrix of coefficients:" << endl;
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++) {
cout << "matrix[" << i + 1 << "][" << j + 1 << "] = ";
cin >> matrix[i][j];
}
cout << "Enter the vector of free members (Vr):" << endl;
    for (int i = 0; i < n; i++) {
cout << "Vr[" << i + 1 << "] = ";
cin >> vectorR[i];
}
    Darray<double> obj(n);
    obj.setDarray(matrix);
    obj.setOarray(vectorR);
    cout << "\nInitial system:\n";
    obj.print_Darray();
    obj.Gelg();
    double *solution = obj.getGsolution();
cout << "\nSolution vector Vn (Gauss method):\n";
    for (int i = 0; i < n; i++)
cout << "Vn[" << i + 1 << "] = " << solution[i] << endl;
```

```
    for (int i = 0; i < n; i++)
        delete[] matrix[i];
    delete[] matrix;
    delete[] vectorR;
    return 0;
}
```

Результати виконання програмного коду:

```
Enter the order of the system (n): 3
Enter the matrix of coefficients:
matrix[1][1] = 1.7
matrix[1][2] = 10
matrix[1][3] = 1.3
matrix[2][1] = 3.1
matrix[2][2] = 1.7
matrix[2][3] = 2.1
matrix[3][1] = 3.3
matrix[3][2] = 7.7
matrix[3][3] = 4.4
Enter the vector of free members (Vr):
Vr[1] = 3.1
Vr[2] = 2.1
Vr[3] = 1.9

Initial system:
Matrix:
1.7 10  1.3
3.1 1.7 2.1
3.3 7.7 4.4
Vector Vr:
```

```
3.1
2.1
1.9

Solution vector Vn (Gauss method):
Vn[1] = 1.0771
Vn[2] = 0.227539
Vn[3] = -0.774198



=== Code Execution Successful ===
```

Індивідуальні завдання

Розв’язати СЛАР у пакеті MathCad Prime:

- з використанням *Solve Block*;
- з використанням функції *lsolve*;
- матричним методом.

Розробити проєкти програмних кодів для розв’язання завдання мовою C++.

Виконати порівняльний аналіз отриманих результатів. Завдання подані у табл. 4.1.

Таблиця 4.1

Варіанти індивідуальних завдань

1. $\begin{cases} x_1 + x_2 + 2x_3 + 3x_4 = 1 \\ 3x_1 - x_2 - x_3 - 2x_4 = -4 \\ 2x_1 + 3x_2 - x_3 - x_4 = -6 \\ x_1 + 2x_2 + 3x_3 - x_4 = -4 \end{cases}$	2. $\begin{cases} x_1 + 2x_2 + 3x_3 - 2x_4 = 6 \\ x_1 - x_2 - 2x_3 - 3x_4 = 8 \\ 3x_1 + 2x_2 - x_3 + 2x_4 = 4 \\ 2x_1 - 3x_2 + 2x_3 + x_4 = -8 \end{cases}$

3. $\begin{cases} x_1 + 2x_2 + 3x_3 + 4x_4 = 5 \\ 2x_1 + x_2 + 2x_3 + 3x_4 = 1 \\ 3x_1 + 2x_2 + x_3 + 2x_4 = 1 \\ 4x_1 + 3x_2 + 2x_3 + x_4 = -5 \end{cases}$	4. $\begin{cases} x_2 - 3x_3 + 4x_4 = -5 \\ x_1 - 2x_3 + 3x_4 = -4 \\ 3x_1 + 2x_2 - 5x_4 = 12 \\ 4x_1 + 3x_2 - 5x_3 = 5 \end{cases}$
5. $\begin{cases} x_1 + 3x_2 + 5x_3 + 7x_4 = 12 \\ 3x_1 + 5x_2 + 7x_3 + x_4 = 0 \\ 5x_1 + 7x_2 + x_3 + 3x_4 = 4 \\ 7x_1 + x_2 + 3x_3 + 5x_4 = 16 \end{cases}$	6. $\begin{cases} x_1 + 5x_2 + 2x_3 - 4x_4 = 20 \\ 3x_1 + x_2 - 2x_3 = 9 \\ 5x_1 - 7x_2 + 10x_4 = -9 \\ 3x_2 - 5x_3 = 1 \end{cases}$
7. $\begin{cases} 2x_1 + x_2 - 5x_3 + x_4 = 8 \\ x_1 - 3x_2 - 6x_4 = 9 \\ 2x_2 - 3x_3 + 2x_4 = -5 \\ x_1 + 4x_2 - 7x_3 + 6x_4 = 0 \end{cases}$	8. $\begin{cases} 2x_1 - x_2 + 3x_3 + 2x_4 = 4 \\ 3x_1 + 3x_2 + 3x_3 + 2x_4 = 6 \\ 3x_1 - x_2 - x_3 + 2x_4 = 6 \\ 3x_1 - x_2 + 3x_3 - x_4 = 6 \end{cases}$
9. $\begin{cases} x_1 + 2x_2 - x_3 + x_4 = 8 \\ 2x_1 + x_2 + x_3 + x_4 = 5 \\ x_1 - x_2 + 2x_3 + x_4 = -1 \\ x_1 + x_2 - x_3 + 3x_4 = 10 \end{cases}$	10. $\begin{cases} 4x_1 + x_2 + x_3 - x_4 = -9 \\ x_1 - 3x_2 + x_3 + 4x_4 = -7 \\ 3x_2 - 2x_3 + 4x_4 = 12 \\ x_1 + 2x_2 - x_3 - 3x_4 = 0 \end{cases}$
11. $\begin{cases} 2x_1 - x_2 + 3x_3 - x_4 = 1 \\ 2x_1 - x_2 - 3x_4 = 2 \\ 3x_1 - x_3 + x_4 = -3 \\ 2x_1 + 2x_2 - 2x_3 + 5x_4 = -6 \end{cases}$	12. $\begin{cases} x_1 + x_2 - x_3 - x_4 = 0 \\ x_2 + 2x_3 - x_4 = 2 \\ x_1 - x_2 - x_4 = -1 \\ -x_1 + 3x_2 - 2x_3 = 0 \end{cases}$
13. $\begin{cases} 5x_1 + x_2 - x_4 = -9 \\ 3x_1 - 3x_2 + x_3 + 4x_4 = -7 \\ 3x_1 - 2x_3 + x_4 = -16 \\ x_1 - 4x_2 + x_4 = 0 \end{cases}$	14. $\begin{cases} 2x_1 + x_3 + 4x_4 = 9 \\ x_1 + 2x_2 - x_3 + x_4 = 8 \\ 2x_1 + x_2 + x_3 + x_4 = 5 \\ x_1 - x_2 + 2x_3 + x_4 = -1 \end{cases}$
15. $\begin{cases} 2x_1 - 6x_2 + 2x_3 + 2x_4 = 12 \\ x_1 + 3x_2 + 5x_3 + 7x_4 = 12 \\ 3x_1 + 5x_2 + 7x_3 + x_4 = 0 \\ 5x_1 + 7x_2 + x_3 + 3x_4 = 4 \end{cases}$	16. $\begin{cases} x_1 + 5x_2 = 2 \\ 2x_1 - x_2 + 3x_3 + 2x_4 = 4 \\ 3x_1 - x_2 - x_3 + 2x_4 = 6 \\ 3x_1 - x_2 + 3x_3 - 4x_4 = 6 \end{cases}$

17.

$$\begin{cases} x_1 - 4x_2 - x_4 = 2 \\ x_1 + x_2 + 2x_3 + 3x_4 = 1 \\ 2x_1 + 3x_2 - x_3 - x_4 = -6 \\ x_1 + 2x_2 + 3x_3 - x_4 = -4 \end{cases}$$

18.

$$\begin{cases} 5x_1 - x_2 + x_3 + 3x_4 = -4 \\ x_1 + 2x_2 + 3x_3 - 2x_4 = 6 \\ 2x_1 - x_2 - 2x_3 - 3x_4 = 8 \\ 3x_1 + 2x_2 - x_3 + 2x_4 = 4 \end{cases}$$

19.

$$\begin{cases} 4x_1 - 2x_2 + x_3 - 4x_4 = 3 \\ 2x_1 - x_2 + x_3 - x_4 = 1 \\ 3x_1 - x_3 + x_4 = -3 \\ 2x_1 + 2x_2 - 2x_3 + 5x_4 = -6 \end{cases}$$

20.

$$\begin{cases} 2x_1 - x_3 - 2x_4 = -1 \\ x_2 + 2x_3 - x_4 = 2 \\ x_1 - x_2 - x_4 = -1 \\ -x_1 + 3x_2 - 2x_3 = 0 \end{cases}$$

21.

$$\begin{cases} -x_1 + x_2 + x_3 + x_4 = 4 \\ 2x_1 + x_2 + 2x_3 + 3x_4 = 1 \\ 3x_1 + 2x_2 + x_3 + 2x_4 = 1 \\ 4x_1 + 3x_2 + 2x_3 + x_4 = -5 \end{cases}$$

22.

$$\begin{cases} 5x_1 + 3x_2 - 7x_3 + 3x_4 = 1 \\ x_2 - 3x_3 + 4x_4 = -5 \\ x_1 - 2x_3 - 3x_4 = -4 \\ 4x_1 + 3x_2 - 5x_3 = 5 \end{cases}$$

23.

$$\begin{cases} x_1 + x_2 - x_3 - x_4 = 0 \\ x_1 + 2x_2 - 2x_4 = 1 \\ x_1 - x_2 - x_4 = -1 \\ -x_1 + 3x_2 - 2x_3 - 2x_4 = 0 \end{cases}$$

24.

$$\begin{cases} 2x_1 + x_2 - x_3 + 3x_4 = -6 \\ 3x_1 - x_2 + x_3 + 5x_4 = 3 \\ x_1 + 2x_2 - x_3 + 2x_4 = 28 \\ 2x_1 + 3x_2 + x_3 - x_4 = 0 \end{cases}$$

25.

$$\begin{cases} 2x_1 - x_2 + 2x_3 + 2x_4 = -3 \\ 3x_1 + 2x_2 + x_3 - x_4 = 3 \\ x_1 - 3x_2 - x_3 - 3x_4 = 0 \\ 4x_1 + 2x_2 + 2x_3 + 5x_4 = -15 \end{cases}$$

26.

$$\begin{cases} x_1 - 2x_2 + 3x_3 - 4x_4 = -2 \\ 2x_1 + 3x_2 + 4x_3 - 5x_4 = 8 \\ 3x_1 - x_2 - x_3 + 7x_4 = -2 \\ 2x_1 - x_2 + 6x_3 - 3x_4 = 7 \end{cases}$$

27.

$$\begin{cases} 3x_1 + 2x_2 + 5x_3 - x_4 = 3 \\ 2x_1 - 3x_2 - 3x_3 + 4x_4 = 2 \\ 4x_1 + x_2 + 3x_3 + 2x_4 = 3 \\ 5x_1 - 2x_2 + x_3 + 3x_4 = 5 \end{cases}$$

28.

$$\begin{cases} 2x_1 + x_2 + 5x_3 - x_4 = 1 \\ 3x_1 + 3x_2 - 2x_3 - 5x_4 = 2 \\ x_1 - x_2 + 2x_3 + 3x_4 = 10 \\ 3x_1 + 2x_2 + 7x_3 - 2x_4 = 1 \end{cases}$$

29. $\begin{cases} 3x_1 + x_2 + 2x_3 - x_4 = 8 \\ 2x_1 - 3x_2 - 3x_3 + x_4 = -3 \\ 4x_1 + 2x_2 + 5x_3 + 3x_4 = 6 \\ x_1 + 2x_2 - 4x_3 - 3x_4 = -3 \end{cases}$	30. $\begin{cases} 2x_1 + 3x_2 + 5x_3 + x_4 = 6 \\ 3x_1 + x_2 - x_3 + 5x_4 = 0 \\ 2x_1 - x_2 + 3x_3 = -5 \\ 2x_1 + 2x_2 - x_3 + 7x_4 = -3 \end{cases}$

РОЗДІЛ 5
РОЗВ'ЯЗУВАННЯ ЗАДАЧ МАТЕМАТИЧНОГО АНАЛІЗУ

До типових задач математичного аналізу належать обчислювання сум та добутків числових послідовностей, елементів масивів та рядів, обчислення границь послідовностей та функцій, обчислення похідних (диференціювання) та інтегралів (інтегрування). У інженерних розрахунках вони формулюються як окремі задачі, проте часто використовуються як складові у складніших розрахунках.

Розуміння суті та вміння розв'язувати окреслені вище задачі є необхідною умовою одержання правильних розв'язків, їх аналіз та інтерпретування.

5.1. Виконання символьних обчислень у середовищі MathCad Prime

Розрахунки у MathCAD можуть здійснюватися як у числовій формі (результатом є число або сукупність чисел), так і в аналітичній формі (результатом є сукупність змінних, математична залежність, тощо). Пакет MathCAD обладнаний спеціальним процесором для виконання аналітичних (символьних) обчислень. Він дозволяє розв'язати багато задач математики аналітично, без застосування числових методів і, відповідно, без похибок обчислень. У результаті, MathCad дозволяє проводити широкий спектр аналітичних перетворень, таких як алгебричні і матричні операції, основні функції математичного аналізу і розрахунки інтегральних перетворень функцій.

Вирази у символьних розрахунках обчислюються за допомогою оператора аналітичного перетворення (→) замість оператора числового обчислення (=). Стандартні оператори MathCad та багато вмонтованих функцій можна обчислювати аналітично.

Використання аналітичних перетворень виразів має такі переваги:

- на відміну від числових обчислень, символьні операції дозволяють обчислювати вирази без присвоєння значень змінним;

- символьні результати можуть виявляти взаємозв’язки між змінними, які не завжди очевидні з числових результатів;
- символьні розрахунки позбавлені від помилок заокруглення, притаманних числовим розрахункам.

Принципи символьних обчислень. Для символьних обчислень у MathCad Prime призначено меню з вкладки *Math* → *Symbolics* → *Operators*, як подано на рис. 5.1.

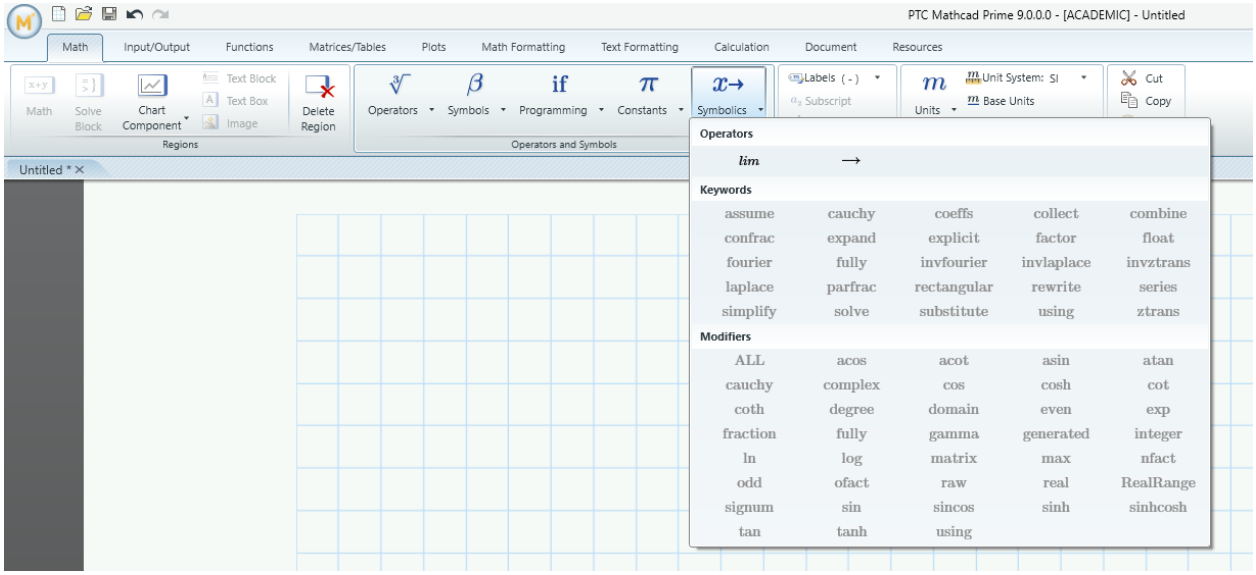


Рис. 5.1. Активування меню з вкладки *Math* → *Symbolics* → *Operators* для виконання символьних обчислень у MathCad Prime

Щоб символьні операції виконувалися, процесору необхідно вказати, над яким виразом ці операції повинні виконуватися, тобто треба відзначити вираз. Для ряду операцій варто не тільки вказати вираз, до якого вони застосовуються, але й відзначити змінну, щодо якої виконується та або інша символьна операція. Сам вираз у такому випадку не відзначається. Таким чином, для виконання операцій із символьним процесором потрібно відзначити об’єкт (цілий вираз або його частину). У літературі використовуються терміни «символьні обчислення» та «символьне виведення» як рівноцінні.

Оператор символьного обчислення (*symbolic evaluation*) позначається символом "→" і у більшості випадків застосовується так само, як оператор

числового виведення, однак внутрішня різниця між функціонуванням цих двох операторів величезна.

Числове виведення – це розрахунок за виразами і числовими методами, а символьне виведення – результат роботи системи штучного інтелекту, вмонтованого в MathCad Prime, який має назву символьного процесора. Етапи символьного обчислення математичного виразу:

- введіть потрібний вираз.
- введіть оператор символьного обчислення за допомогою вкладки *Math* → *Symbolics* → *Operators* (рис. 5.1).
- після цього праворуч від символу оператора символьного обчислення візуалізується аналітичне значення виразу, як подано на рис. 5.2 у прикладах.

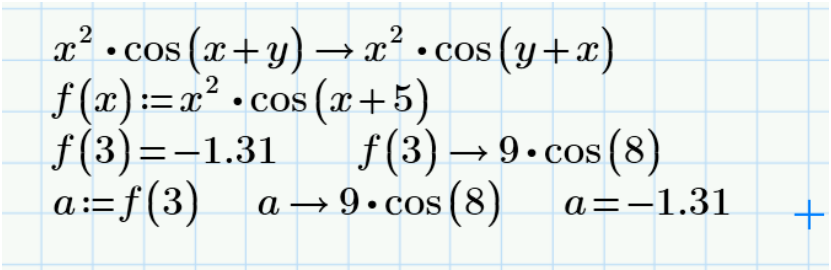


Рис. 5.2. Приклади виконання символьних обчислень

Слід пам’ятати, що для символьного обчислення *не потрібно попередньо означувати значення змінних*, які входять до лівої частини виразу. Якщо ж змінним були присвоєні раніше значення, символьний процесор підставить їх у спрощений вираз і візуалізує результат з урахуванням цих значень.

5.1.1. Виконання алгебричних перетворень

Спрощення виразів – найбільш часто уживана операція. Символьний процесор MathCad Prime так перетворює вираз, щоб вираз набув найбільш просту форму. За цієї обставини використовуються різні арифметичні формули, зведення подібних доданків, тригонометричні тотожності, перерахунок зворотних функцій, тощо. Для спрощення виразу за допомогою оператора символьного обчислення використовуйте функцію *simplify*.

Зауважимо, якщо деяким змінним, які входять у вираз, раніше були присвоєні значення, тоді вони будуть підставлені у цей вираз за виконання символьного виведення.

Спрощення виразів, які містять числа, виконується неоднаково, а у залежності від наявності у числах десяткової крапки. Якщо вона наявна, тоді виконується безпосереднє обчислення виразу. Викладені перетворення подано на рис. 5.3.

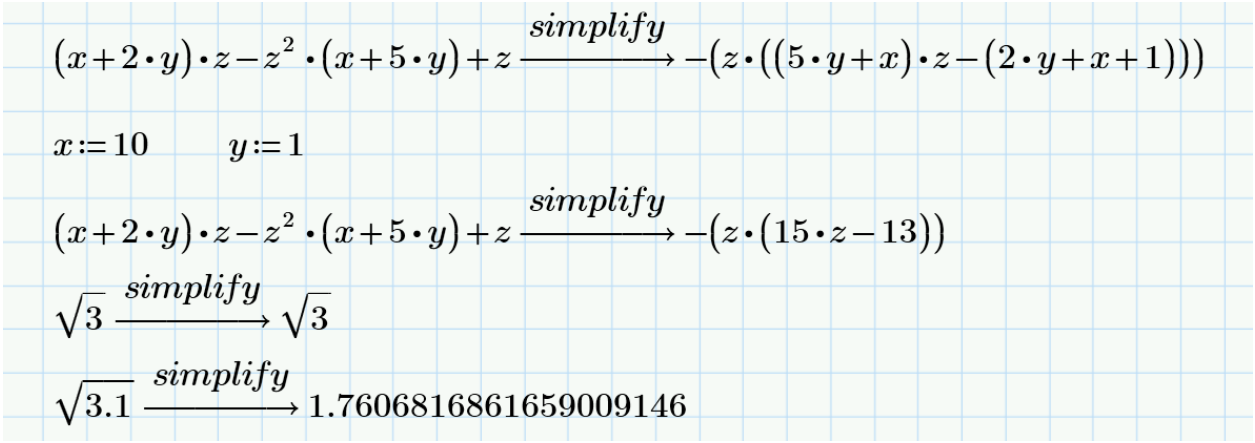


Рис. 5.3. Спрощення виразів з використанням функції *simplify*

5.1.2. Розкладання виразів (функція *expand*)

Операція символьного розкладання виразів є протилежною за змістом до операції спрощення. У процесі розкладання розкриваються всі суми і добутки, а тригонометричні залежності розкладаються за допомогою тригонометричних тотожностей. Розкладання виразів проводиться шляхом використання разом з оператором символьного виведення функції *expand*, рис. 5.4.

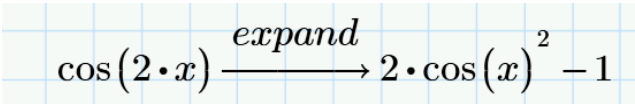


Рис. 5.4. Приклад використання операції символьного розкладання

5.1.3. Розкладання на множники (функція *factor*)

Розкладання виразів на прості множники проводиться використанням разом з оператором символьного обчислення функції *factor*, рис. 5.5.

Ця операція дозволяє розкласти поліноми на добуток простіших поліномів, а цілі числа – на прості множники.

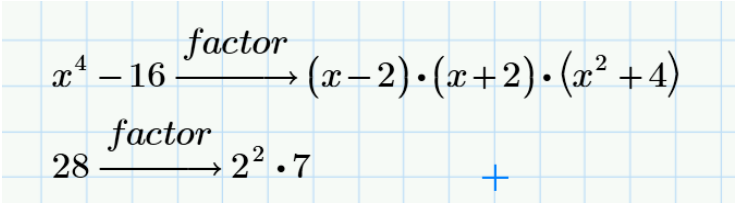


Рис. 5.5. Розкладання виразів на прості множники з використанням функції *factor*

5.1.4. Зведення подібних доданків (*collect*)

Щоб звести подібні доданки за допомогою оператора символьного обчилення потрібно:

- ввести вираз;
- вибрати у вкладці *Math* → *Symbolics* → *collect*;
- після імені функції *collect*, після коми запишіть ім'я змінної, щодо якої потрібно привести подібні доданки (наприклад, змінна *x*, рис. 5.6).

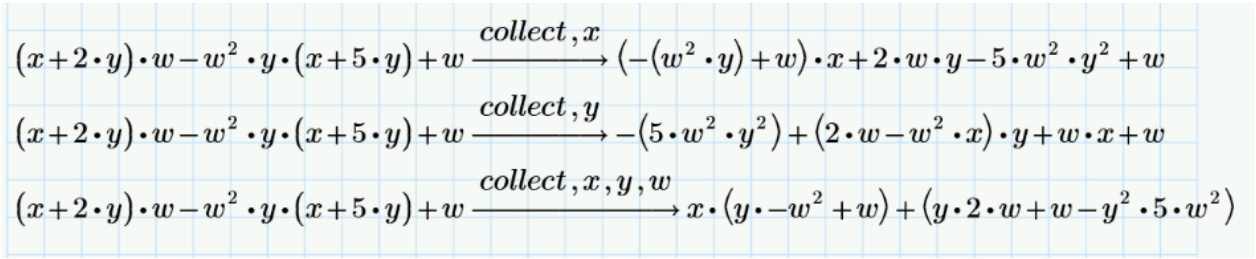


Рис. 5.6. Процес приведення подібних доданків з використанням функції *collect*

Як видно з рисунку, після імені функції *collect* допускається використання кількох змінних, імена яких записуються через кому. У цьому випадку приведення подібних доданків виконується послідовно за усіма змінними.

5.1.5. Коефіцієнти полінома (*Polynomial Coefficients*)

Якщо вираз є поліномом щодо деякої змінної x, заданим не в звичайному вигляді $a_0 + a_1x + a_2x_2 + ...$, а як добуток інших, простіших поліномів, тоді коефіцієнти a_0, a_1, a_2 ... легко визначаються символьним

процесором MathCad. Коефіцієнти самі можуть бути функціями (часом, досить складними) від інших змінних.

Щоб обчислити коефіцієнти полінома за допомогою оператора символьного виведення:

- введіть вираз;
- виберіть у вкладці *Math* → *Symbolics* → *collect*;
- після імені функції *coeffs* через кому введіть аргумент полінома.

Визначення коефіцієнтів можливе не тільки для окремих змінних, але для більш складних виразів, які входять до даного виразу в якості складової частини.

Слід пам’ятати, що не кожен довільний вираз піддається аналітичним перетворенням. Якщо це так (через те, що задача зовсім не має аналітичного розв’язання, або якщо вона виявляється занадто складною для символьного процесора MathCad), тоді як результат виводиться сам вираз.

Приклади розрахунків коефіцієнтів полінома для різних аргументів подано на рис. 5.7.

$$(x+2\cdot y)\cdot w - w^2\cdot y\cdot(x+5\cdot y)+w \xrightarrow{coeffs,x} \begin{bmatrix} -(5\cdot w^2\cdot y^2)+2\cdot w\cdot y+w \\ -(w^2\cdot y)+w \end{bmatrix}$$

$$(x+2\cdot y)\cdot w - w^2\cdot y\cdot(x+5\cdot y)+w \xrightarrow{coeffs,w} \begin{bmatrix} 0 \\ 2\cdot y+x+1 \\ -(5\cdot y^2)-x\cdot y \end{bmatrix}$$

$$(x-4)\cdot(x-7)\cdot x+99 \xrightarrow{coeffs,x} \begin{bmatrix} 99 \\ 28 \\ -11 \\ 1 \end{bmatrix}$$

$$\cos(2\cdot x) \xrightarrow{expand} 2\cdot\cos(x)^2-1$$

$$\cos(x) \xrightarrow{expand} \cos(x)$$

Рис. 5.7. Приклади розрахунків коефіцієнтів полінома для різних аргументів

5.2. Обчислювання суми

Завантаження шаблону для обчислення суми здійснюється у вкладці *Math→Operators→Calculus*. Якщо підвести вказівник мишки на піктограму шаблону – система візуалізує коротке, проте досить вичерпне повідомлення про функціональне призначення цього шаблону, а також альтернативний варіант завантаження з використанням функціональних клавіш клавіатури (рис. 5.8).

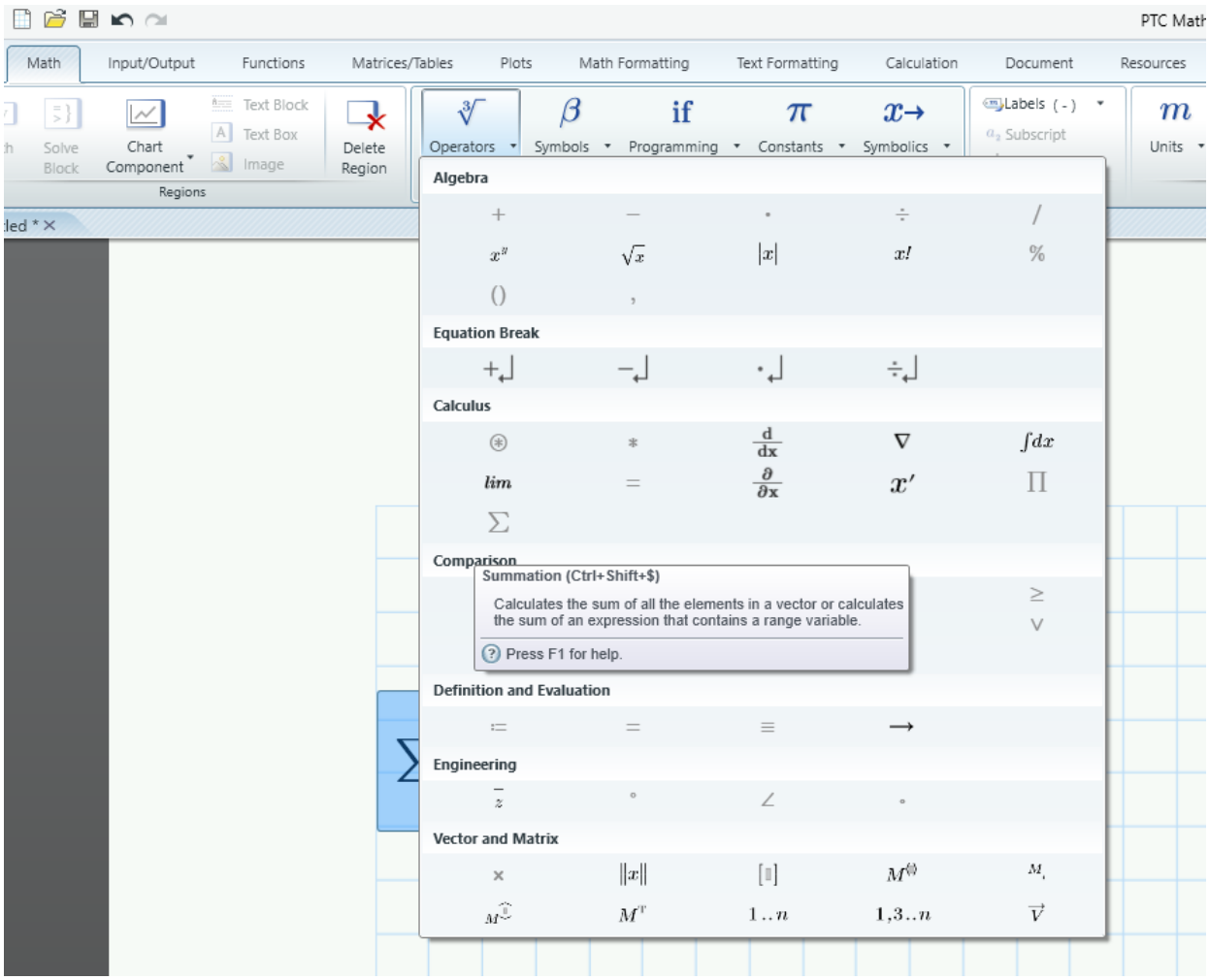


Рис. 5.8. Завантаження шаблону для обчислення суми

Щоб числово обчислити суму, треба ввести відповідний шаблон, заповнити його поля введення і ввести символ ″= ″.

Числовим способом можна розрахувати значення суми скінченної кількості членів арифметичного ряду, послідовності чисел (значень функції),

суму всіх або частини елементів вектора, матриці. Приклади використання подано на рис. 5.9. у наступному MathCad-документі.

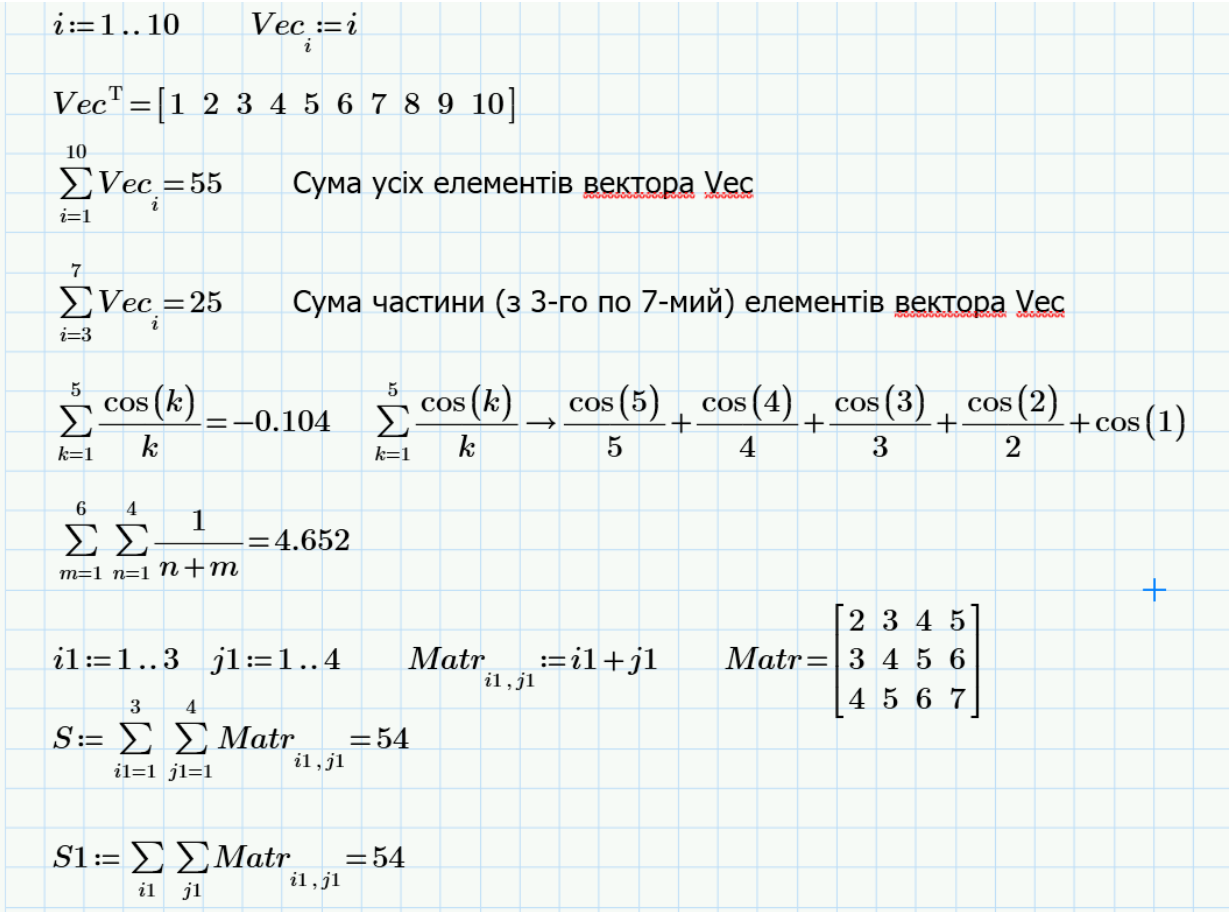


Рис. 5.9. Приклади обчислення суми скінченної кількості послідовності чисел (значень функції), суми елементів вектора та матриці.

У поданому прикладі значення системної змінної ORIGIN дорівнює 1 і визначене у вкладці *Calculation*.

Обчислюючи суми *S* та *S1* використано неповну форму шаблону суми. Це можливо за умови, що значення ранжованих змінних (у нашому випадку *i1* та *j1*) означене до моменту використання неповної форми шаблону.

Символьне (аналітичне) підсумовування елементів (доданків) має загалом ширші можливості за числове. Зокрема, можна обчислювати не лише часткову суму ряду (суму скінченної кількості членів ряду), але і нескінченну суму, обчислювати суму функціональних рядів тощо.

Для обчислювання суми символьним способом потрібно після заповнення всіх полів замість символу ″=″ увести символ ″→″. Значення суми обчислюється за умови, що існує аналітичний вираз для подання результату.

5.3. Обчислювання добутку

Завантаження шаблону для обчислення добутку здійснюється у вкладці *Math→Operators→Calculus*. Якщо підвести вказівник мишки на піктограму шаблону – система також візуалізує коротке, проте досить вичерпне повідомлення про функціональне призначення цього шаблону, а також альтернативний варіант завантаження з використанням функціональних клавіш клавіатури.

Інформація, яка вводиться у поля шаблону перемножування, за змістом є аналогічною, що й для полів шаблону сумування. Обчислювання добутку можна виконувати як числово (задається введенням символу ″=″), так і символьно (задається введенням оператора аналітичного перетворення ″→″). Правила для обчислювання добутку скінченної кількості членів арифметичного ряду, послідовності чисел (значень функції), добутку всіх або частини елементів вектора або матриці тощо, відповідають описаним вище для суми і подані на рис. 5.10.

5.4. Обчислювання похідних (диференціювання функцій)

У пакеті MathCad реалізовано функціонально повні та зручні можливості для диференціювання функцій числовим та символьним способами. У загальному випадку числове диференціювання виконується швидше, проте з меншою точністю порівняно з символьним.

Вибір способу обчислювання похідних (символьний або числовий) важливий під час реалізування ітераційних обчислень, де похідні у циклах треба обчислювати багаторазово − сотні й більше разів.

Зважаючи на високу швидкодію сучасних комп’ютерів, перевагу доцільно надавати символьному способу обчислювання похідних (як і для інших прикладних задач), який є абсолютно точним.

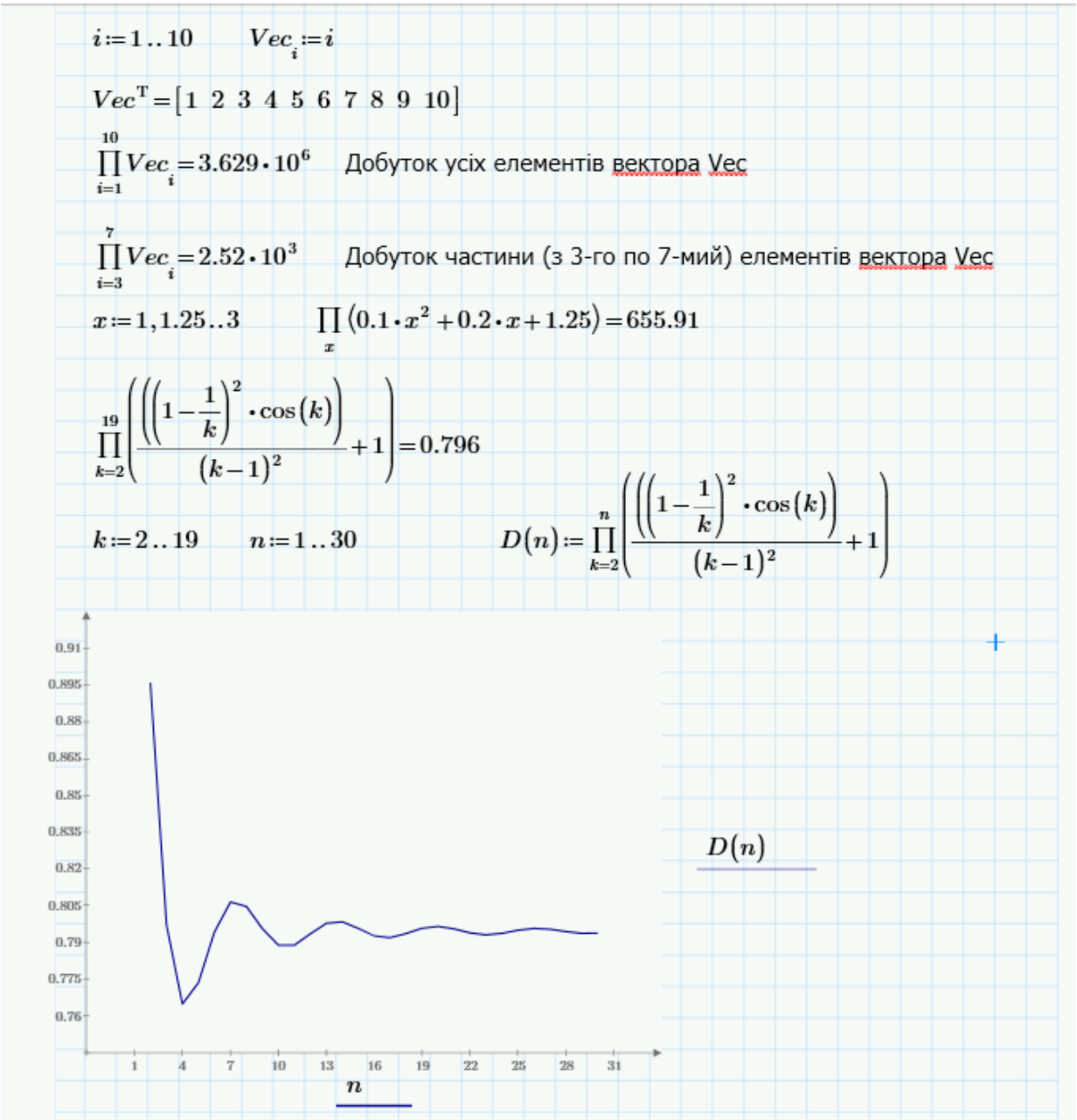


Рис. 5.10. Приклади використання шаблону для обчислення добутків

Для виконання процедури диференціювання у пакеті MathCad передбачені кілька шаблонів, які активуються у вкладці *Math* → *Operators* → *Calculus*. Нагадаємо, якщо підвести вказівник маніпулятора мишки на піктограму шаблону – система візуалізує коротке повідомлення про функціональне призначення цього шаблону, а також альтернативний варіант активування з використанням функціональних клавіш клавіатури.

Після заповнення полів, необхідно увести символ "=" або "→" для обчислювання похідної числово / символьно, відповідно. Подані шаблони

дають змогу обчислювати похідні функцій довільної складності, отримувати похідні від параметрично заданих функцій, функцій декількох змінних, тобто, часткових похідних. Відзначимо, що символьний процесор *Engineering Notebook* повертає вирази похідних не завжди найпростіші, проте вони є правильними. Приклади використання числового і символьного диференціювання подано на рис. 5.11.

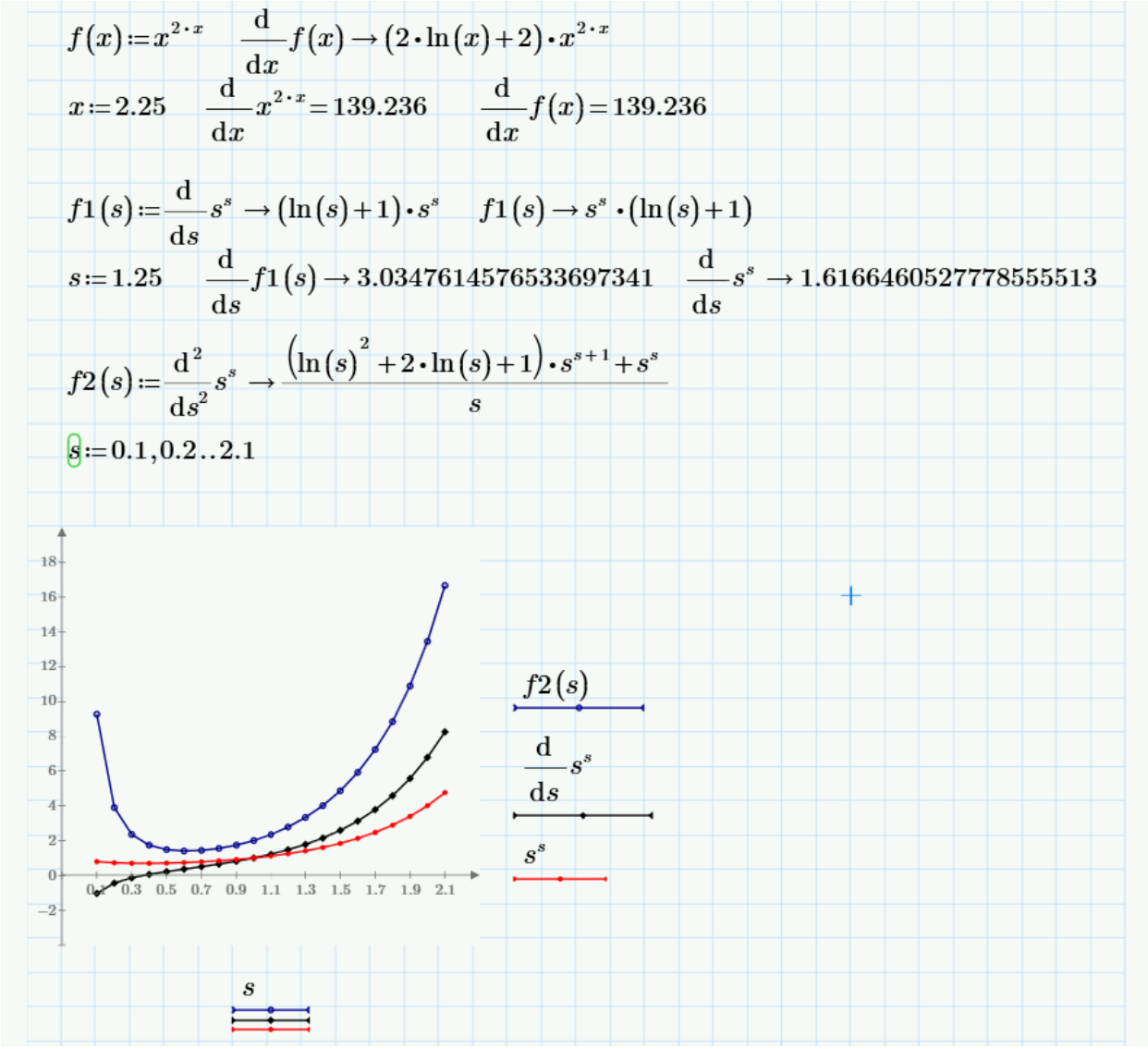


Рис. 5.11. Приклади використання числового і символьного диференціювання

Коли необхідно отримувати значення похідних у багатьох точках відтинку диференціювання, особливо, це важливо в ітераційних обчислювальних алгоритмах, тоді доцільно символьно отримати аналітичний вираз для похідної, оголосити для неї функцію користувача, і, надалі, отримувати значення

похідної як значення цієї функції користувача у заданих точках, тобто за допомогою її табулювання. За такого підходу одержують аналітично точний результат за максимальної швидкодії обчислювального (зокрема, ітераційного) процесу.

5.5. Символьне розв'язання за допомогою функції *find*

Для обчислення символьного розв'язку рівняння або системи рівнянь можна використовувати функцію *find* у *Solve Block*.

Функцію *find* можна використовувати *лише* всередині *Solve Block*.

Розглянемо приклад символьного розв'язання нелінійного рівняння з використанням функції *find* у *Solve Block*, який подано на рис. 5.12.

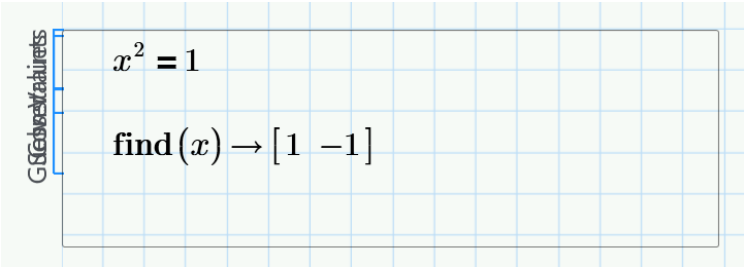


Рис. 5.12. Приклад символьного розв'язання нелінійного рівняння з використанням функції *find* у *Solve Block*

Тепер розглянемо варіант символьного обчислення системи рівнянь за допомогою функції *find* всередині блоку *Solve Block* (рис. 5.13).

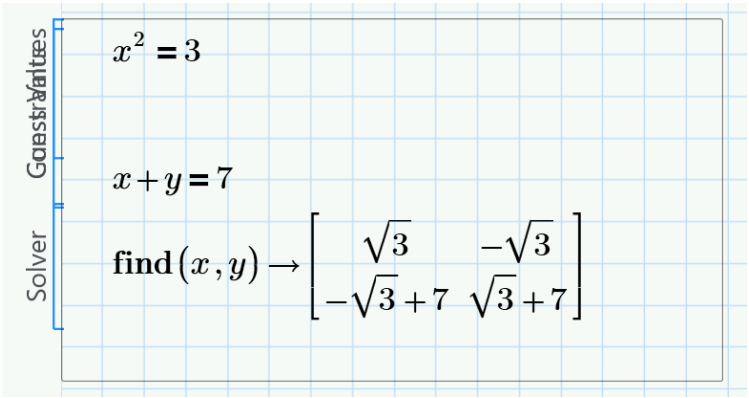


Рис. 5.13. Символьне обчислення системи рівнянь за допомогою функції *find* всередині блоку *Solve Block*

Випадок символьного присвоєння розв'язку новій змінній та обчислення результату поза *Solve Block* подано на рис. 5.14.

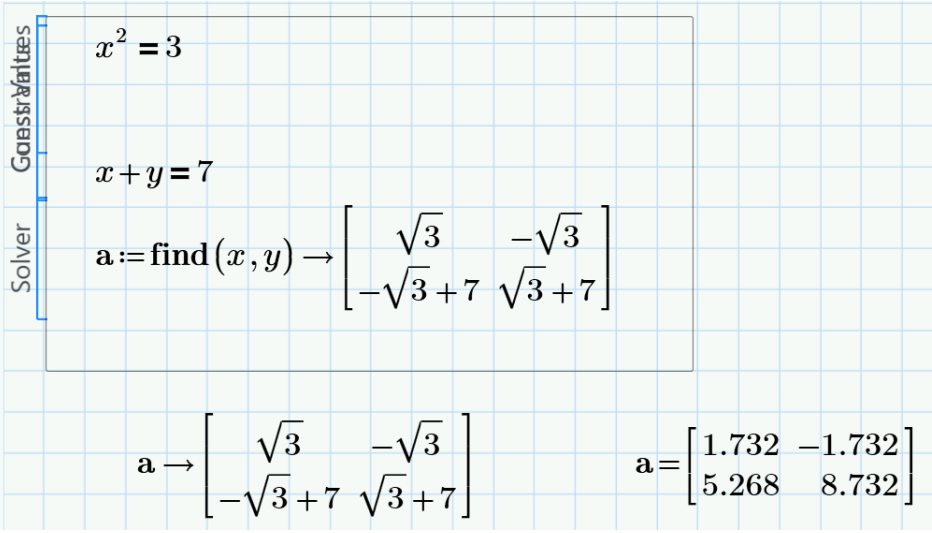


Рис. 5.14. Символьне присвоєння розв'язку новій змінній та обчислення результату поза *Solve Block*

Використовуючи функцію *find* для отримання розв'язків, ви можете використовувати початкові наближення лише для числових обчислень.

У випадку виконання символьних обчислень для отримання розв'язків використовуйте логічні відношення (рис. 5.15).

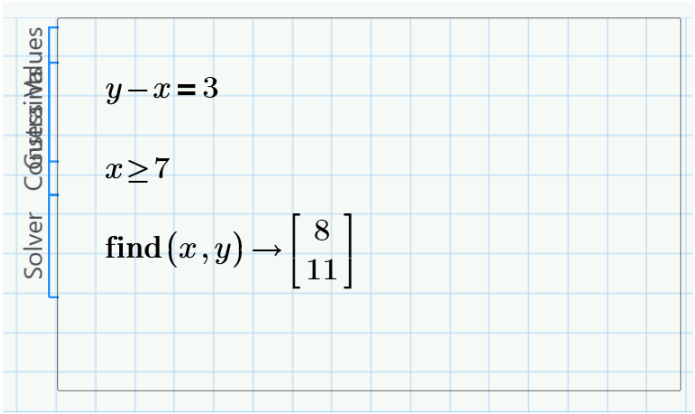


Рис. 5.15. Використання відношення при виконанні символьних обчислень для отримання розв'язків

Індивідуальні завдання

Завдання 1. Обчислити символьно та числово значення суми та добутку *n* елементів послідовності (*n* вибрати з діапазону [10, 20]). Побудувати графіки

зміни значень доданків (множників) від їх номера та зміни поточної суми (добутку) від кількості доданків (множників).

№ вар.	Вираз для суми членів ряду	Вираз для добутку
1	$\sum_{k=1}^{n} cos(0.5k)\left(ln(k)\right)^{-(k-1)}$	$\prod_{k=1}^{n}\left(cos\left(e^{-(k+1)}\right)+1/k!\right)$
2	$\sum_{k=1}^{n}\frac{kcosk}{k^3+e^{k/2}}$	$\prod_{k=1}^{n}\left\|sin\left(arctg(k^{0.5})\right)-1/k^2\right\|$
3	$\sum_{k=1}^{n}\left(\frac{3^{\kappa}}{\kappa!}+arcsin\frac{1}{k}\right)$	$\prod_{k=1}^{n}\left((-1)^k\ \frac{k+1}{k^{k/5}}+1\right)$
4	$\sum_{k=1}^{n}\left(\frac{k}{(2k)^2}+tg\frac{1}{k^2}\right)$	$\prod_{k=1}^{n}\left(1+\frac{sin^k 2k}{k^{1.5}}\right)$
5	$\sum_{k=1}^{n}\left(\frac{sin^k k}{k^{1.5}}+e^{-k}\right)$	$\prod_{k=1}^{n}\left(\frac{2}{k^3}+tg\left(\frac{\pi}{4}-\frac{1}{k^2}\right)\right)$
6	$\sum_{k=1}^{n}(-1)^{k+1}\frac{cos\,1.5k}{k^2}$	$\prod_{k=1}^{n}\left(\frac{6k^3}{k^6+4k^3+5}+\frac{(k+5)}{k^{1.5}}\right)$
7	$\sum_{k=1}^{n}(-1)^k\frac{5\cdot k^5}{2^{3k+1}}$	$\prod_{k=1}^{n}\left(1+\frac{lnk}{(2k^2+1)^2}-e^{-\mathrm{k}}\right)$
8	$\sum_{k=1}^{n}1.67^k\frac{k^2}{(k+1)!}$	$\prod_{k=1}^{n}\left(1+\frac{5}{(2k+1)k}-\frac{\lvert\sin k\rvert}{k^2}\right)$

9	$\sum_{k=1}^{n} \frac{5.35^{k+1}}{k!}$	$\prod_{k=1}^{n} \left(\frac{1}{k(k+1)} - \frac{\cos k}{e^k} + 1 \right)$
10	$\sum_{k=1}^{n} \frac{(-1)^{k+1}(k+1)}{1.25 \cdot k^{2.5}}$	$\prod_{k=1}^{n} \left(\frac{(k+1)^{1/3}}{k!} + k^{-0.01/k} \right)$
11	$\sum_{k=1}^{n} \frac{(k^2+1) \cdot 0.25^k}{2(k-1)!}$	$\prod_{k=1}^{n} \left(\cos 3^{-10k} - \frac{2^{k+2}}{(k+2)!} \right)$
12	$\sum_{k=1}^{n} \frac{k^3}{k^4 + 2k^3 + k!}$	$\prod_{k=1}^{n} \left(\frac{k^2}{k^3 + 2k^2 + 3} + 3^{-1/k} \right)$
13	$\sum_{k=1}^{n} \frac{\cos 0.75k}{k^{2.5} + 2}$	$\prod_{k=1}^{n} \left(e^{-2/k} + \frac{k}{2.5^k} \right)$
14	$\sum_{k=1}^{n} (-1)^{k+1} \frac{0.67^{0.25k-1}}{3k-1} \cdot e^{-0.2k}$	$\prod_{k=1}^{n} \left(2^{-1/k} + \frac{\sin k}{k^{1.5}} \right)$
15	$\sum_{k=2}^{n} \left(\frac{k \sin 2k}{3 + k^2} \right)^3$	$\prod_{k=1}^{n} ctg^2 \left(\frac{\pi}{4} + tg \frac{1}{k^3} \right)$
16	$\sum_{k=1}^{n} \frac{\sin(2k-1)\, k^{1/2}}{(k+1)^{2.5}}$	$\prod_{k=1}^{n} \left(tg \left(\frac{\pi}{4} - \frac{1}{k^2} \right) - \frac{2k}{k+2} \right)$
17	$\sum_{k=1}^{n} \frac{k^{2.5} \cos(2k-1)}{3k^3}$	$\prod_{k=1}^{n} \left(1 + \frac{\sin k}{(2k)^{1.5}} + k^{-2} \right)$

18	$\sum_{k=1}^{n} \frac{k^{-1} + cos2k}{(k^3+5)^{0.5}} e^{\frac{-k}{7.5}}$	$\prod_{k=1}^{n} \left(\frac{e^{-0.5/k} \cos k}{k^2} + 1 \right)$
19	$\sum_{k=1}^{n} \left(k^{-k} - \frac{sin\ 2k}{k!^{0.25}} \right)$	$\prod_{k=1}^{n} \left(\cos \frac{1}{k^2} + \frac{\sin 2k}{k^{1.5}} \right)$
20	$\sum_{k=1}^{n} \frac{(k\ -\ 2)^2}{(k\ -\ 1)!} * cos^k k$	$\prod_{k=1}^{n} \left(\frac{(k\ -\ 1)^2}{(k\ -\ 1)!} - \left\| e^{-k}\ cos^k k \right\| \right)$
21	$\prod_{k=1}^{n} \left(\frac{(k\ -\ 1)^2}{(k\ -\ 1)!} - \left\| e^{-k}\ cos^k k \right\| \right)$	$\prod_{k=1}^{n} \left(\frac{(k\ -\ 1)^2}{(k\ -\ 1)!} - \left\| e^{-k}\ cos^k k \right\| \right)$
22	$\sum_{k=1}^{n} \frac{\sin(k+1) + k \cos(2k)}{k^2 1.15^k}$	$\prod_{k+1}^{n} \left(\frac{\|1 + k \cos 2k\|}{1.5^{k-1}} - 1 \right)$
23	$\sum_{k=1}^{n} (2^{-k} + sin^k 3k)$	$\prod_{k+1}^{n} \left(1 + \frac{sin^k 3k}{k^{1.2}} \right)$
24	$\sum_{k=1}^{n} \left(\frac{1 - \|cosk\|^3}{k^{1.5}} \right)$	$\sum_{k=1}^{n} \frac{(1 - \|cosk\|^3}{k^{1.5}}$
25	$\sum_{k=1}^{n} \frac{(k+1)^{-k} + 1}{\sin(k)\ (k+1)^{-0.75}}$	$\prod_{k+1}^{n} \left(k^{-1/k} - \frac{k^{cosk}}{k! - 2^k} \right)$
26	$\sum_{k=1}^{n} \frac{2k \cos 2k}{k^{2.5} + tg\ (k/3)}$	$\prod_{k=1}^{n} \left(\frac{\sin k}{k^{1.25}} + \frac{In\ k^2}{(k^3+1)^2} + 1 \right)$

27	$\sum_{k=1}^{n}(-1)^k \frac{2\cos(k^2)}{k^{1.35}}$	$\prod_{k=1}^{n}\left(\frac{k^2}{k^4+2k^2+2}+5^{-1/k}\right)$
28	$\sum_{k=1}^{n} 0.7^k \frac{k^3 tg\ (3k)}{(k-1)!}$	$\prod_{k=1}^{n}\left(\frac{\sin k}{\sqrt[10]{k^k}}+1\right)$
29	$\sum_{k=1}^{n}(-1)^{k+1} \frac{0.9^{2k-1}}{2k+1}$	$\prod_{k=1}^{n}\left\lvert 1/k\ +\cos\left(arctg(k)-\frac{\pi}{4}\right)/k^{0.25}\right\rvert$
30	$\sum_{k=1}^{n} \frac{sin^k(3k^2)}{\sqrt[3]{k^{0.25k}}}$	$\prod_{k=1}^{n}\left(k^{-0.004/k}+\frac{(k+1)^{0.125}\sin(k)}{k^{1.25}}\right)$

Завдання 2. Отримати аналітично та засобами символьної математики вирази для обчислення першої та другої похідних функції *y = f(x)* і числово обчислити їхні значення у заданій точці *x*.

Побудувати в одній графічній області графіки функції *y = f(x)* та її перших двох похідних на заданому інтервалі $x \in [a, b]$.

№ вар.	**Функція *y = f(x)***	***x*, $x \in [a, b]$**
1.	$e^{-x/2}$	1.5, [-5, 5]
2.	$x \cdot sin(1/x)$	0.2, [0.1, 0.4]
3.	$(2x^2+3x)(x^2-2)$	1, [-2, 2]
4.	$\sqrt{9-x^2}$	-1, [-2, 2]
5.	$ln(2x+1)$	1.5, [0.5, 2.5]
6.	$e^{2x}+x^2$	0.5, [0, 1]
7.	$cos(3x)$	1.5, [1, 2]

8.	$arctg(1/x)$	0.5, [0.1, 1]
9.	$log_3 x^2$	0.3, [0.2, 0.5]
10.	$ctg(x)$	0.4, [0.2, 0.7]
11.	$5ln(x) + e^x$	0.7, [0.2, 1]
12.	$(x^3 + 1)/x$	1.5, [1, 2]
13.	$ln(5x^2 + 1)$	0.75, [0.1, 1]
14.	$(x^3 + 1)/(x^2 + 1)$	0.6, [0, 1]
15.	$(x^2 + 1)/(x^2 - 1)$	3, [2, 4]
16.	$\sqrt{(x + 1)/(x - 1)}$	2.25, [1.5, 4]
17.	$9x/(x^2 + 1)$	1.5, [-2, 2]
18.	$x^2 cos(2x)$	2, [π/2, π]
19.	$(e^x - e^{-x})(e^x + e^{-x})$	1.5, [-3, 3]
20.	$(x^2 - 2)/(x + 2)$	1.25, [-2, 3]
21.	$x^2 + xcos(x)$	2.75, [-4, 4]
22.	$e^x/2^x$	0.75, [0.2, 1]
23.	$e^{-x} sin(2x)$	-1.5, [-2, 2]
24.	$ln(\sqrt{x})ln(x^2)$	1.5, [1, 2]
25.	$cos(x)cos(2x)$	2, [π/2, π]
26.	$e^x sin(3x)$	1.5, [-1, 1]

27.	$tg(x)sin(x)$	0.75, [0.1, 1]
28.	$ln(2x + 1)$	1.5, [0.5, 2.5]
29.	$5ln(x) + e^x$	0.7, [0.2, 1]
30.	$x \cdot sin(1/x)$	0.2, [0.1, 0.4]

РОЗДІЛ 6
ЧИСЛОВЕ ІНТЕГРУВАННЯ

У MathCad Prime процедуру обчислювання означених інтегралів, як і інші попередньо розглянуті операції математичного аналізу, можна виконувати як числово, так і символьно, а неозначених − лише символьно.

Для виконання процедури інтегрування у *Engineering Notebook* передбачені кілька шаблонів, які активуються у вкладці *Math* → *Operators* → *Calculus*.

Результат числового інтегрування після заповнення полів шаблону виводиться після уведення символу ″=″. Під час числового інтегрування складних функцій треба приймати компроміс між точністю отримуваного результату і часом інтегрування. Точність числового обчислення означеного інтегралу задається змінною TOL. Нагадаємо, що точність ітераційних обчислень цією змінною допустимо задавати у межах від 10–15 (найвища) до 1, за замовчуванням TOL = 0.001. Зрозуміло, що у разі встановлення високої точності час обчислення інтегралу зростає, що особливо важливо для складних підінтегральних функцій.

У *Engineering Notebook* реалізовано ефективний алгоритм автоматичного вибору методу числового інтегрування. Досвід показує, що зазвичай потрібно погоджуватися з автоматичним вибором числового методу інтегрування ядрами *Engineering Notebook*. На рис. 6.1. подано приклади обчислення неозначених та означених інтегралів.

Тепер перейдемо до обчислення означених інтегралів з використанням мови програмування C++. Проте, спочатку викладемо деякі теоретичні міркування щоб написання програмного коду давалося простіше. На відміну від проєктування програмних кодів алгоритмічними мовами програмування у пакеті MathCad такі обчислення можна провести навіть маючи дуже наближені знання у цій галузі. Достатньо правильно заповнити поля шаблонів.

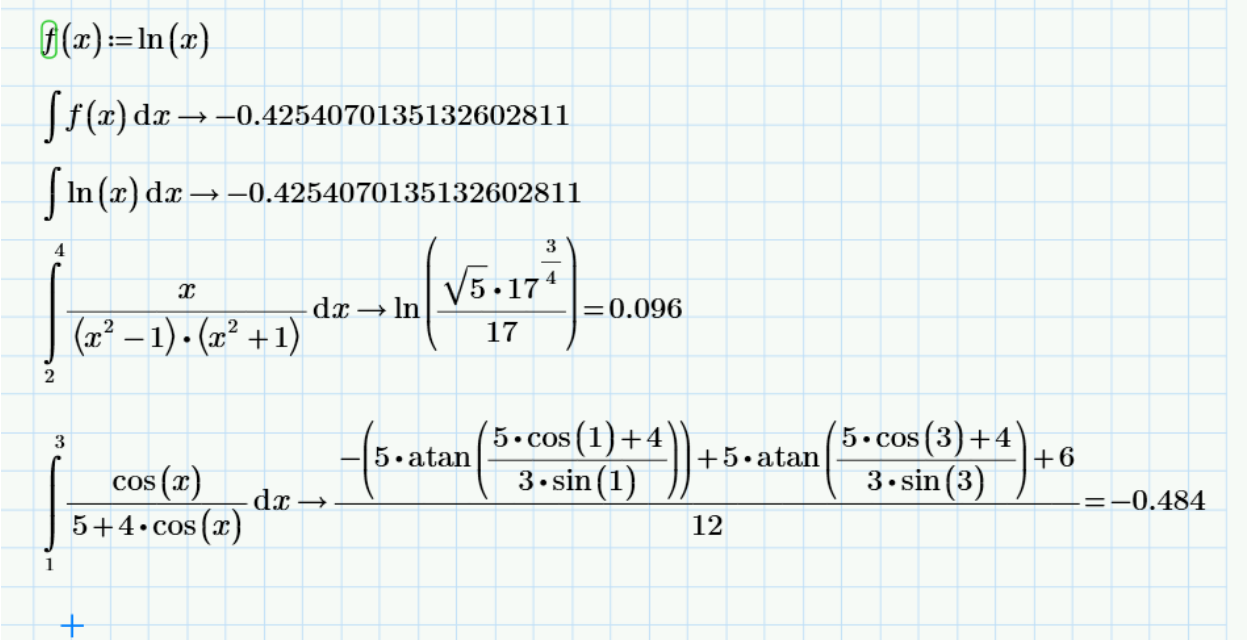


Рис. 6.1. Приклади обчислення неозначених та означених інтегралів

6.1. Означений інтеграл

Потрібно обчислити означений інтеграл

$$\int_a^b f(x)dx$$

де *f(x)* – задана функція, неперервна на проміжку [*a, b*].

Якщо функція задана аналітично і можна знайти її первісну *F(x)*, тоді обчислення проводиться за допомогою формули Ньютона-Лейбніца:

$$\int_a^b f(x)dx = F(b) - F(a)$$

Означений інтеграл числово дорівнює площі криволінійної трапеції, обмеженої кривою $y = f(x)$, віссю OX і прямими $x = a$ і $x = b$ (рис. 6.2).

Отже, задача наближеного обчислення означеного інтеграла зводиться до наближеного обчислення площі криволінійної трапеції. Існує декілька методів наближеного обчислення інтегралів. Розглянемо деякі з них.

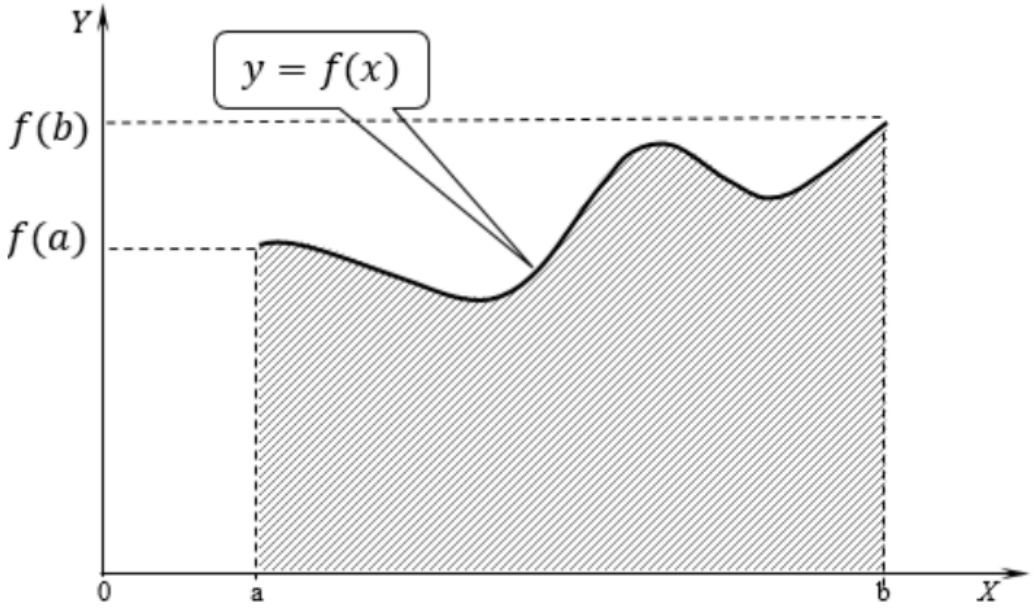


Рис. 6.2. Геометричне інтерпретування означеного інтегралу

6.2. Метод прямокутників

Отже, на відтинку $[a; b]$ задана функція $y = f(x)$. За допомогою точок x_0, x_1, …, x_n розіб'ємо відтинок $[a; b]$ на n елементарних відтинків $[x_{i-1}; x_i]$ $(i = 1,2,\dots n)$, причому $x_0 = a$, $x_n = b$. На кожному з цих відтинків виберемо довільну точку ξ_i $(x_{i-1} \le \xi \le x_i)$ (рис. 6.3):

Обчислимо добутки s_i значення функції у цій точці $f(\xi_i)$ на довжину елементарного відрізка $\Delta x_i = x_i - x_{i-1}$:

$$s_i = f(\xi_i) \cdot \Delta x_i. \tag{6.1}$$

Додамо суму всіх таких добутків:

$$S_n = s_1 + s_2 + \dots + s_n = \sum_{i=1}^{n} f(\xi_i)\Delta x_i. \tag{6.2}$$

Суму S_n називають інтегральною сумою.

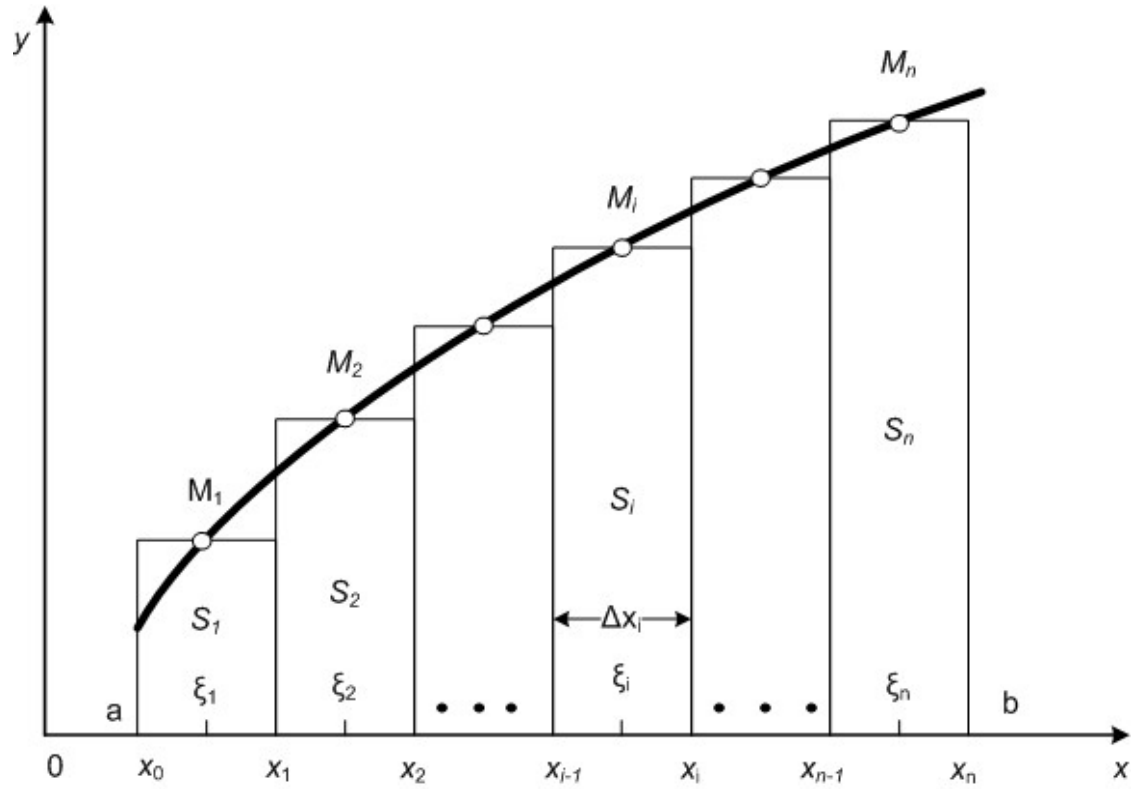


Рис. 6.3. Геометричне інтерпретування методу прямокутників

6.3. Метод лівих і правих прямокутників

Метод прямокутників використовує заміну означеного інтеграла інтегральною сумою (6.2). Як точки ξ_i можуть обиратися ліві $(\xi_i = x_{i-1})$ або праві $(\xi_i = x_i)$ межі елементарних відтинків (рис. 6.4).

Позначивши $f(x_i) = y_i$, $\Delta x_i = h_i$ отримуємо *формули методу прямокутників.*

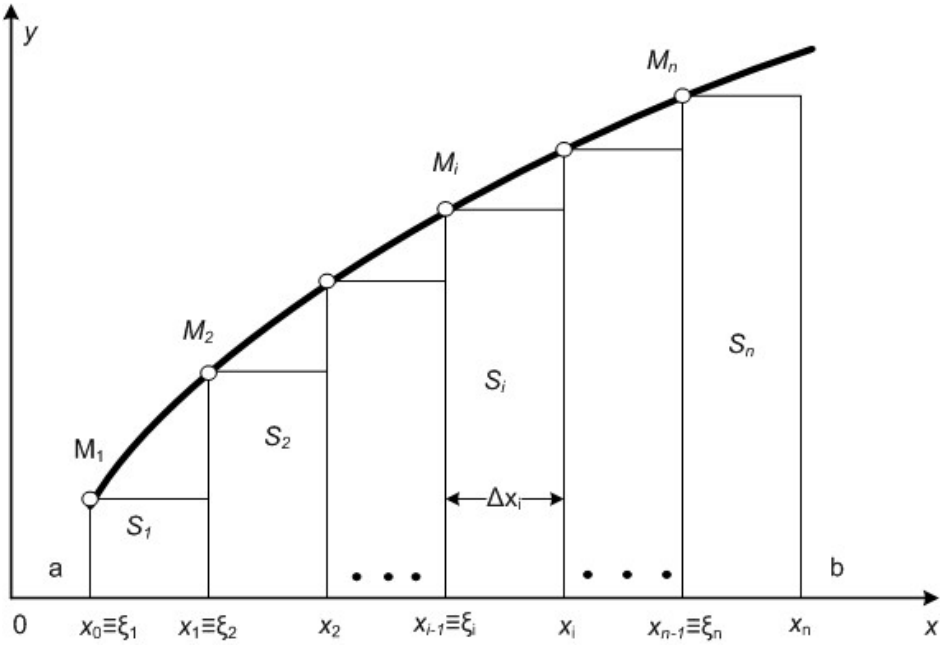
Формула лівих прямокутників:

$$\int_a^b f(x)dx = h_1y_0 + h_2y_1 + \ldots + h_ny_{n-1}.$$

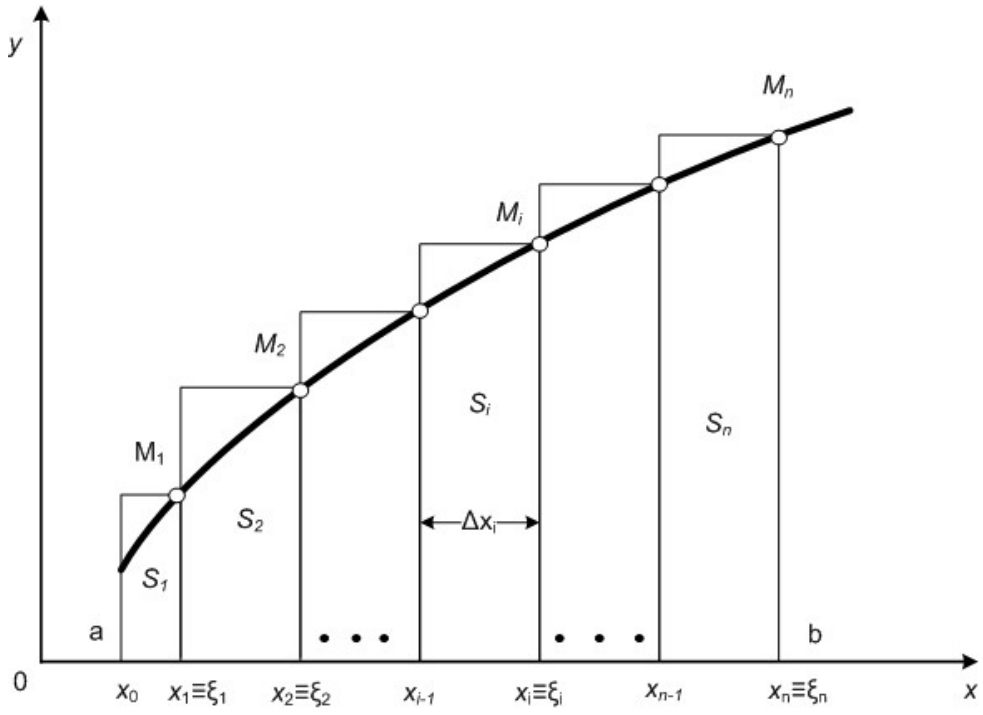
Формула правих прямокутників:

$$\int_a^b f(x)dx = h_1y_1 + h_2y_2 + \ldots + h_ny_n.$$

Проте більш точним є вигляд формули прямокутників, яка використовує значення функції у середніх точках елементарних відтинків.



а



б

Рис. 6.4. Геометричне інтерпретування: а – методу лівих прямокутників; б – методу правих прямокутників

Формула середніх прямокутників:

$$\int_a^b f(x)dx = \sum_{i=0}^{n-1} h_i f\left(x_i + \frac{h_i}{2}\right).$$

Важливим окремим випадком розглянутих формул є їхнє застосування під час числового інтегрування з постійним кроком:

$$h_i = h = \frac{b-a}{n},\ (i = 1,2,\dots,n).$$

Формула середніх прямокутників:

$$\int_a^b f(x)dx = \sum_{i=0}^{n-1} h_i f\left(x_i + \frac{h_i}{2}\right).$$

Важливим окремим випадком розглянутих формул є їхнє застосування під час числового інтегрування з постійним кроком:

$$h_i = h = \frac{b-a}{n},\ (i = 1,2,\dots,n).$$

У такому випадку формули прямокутників спрощуються.

Формула лівих прямокутників: $\int_a^b f(x)dx = h\sum_{i=0}^{n-1} y_i$

Формула правих прямокутників: $\int_a^b f(x)dx = h\sum_{i=1}^{n} y_i$.

Формула середніх прямокутників: $\int_a^b f(x)dx = h\sum_{i=0}^{n-1} f\left(x_i + \frac{h}{2}\right)$.

6.4. Метод Сімпсона

Метод Сімпсона використовує квадратичну інтерполяцію. Відтинок інтегрування $[a; b]$ розбивається на парне число *n* рівних частин із кроком *h*.

На кожному відтинку $[x_0; x_2]$, $[x_2; x_4]$, …, $[x_{n-2}; x_n]$ підінтегральну функцію $y = f(x)$ замінюють інтерполяційним багаточленом $\varphi(x)$ другого степеня (наприклад, багаточленом Лагранжа). Геометрично – замінюють параболою, яка проходить через точки $M_{i-1}(x_{i-1}; y_{i-1})$, $M_i(x_i; y_i)$ та $M_{i+1}(x_{i+1}; y_{i+1})$, $(i = 1,3,5,\dots,n-1)$ (рис. 6.5).

На відтинку $[x_0; x_2]$ елементарна площа s_1 може бути обчислена за допомогою означеного інтеграла

$$s_1 = \int_{x_0}^{x_2} \varphi_1(x)dx = \frac{h}{3}(y_0 + 4y_1 + y_2).$$

Враховуючи, що *n* – парне, провівши такі ж самі обчислення для кожного елементарного відрізка $[x_{i-1}; x_{i+1}]$, $(i = 1,3,5,\dots,n-1)$ та додавши отримані вирази, матимемо:

$$\int_a^b f(x)dx = \frac{h}{3}[y_0 + 4(y_1 + y_3 + y_5 + \dots + y_{n-1}) + 2(y_2 + y_4 + y_6 + \dots + y_{n-2}) + y_n].$$

Цю залежність називають формулою Сімпсона.

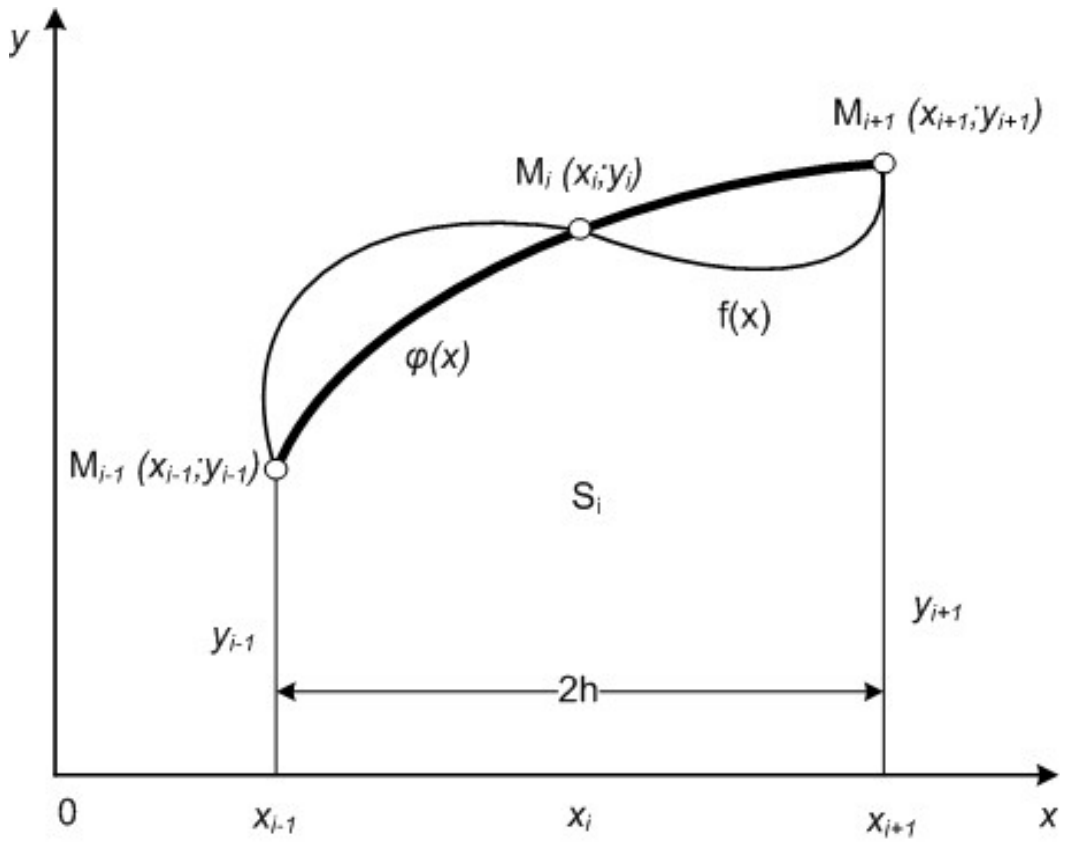


Рис. 6.5. Геометричне інтерпретування методу Сімпсона

6.5. Метод трапецій

Метод трапецій використовує лінійне інтерполювання. Геометрично графік функції $y = f(x)$ подається у вигляді ламаної, що з'єднує точки $M_i(x_i; y_i)$, $(i = 0,1,2,\dots,n)$ (рис. 6.6).

У цьому випадку площа всієї фігури (криволінійної трапеції) складається з площ елементарних прямолінійних трапецій.

Площа кожної такої трапеції дорівнює добутку півсуми основ на висоту:

$$s_i = \frac{y_{i-1}+y_i}{2} h_i,\ i = 1,2,\dots,n.$$

Додавши отримані вирази, отримуємо формулу трапецій для числового інтегрування:

$$\int_a^b f(x)dx = \frac{1}{2}\sum_{i=1}^{n} h_i(y_{i-1} + y_i).$$

Важливим частковим випадком розглянутих формул є їхнє застосування для числового інтегрування з постійним кроком $h_i = h = \frac{b-a}{n}$, $(i = 1,2,\dots,n)$.

Формула трапецій у цьому випадку приймає вигляд:

$$\int_a^b f(x)dx = h\left(\frac{y_0}{2} + y_1 + y_2 + \ldots + y_{n-1} + \frac{y_n}{2}\right) = h\left(\frac{y_0+y_n}{2} + \sum_{i=1}^{n-1} y_i\right).$$

Приклад 1. З точністю до 0.0001 обчислити значення означеного інтегралу $\int_{0.7}^{1.3} \frac{dx}{\sqrt{2x^2+0.3}} \approx 0.4042$ за формулами лівих, правих та середніх прямокутників за умови $n = 30$.

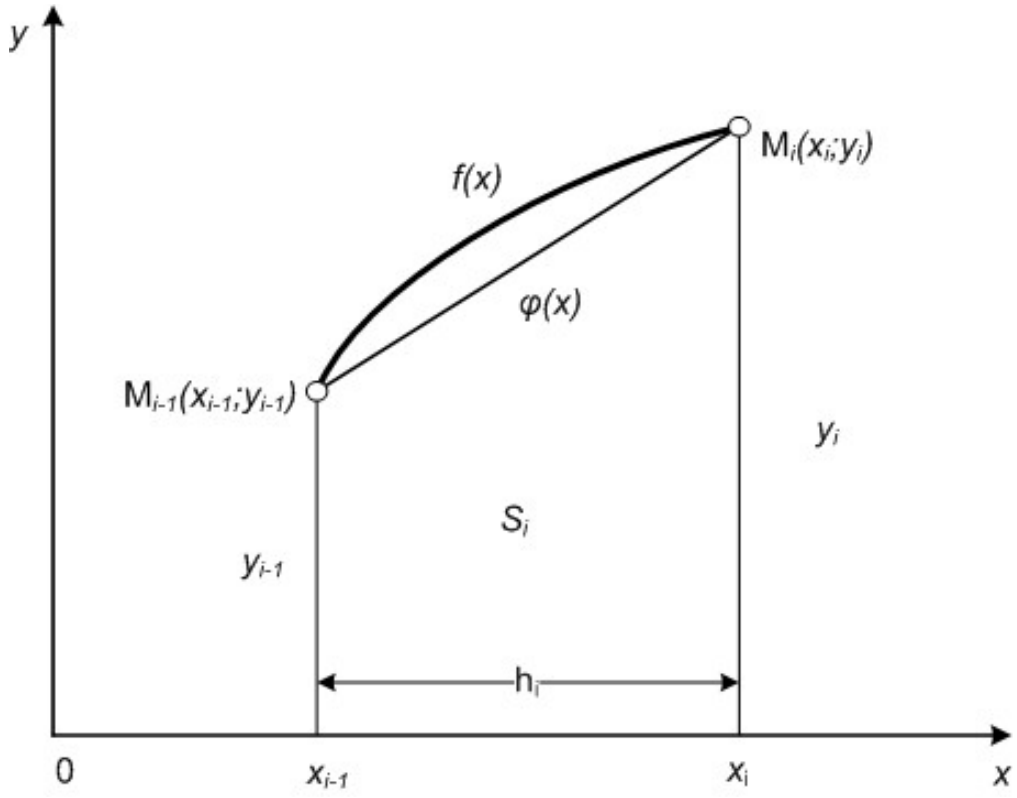


Рис. 6.6. Геометричне інтерпретування методу трапецій

Приклад проєкту програмного коду обчислення означеного інтегралу методами прямокутників, трапецій та Сімпсона мовою C++.

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

double FnI(double x)
{
// My function
    return 1/sqrt(2 * x * x + 1);
```

```
}

double Oi_Left(double(*FnI)(double ), double a, double b,
int n)
// Method of left rectangles
{
    double s = 0;
    double x = a;
    double h = (b - a) / n;
    for (int i = 0; i < n; i++)
    {
        s += FnI(x);
        x += h;
    }
    s *= h;
    return s;
}

double Oi_Right(double(*FnI)(double), double a, double b,
int n)
// Method of right rectangles
{
    double s = 0;
    double h = (b - a) / n;
    double x = a + h;
    for (int i = 1; i <= n; i++)
    {
        s += FnI(x);
        x += h;
    }
```

```
    s *= h;
    return s;
}

double Oi_Middle(double(*FnI)(double), double a, double
b, int n)
// Method of average rectangles
{
    double s = 0;
    double x = a;
    double h = (b - a) / n;
    for (int i = 0; i < n; i++)
    {
        s += FnI(x + h / 2);
        x += h;
    }
    s *= h;
    return s;
}

double Oi_Symps(double(*FnI)(double), double a, double b,
int n)
// Simpson's method
{
    double s = 0;
    double s1 = 0;
    double s2 = 0;
    double h = (b - a) / n;
    double x = a + h;
    for (int i = 1; i < n; i += 2)
```

```
    {
        s1 += FnI(x);
        x += 2 * h;
    }
    x = a + 2 * h;
    for (int i = 2; i < n; i += 2)
    {
        s2 += FnI(x);
        x += 2 * h;
    }
    s += FnI(a) + FnI(b) + s1 * 4 + s2 * 2;
    s *= h / 3;
    return s;
}

double Oi_Tr(double(*FnI)(double), double a, double b,
int n)
// Trapezium method
{
    double s = 0;
    double x = a;
    double h = (b - a) / n;
    for (int i = 1; i < n; i++)
    {
        s += FnI(x);
        x += h;
    }
    s += FnI(a) / 2 + FnI(b) / 2;
    s *= h;
    return s;
```

```
}

int main()
{
    double a;
    double b;
    int n;
    cout << "Integration of a function
f(x)=1/sqrt(2*x*x+1) on the interval [a; b]";
    cout << endl;
    cout << "\nBegin of interval a = ";
    cin >> a;
    cout << "\nEnd of interval b = ";
    cin >> b;
    cout << "Enter the number of division segments n = ";
    cin >> n;
    cout << " Method of left rectangles:";
    cout << " Oi = " << Oi_Left(FnI, a, b, n) << endl;
    cout << " Method of right rectangles:";
    cout << " Oi = " << Oi_Right(FnI, a, b, n) << endl;
    cout << " Method of average rectangles:";
    cout << " Oi = " << Oi_Middle(FnI, a, b, n) << endl;
    cout << " Simpson's method:";
    cout << " Oi = " << Oi_Symps(FnI, a, b, n) << endl;
    cout << " Trapezium method:";
    cout << " Oi = " << Oi_Tr(FnI, a, b, n) << endl;
    system("pause");
    return 0;
}
```

Результат виконання програмного коду:

```
Integration of a function f(x)=1/sqrt(2*x*x+1) on the
interval [a; b]
a = 0.7
b = 1.3
Enter the number of division segments n = 30
 Method of left rectangles: Oi = 0.352161
 Method of right rectangles: Oi = 0.347504
 Method of average rectangles: Oi = 0.349821
 Simpson's method: Oi = 0.349825
 Trapezium method: Oi = 0.354375
```

6.6. Обчислення екстремумів функції

Точками екстремуму визнаються такі точки на графіку функції, у яких досягається її мінімальне або максимальне значення. Точки екстремуму – це загальна назва точок максимумів та мінімумів. Щоб дізнатися кількість екстремумів кожного типу та їх розміщення, слід дослідити графік функції у заданому діапазоні.

Екстремуми можуть бути глобальними та локальними.

Глобальний екстремум є точкою на графіку функції, у якій досягається її мінімальне або максимальне значення.

Локальний екстремум є точкою на графіку функції, у якій досягається її найменше або найбільше значення виключно у рамках заданого діапазону.

Необхідна умова екстремуму функції – у точці екстремуму похідна дорівнює нулю.

Достатня умова екстремуму функції – у деякій точці похідна перетворюється на нуль і, окрім того, похідна, проходячи через неї, змінює свій знак, тоді у цій точці функція досягає екстремуму.

Для пошуку екстремумів функцій $f(x_1, x_2, \ldots, x_n)$ однієї або декількох змінних у MathCad є дві внутрішні функції:

- *minimize(f, x_1, x_2, ...)* – для обчислення координат мінімуму;
- *maximize(f, x_1, x_2, ...)* – для обчислення координат максимуму.

Обидві функції повертають значення аргументів досліджуваної функції. При звертанні до цих функцій необхідно задавати початкове наближення розв'язку. Наприклад, обчислення екстремумів функції однієї змінної без додаткових умов (рис. 6.7)

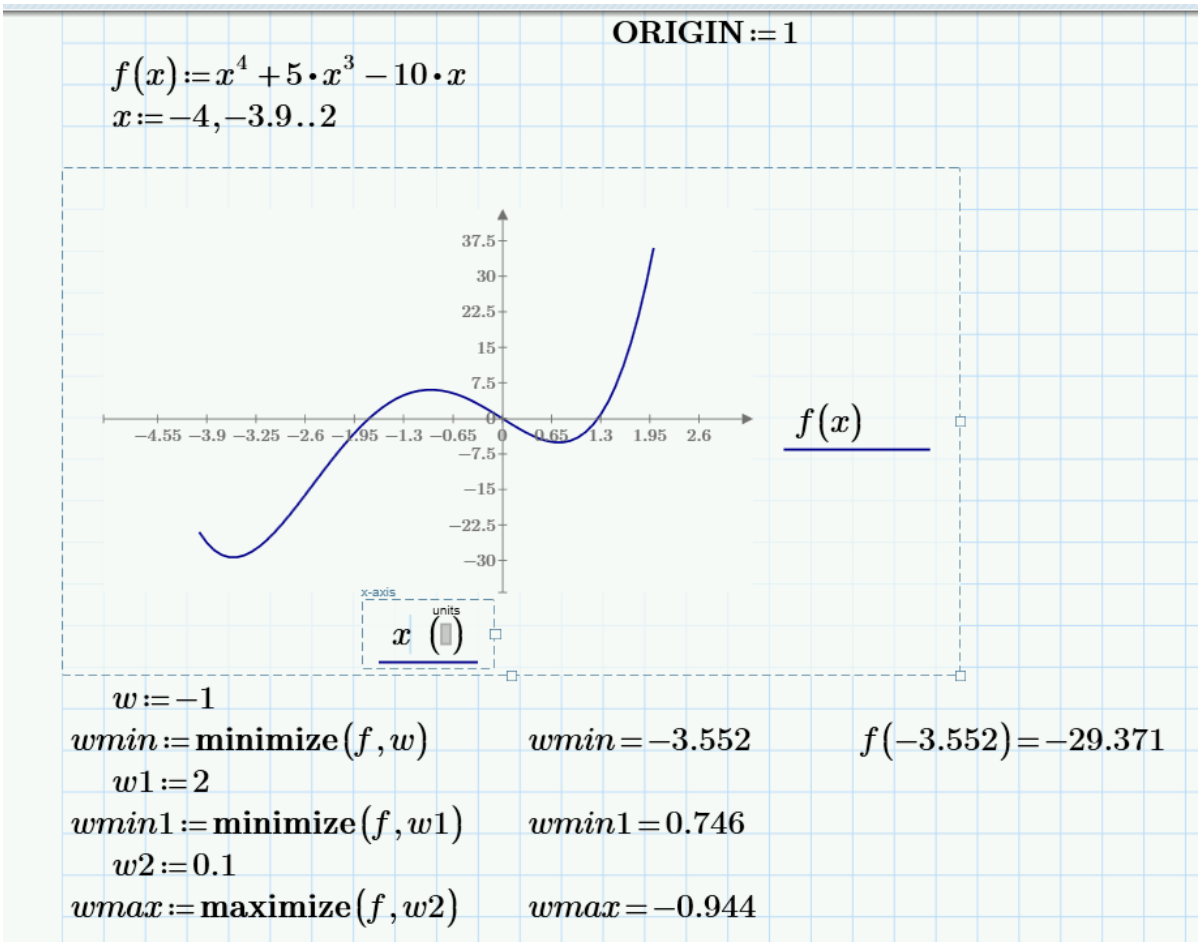


Рис. 6.7. Обчислення екстремумів функції однієї змінної без додаткових умов

Індивідуальні завдання

Завдання 1. Обчислити символьно неозначений та числово і символьно означений інтеграл функції *y = f(x)* на відтинку інтегрування [*a, b*] у *Engineering Notebook* MathCad Prime. Побудувати в одній графічній області графіки функцій *y = f(x) та Integ(x)* = $\int_a^x f(x)dx$ на відтинку $x \in [a, b]$.

Розробити проєкт програмного коду для обчислення означеного інтегралу мовою С++ методами лівих та правих прямокутників, методом трапецій та методом Сімпсона. Значення кількості інтервалів для обчислення інтегралу *n* – вводити з клавіатури (*n=10, 100, 1000*). Також для виконання обчислень у табл. 6.1 подано значення відтинків інтегрування [*a, b*].

Варіанти підінтегральних функцій подано у таблиці 6.1.

Таблиця 6.1

Індивідуальні варіанти підінтегральних функцій

№ з/п варіанту	*f(x)*	№ з/п варіанту	*f(x)*
1	$x^3 + 2x^2 + 2$	2	$x^3 + 0{,}2x^2 + 0{,}5x + 0{,}8$
3	$x^3 - 3x^2 + 9x - 10$	4	$x^3 + 4x - 6$
5	$x^3 - 2x + 2$	6	$x^3 + 0{,}1x^2 + 0{,}4x - 1{,}2$
7	$x^3 + 3x - 1$	8	$x^3 + 3x^2 + 6x - 1$
9	$x^3 + x - 3$	10	$x^3 - 0{,}1x^2 + 0{,}4x - 1{,}5$
11	$x^3 + 0{,}4x^2 + 0{,}6x - 1{,}6$	12	$x^3 - 3x^2 + 6x - 2$
13	$x^3 - 0{,}2x^2 + 0{,}4x - 1{,}4$	14	$x^3 - 0{,}2x^2 + 0{,}3x - 1{,}2$
15	$x^3 - 0{,}1x^2 + 0{,}4x + 2$	16	$x^3 - 3x^2 + 12x - 9$
17	$x^3 + 3x^2 + 12x + 3$	18	$x^3 + 0{,}2x^2 + 0{,}5x - 1{,}2$
19	$x^3 - 0{,}2x^2 + 0{,}5x - 1$	20	$x^3 + 3x + 1$
21	$x^3 - 0{,}1x^2 + 0{,}4x + 1{,}2$	22	$x^3 + 0{,}2x^2 + 0{,}5x - 1{,}2$
23	$x^3 - 3x^2 + 6x - 5$	24	$x^3 - 3x^2 + 9x + 2$
25	$x^3 - 0{,}2x^2 + 0{,}5x - 1{,}4$	26	$x^3 - 0{,}1x^2 + 0{,}4x - 1{,}5$
27	$x^3 + 2x + 4$	28	$x^3 - 3x^2 + 6x + 3$
29	$x^3 - 3x^2 + 12x - 12$	30	$x^3 - 0{,}1x^2 + 0{,}3x - 0{,}6$

Виконати порівняльний аналіз отриманих результатів.

РОЗДІЛ 7
МЕТОДИ ОБРОБЛЕННЯ ЕКСПЕРИМЕНТАЛЬНИХ ДАНИХ. ІНТЕРПОЛЮВАННЯ ТА ЕКСТРАПОЛЮВАННЯ ФУНКЦІЙ У ПАКЕТІ MathCad

У *Engineering Notebook* Mathcad Prime для інтерполювання (обчислення значень у межах діапазону даних) і екстраполювання (прогнозування значень за межами діапазону даних) використовуються спеціальні вмонтовані функції, які працюють з векторами даних.

У цьому розділі розглянемо найпростіші методи оброблення даних – інтерполювання/екстраполювання. Вважатимемо, що основним об'єктом дослідження буде наявність експериментальних даних, які найчастіше подаються у вигляді масиву, який складається з пар чисел (x_i, y_i).

Зважаючи на цю обставину постає завдання апроксимування дискретної залежності $y(x_i)$ безперервною функцією $f(x)$. Функція $f(x)$, залежно від специфіки завдання, може відповідати різним вимогам:

- $f(x)$ повинна проходити через точки (x_i, y_i), тобто $f(x_i) = y_i$, $i = 1...n$. У такому випадку говорять про інтерполювання даних функцією $f(x)$ у внутрішніх точках між x_i або екстраполювання за межами інтервалу, який містить всі x_i;
- $f(x)$ має деяким чином (наприклад, у вигляді певної аналітичної залежності) наближати $y(x_i)$, не обов'язково проходячи через точки (x_i, y_i).

Таким є формулювання задачі регресії, яку в багатьох випадках також можна назвати згладжуванням даних. Узагальнено, згладжування є окремим випадком фільтрування даних, започаткованого на зменшенні шумової компоненти вимірювань.

У *Engineering Notebook* для інтерполювання та екстраполювання використовуються вмонтовані функції, які дозволяють «з'єднати» точки даних кривою різного ступеня гладкості.

7.1. Основні функції та методи

У *Engineering Notebook* вмонтовано такі функції для інтерполювання та екстраполювання, зокрема:

- *linterp(vx, vy, x)*: виконує лінійне інтерполювання. Вона обчислює значення *y* для заданого *x*, проводячи пряму лінію між двома найближчими точками даних;
- *interp(vs, vx, vy, x)*: більш універсальна функція, яка використовує коефіцієнти, згенеровані іншими функціями (наприклад, *cspline*, *lspline*, *pspline*, *bspline*), для інтерполювання або екстраполювання.
- *cspline(vx, vy)*, *lspline(vx, vy)*, *pspline(vx, vy)*: ці функції генерують вектор коефіцієнтів (*vs*), який потім використовується функцією *interp* для побудови кривої (кубічний сплайн, лінійний сплайн, параболічний сплайн, відповідно).

Щоразу, коли ви використовуєте масиви у довільній із функцій, описаних у цьому розділі, переконайтеся, що кожен елемент масиву містить значення даних. Майте на увазі, що кожному елементу масиву необхідно присвоїти значення, *Engineering Notebook* присвоює 0 довільним елементам, які ви явно не ініціалізували.

7.2. Використання функції *interp*

1. Функція *interp* у MathCad Prime використовується для виконання кубічного сплайн-інтерполювання. Вона визначає інтерполюючу криву на основі заданих наборів даних, яка проходить через всі задані точки.

2. Для використання функції *interp* вам необхідно виконати два основні кроки:

❖ Обчислити вектор других похідних (*VS*): Спочатку потрібно розрахувати вектор, який містить другі похідні інтерполяційної кривої у кожній із заданих точок. Для цього використовуються одна з таких функцій:

cspline(X, Y): умова вільних кінців сплайну;

pspline(X, Y): умова періодичних кінців сплайну;

lspline(X, Y): умова лінійних кінців сплайну.

Тут X і Y це вектори початкових даних (координати точок), причому значення у векторі X повинні бути відсортовані за зростанням.

❖ Застосувати функцію *interp*: Після отримання вектора *VS* функція *interp* використовується для обчислення значення інтерпольованої функції у конкретній проміжній точці *x*.

$$VS:=cspline(X, Y)$$

$$y\_interp:=interp(VS, X, Y, x),$$

де: *X*, *Y* – вектори початкових даних (мають однакову довжину); *VS* – вектор других похідних, обчислений за допомогою однієї з функцій сплайн;

x – значення незалежної змінної, для якої потрібно обчислити інтерпольоване значення; *y_interp* – обчислене інтерпольоване значення функції у точці *x*.

7.3. Лінійне інтерполювання

Найпростішим є лінійне інтерполювання, яке полягає у наближенні таблично означеної функції між вузлами інтерполювання відтинками прямих. Для реалізування лінійного інтерполювання у MathCad є вмонтована функція *linterp(vx, vy, x)*, де *vx, vy* – вектори, які сформовані з координат експериментальних (вузлових) точок;

x – абсциса точки, у якій необхідно обчислити значення інтерпольованої функції.

Використання функції *linterp(vx, vy, x)* дає змогу не лише відтворити графічно значення експериментальних точок, тобто побудувати інтерпольовану ламану криву, але і отримати значення фізичної величини у довільній точці між ними.

Приклад 1. Лінійне інтерполювання. Припустимо, у нас є такі дані: вектор Y та відповідний набір значень X. Використаємо функцію *linterp* і для виконання лінійного інтерполювання виконаємо таку послідовність дій:

1. Визначимо вектори даних.
2. Звертаємося до функції *linterp* для виконання лінійного інтерполювання між точками даних.

3. У спільній графічній області будуємо точки даних та інтерпольовану криву.

Цей процес виконання обчислень подано MathCad-документом на рис. 7.1.

Інтерполювання допускає, що остання апроксимаційна функція, та, яка обчислюється між двома останніми точками, залишається істинною для всіх інших точок на цьому кінці діапазону. Це припущення є не зовсім доречним. Існують різні методи прогнозування значень поза діапазоном даних. На додаток, крива, створена за допомогою лінійного інтерполювання, не є гладкою у кожній сполучній точці даних, тому немає визначених похідних інтерпольованої кривої. З огляду на цю обставину ви можете використовувати кубічний сплайн для апроксимування ваших даних.

Як і очікувалося, функція *linterp* виводить пряму лінію між кожним набором точок даних.

7.4. Інтерполювання кубічними сплайнами

Складнішим є інтерполювання кубічними сплайнами (сплайн інтерполювання). У цьому разі експериментальні точки з'єднуються відтинками поліномів третього степеня, тобто кубічними сплайнами.

Тут повинна не лише виконуватися умова проходження інтепольованої кривої через експериментальні точки, але і дотримується умова рівності першої і другої похідних ліворуч та праворуч від експериментальних точок. Тому, інтерпольована крива є згладженою. Особливими є перша та остання точки, у яких немає сплайнів ліворуч і праворуч.

Тому, залежно від того, які умови накладаються на сплайни перших та останніх точок, виокремлюють три види інтерполювання кубічними сплайнами. Відповідно використовують три різні способи формування векторів других похідних інтерпольованої кривої в усіх експериментальних точках, які означені у векторах *vx, vy*. *Engineering Notebook* має три вмонтовані функції для формування вектора других похідних.

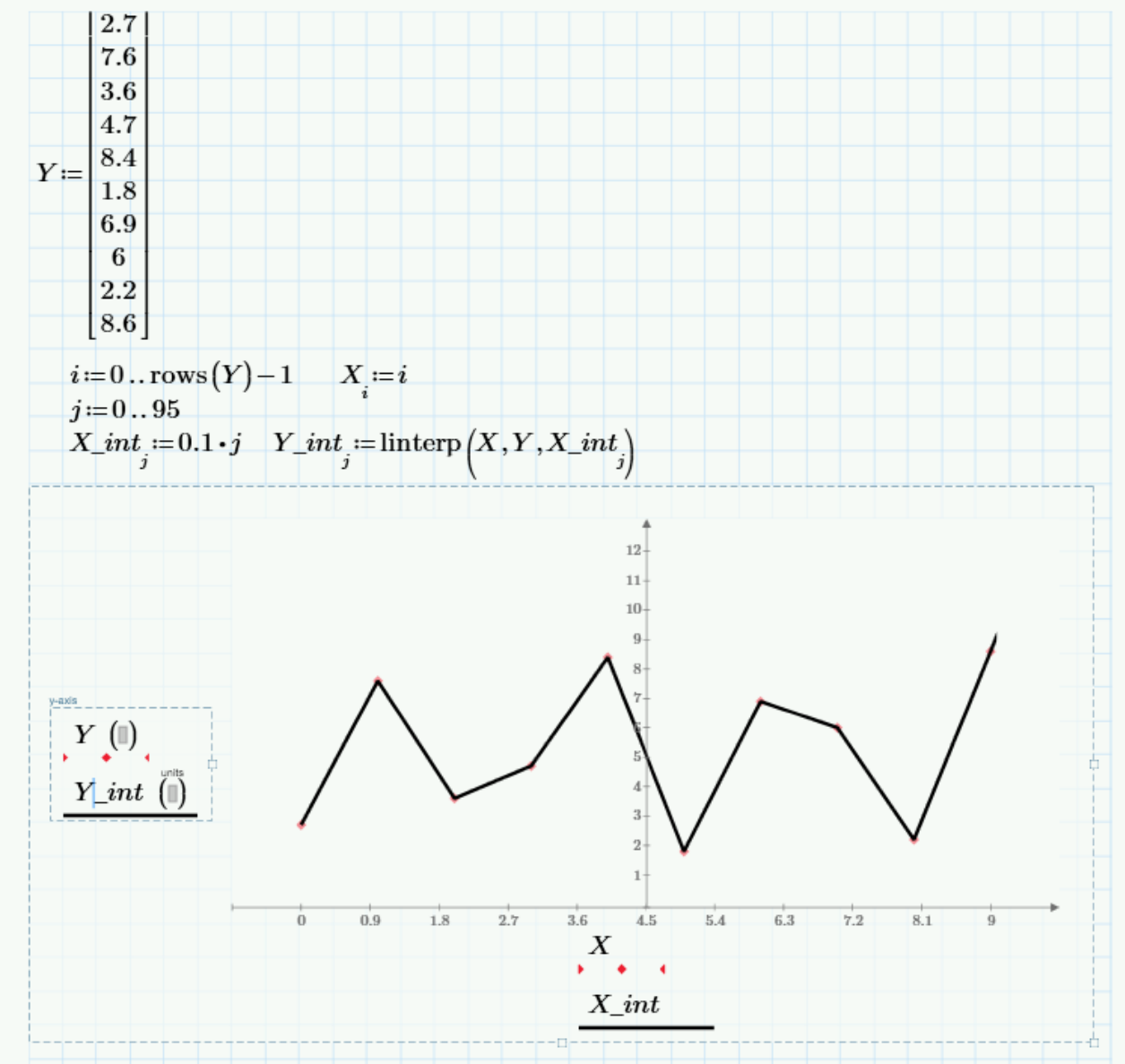


Рис. 7.1. Лінійне інтерполювання з використанням функції *linterp*

Цей вектор надалі використовується для побудови інтерпольованої кривої:

- *lspline(vx, vy)* – у першій та останній точках виконується умова рівності нулю другої похідної;
- *pspline(vx, vy)* – на першому та останньому інтервалах використовується параболічний сплайн, тобто сплайн з нульовим коефіцієнтом при x^3;
- *cspline(vx, vy)* – поліномні коефіцієнти сплайнів при x^3 на двох перших та двох останніх інтервалах приймаються однаковими.

Розглянемо приклад використання функцій *lspline*, *pspline* та *cspline* для побудови кубічних сплайнів та інтерполювання між точками даних. Обчислення виконаємо за таким алгоритмом:

1. Визначимо матрицю *Cu*.
2. Скористаємо з функції *csort*, щоб відсортувати дані так, аби другий стовпець матриці *Cu* був відсортований у порядку зростання.

Ще раз нагадаємо: значення *x*, які передаються до сплайн-функцій, обов'язково повинні бути відсортовані у порядку зростання.

3. Створимо вектори, які містять початкові дані *x* та *y*.
4. Звернемося до функції *cspline* для створення кубічного сплайн-вектора, а потім звернемося до функції *interp* для отримання інтерпольованих значень.
5. Звернемося до функції *lspline* для створення лінійного сплайн-вектора, а потім звернемося до функції *interp* для отримання інтерпольованих значень.
6. Звернемося до функції *pspline* для створення параболічного сплайн-вектора, а потім звернемося до функції *interp* для отримання інтерпольованих значень.
7. Побудуємо графік початкових точок даних та кубічних сплайнів у спільній області побудови.
8. Збільшимо масштаб перших двох точок даних.
9. Обчислимо другу похідну інтерпольованого лінійного сплайнового вектора та покажемо, що вона дорівнює 0 у кінцевих точках.
10. Обчислимо другу похідну інтерпольованого параболічного сплайна та покажемо, що у кінцевих точках вона дорівнює значенню найближчої точки.

- Обчислимо другу похідну в першій та другій точках і покажемо, що вони рівні.
- Обчислимо другу похідну в передостанній та останній точках і покажемо, що вони рівні.

Реалізування пунктів 1–7 розглянутого алгоритму подано у MathCad-документі на рис. 7.2.

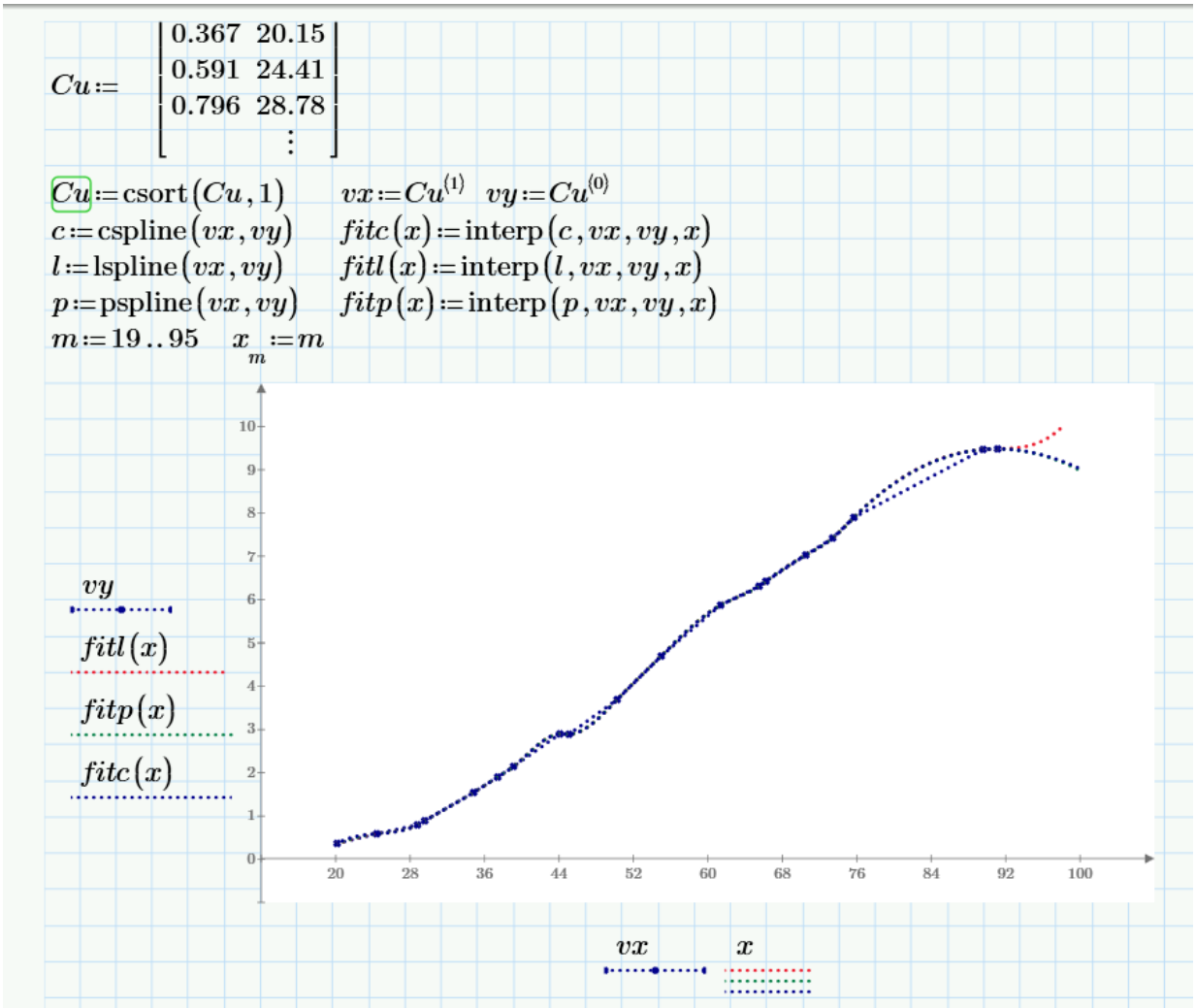


Рис. 7.2. Реалізування пунктів 1–7 розглянутого алгоритму подано у MathCad-документі

Екстраполювання виконується на основі інформації про поведінку функції на відтинку $vx_i \in [x_{min}, x_{max}]$ і дає змогу прогнозувати значення функції праворуч від крайньої точки відтинка, де вона означена.

Реалізування пунктів 8–10 розглянутого алгоритму подано у MathCad-документі на рис. 7.3.

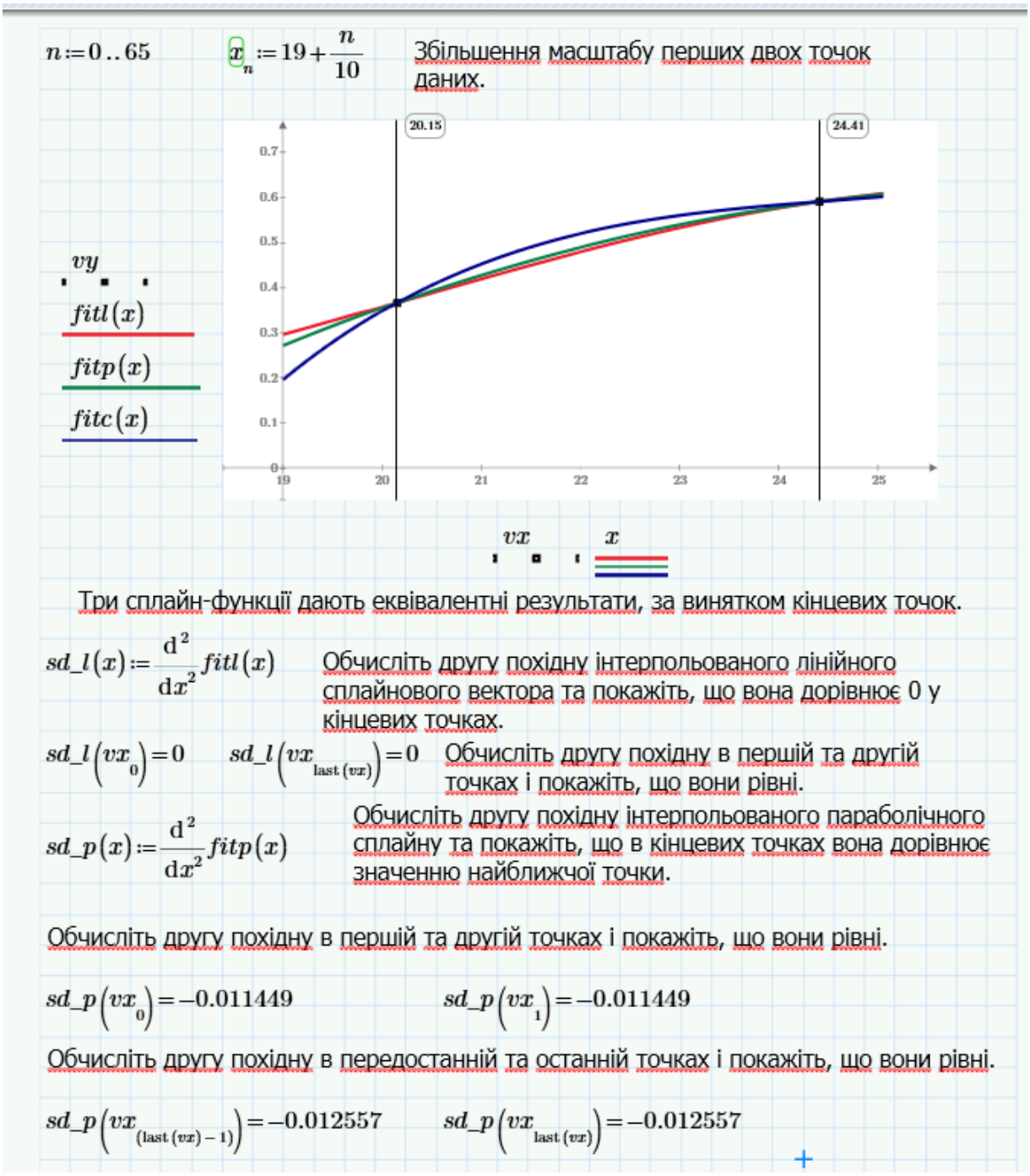


Рис. 7.3. Реалізування пунктів 8–10 розглянутого алгоритму подано у MathCad-документі

У Mathcad Prime для екстраполювання зазвичай використовують функцію

predict(vy, m, n),

де *vy* – вектор відомих значень функції (вектор *vy* значень аргументів у ній явно не задається і вважається, що на відтинку означення функції точки розподілені рівномірно);

m – кількість елементів вектора *vy*, на основі яких розраховуються прогнозовані значення функції (тобто вибираються *m* останніх точок, які наближені до правої границі x_{max});
n – бажана кількість прогнозованих значень функції.

Функція *predict(vy, m, n)* повертає вектор *n* прогнозованих значень функції для рівновіддалених значень аргументів, які припасовуються праворуч від правої межі x_{max} .

Вона забезпечує прогнозування значень за межами діапазону початкових даних на основі лінійної предиції (прогнозування).

Використання функції *redict.* Функція *predict(v, m, n)* використовує лінійний алгоритм прогнозування для передбачення наступних значень на основі існуючого ряду даних. Хоча *predict* є основною функцією *Engineering Notebook* для екстраполювання, також можна використовувати функції інтерполювання, такі як *linterp* (лінійне інтерполювання) або функції кубічного сплайн-інтерполювання (*cspline*, *pspline*, *lspline* з наступним використанням *interp*), якщо точка екстраполювання обчислюється близько до відомого діапазону.

Проте слід бути обережним: функції інтерполювання призначені перш за усе для обчислення значень *у межах* заданого діапазону даних. Їх застосування для екстраполювання (поза діапазоном) може призвести до менш точних або компрометованих результатів, залежно від характеру даних.

Приклади використання функції *predict.* Для прикладу екстраполювання у Mathcad Prime використаємо функцію *predict*.

Приклад 1. Припустимо, у нас є масив значень, які є вимірюваннями за певний час (наприклад, виторги за останні 10 місяців), і ми хочемо

спрогнозувати виторги на наступні 3 місяці. Результат обчислень із такими початковими даними подано на (рис. 7.4):

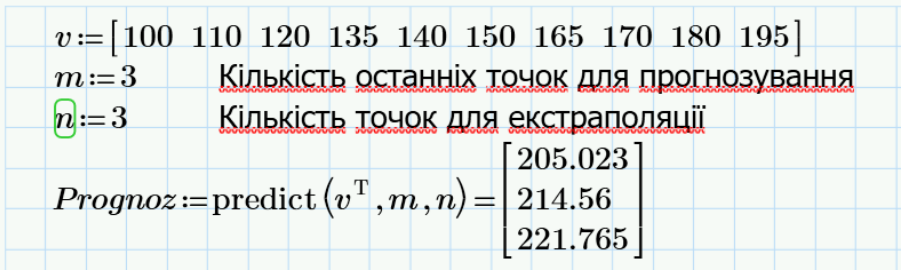


Рис. 7.4. Результати використання функції *predict* (приклад 1)

Щоб зрозуміти, як працює функція, визначимо часовий ряд та кількість попередніх і майбутніх значень. Функція прогнозування повинна обчислювати ваговий коефіцієнт для кожного апріорного значення, яке використовується для прогнозів. Для n невідомих функція прогнозування потребує n рівнянь для роботи. Вона формує рівняння на основі наступної моделі прогнозування: де X – це часовий ряд, а c – вектор вагових коефіцієнтів. Вагові коефіцієнти розраховуються за допомогою методу, відомого як метод Бурга. Функція прогнозування тепер може оцінювати майбутні значення. Процес виконання обчислень подано на рис. 7.5.

Збільшення або зменшення кількості точок у часовому ряді впливає на повернені прогнозовані значення, оскільки *predict* використовує усі значення X для обчислення вагових коефіцієнтів, які використовуються для лінійного прогнозування.

Повідомлення про помилки, які генеруються функцією *predict,* часто зумовлені її аргументами. В одному випадку повідомлення про помилку пов’язане з самим алгоритмом розрахунку (рис 7.6). Прогнозовані значення

не можуть бути лінійною функцією всіх точок даних. Ви можете використовувати до *(n - 1)* точок даних:

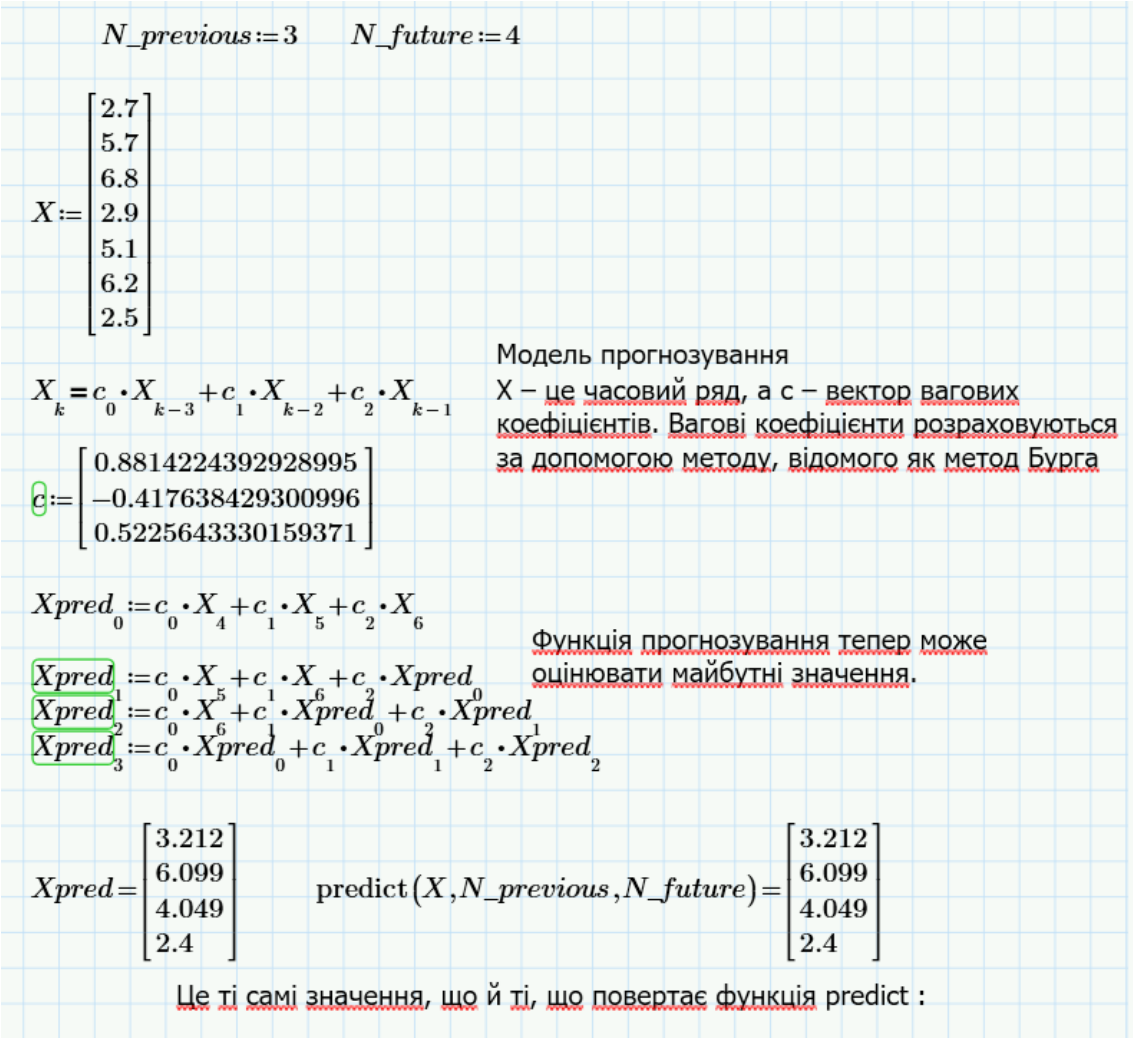


Рис. 7.5. Процес виконання обчислень функції прогнозування *predict*

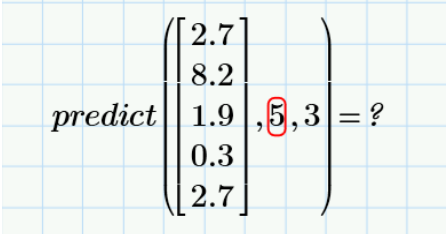


Рис. 7.6. Генерування помилки компілятором, яка пов’язана з алгоритмом

Приклад 2. Використовуємо функцію *predict* для обчислення прогнозованих значень.

Engineering Notebook має більш розвинений інструмент екстраполювання, який враховує розподіл даних вздовж усього інтервалу. У функцію *predict* вмонтований лінійний алгоритм передбачення поведінки функції, який ґрунтується на аналізі, у тому числі осциляції:

- *predict(y, m , n)* – функція передбачення вектора, який екстраполює вибірку даних:
- *y* – вектор дійсних значень, взятих через рівні проміжки значень аргументу;
- *m* – кількість послідовних елементів вектора, згідно з якими будується екстраполяція;
- *n* – кількість елементів вектора передбачень.

Приклад використання функції прогнозування осцилюючих даних y_j з мінливою амплітудою подано на рис. 7.7.

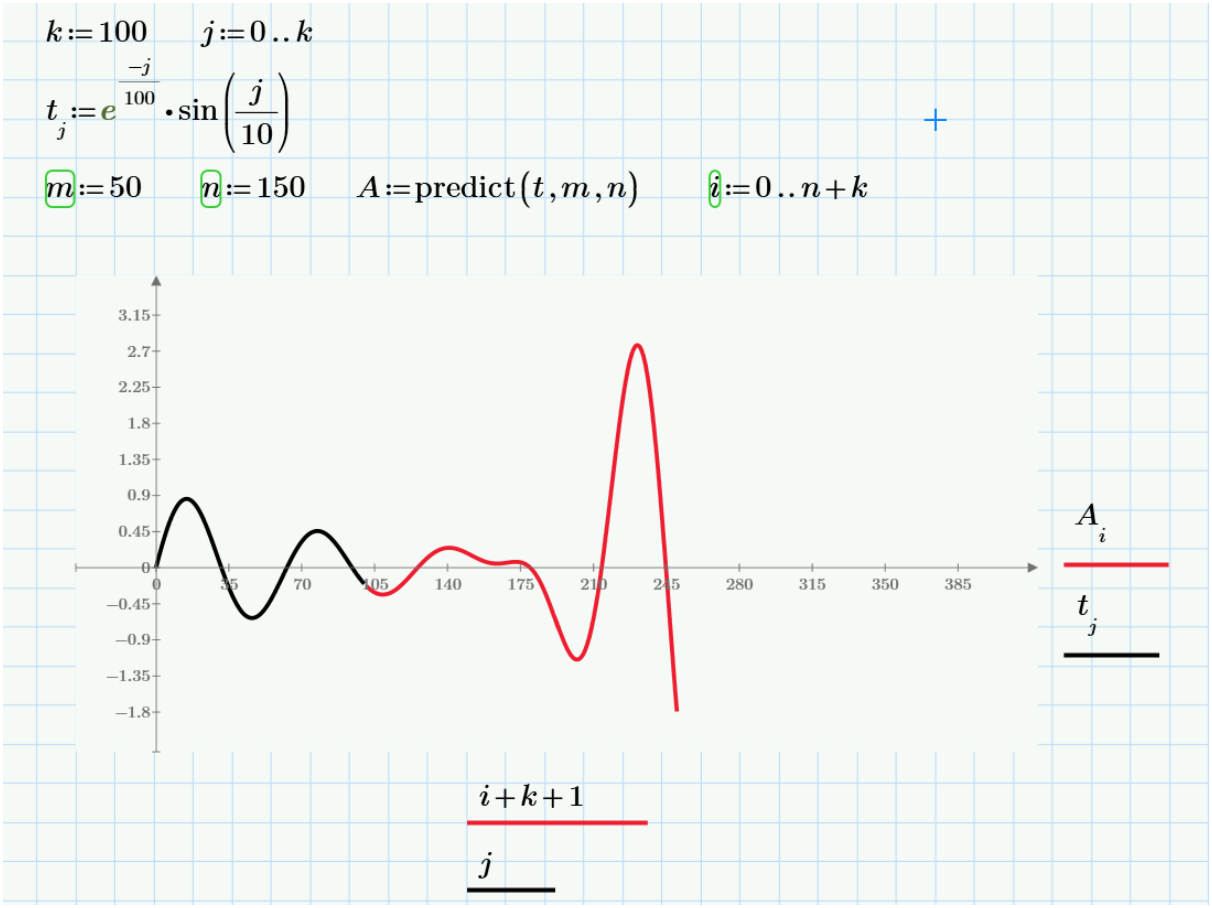


Рис. 7.7. Використання функції прогнозування осцилюючих даних y_j з мінливою амплітудою

Отримано графік екстраполювання, поряд із самою функцією. Аргументи та принцип дії функції *predict* відрізняються від розглянутих вище вмонтованих функцій інтерполювання/екстраполювання.

Значення аргументу для даних не потрібне, оскільки за визначенням функція працює з даними, які йдуть один за одним з рівномірним кроком. Зауважте, що результат функції *predict* вставляється «у хвіст» початкових даних.

Приклад 3. Прогнозування лінійних даних. Означимо лінійний набір даних та побудуємо графік лінійного набору даних (рис. 7.8):

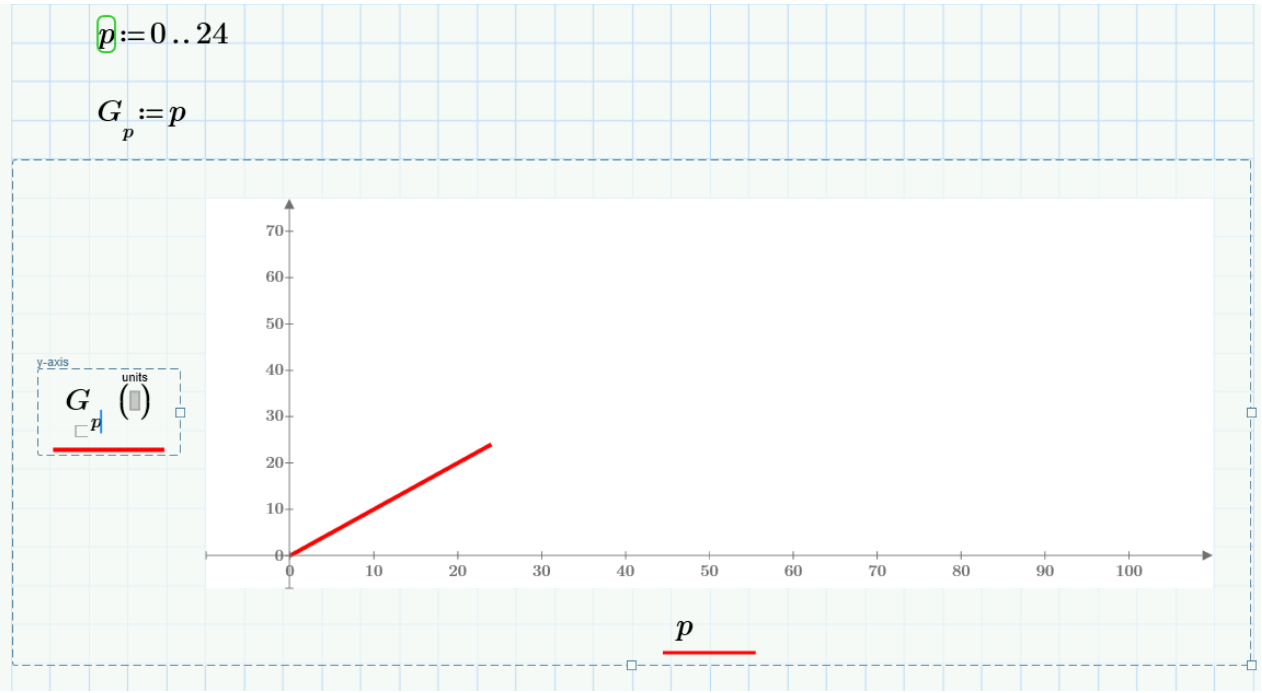


Рис. 7.8. Означення лінійного набору даних та його графік

Тепер обчислимо наступні 50 точок функції та використаємо функцію *predict* для екстраполювання наступних 100 точок даних. Потім додамо розраховані та екстрапольовані точки даних до початкового графіка, після чого порівняємо отримані результати (рис. 7.9).

На початковому етапі лінійне прогнозування дає адекватні результати. Проте, далі функція прогнозування забезпечує періодичність, що не відповідає дійсності. Загалом не рекомендується прогнозувати занадто далеко вперед.

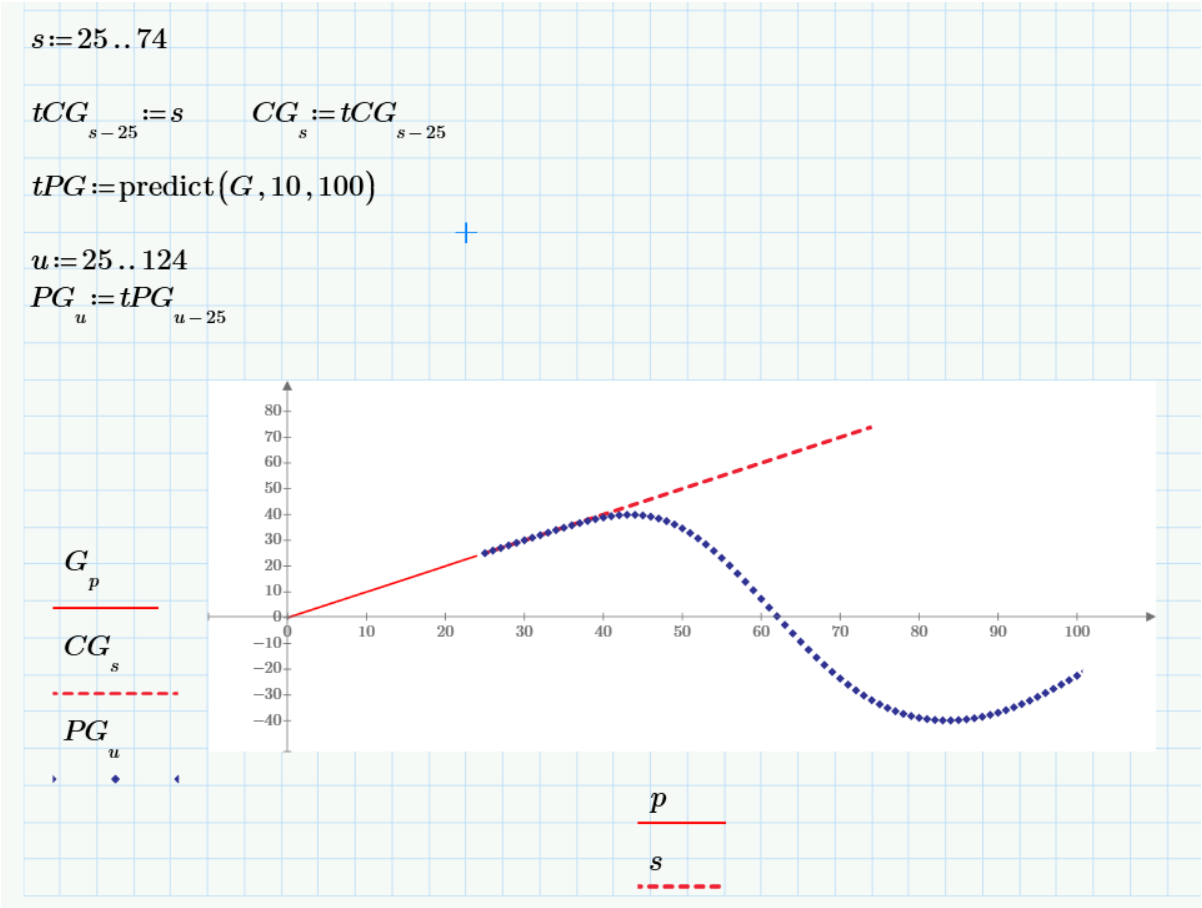


Рис. 7.9. Розраховані та екстрапольовані точки даних (приклад 3)

Приклад 4. Прогнозування періодичних даних. Використаємо функцію *predict*, щоб обчислити майбутні значення періодичних наборів даних.

Скористаємо з функцій *sin* та *cos* для означення періодичного набору даних. Побудуємо графік періодичного набору даних.

Обчислимо наступні 20 точок функції.

Використаємо функцію *predict* для екстраполювання наступних 20 точок даних. Додамо розраховані та екстрапольовані точки даних до початкового графіка, а потім порівняємо результати (рис. 7.10).

Як видно з графіка, функція прогнозування *predict* дуже добре працює з періодичними даними. Це пояснюється тим, що вона залежить від автокорелювання.

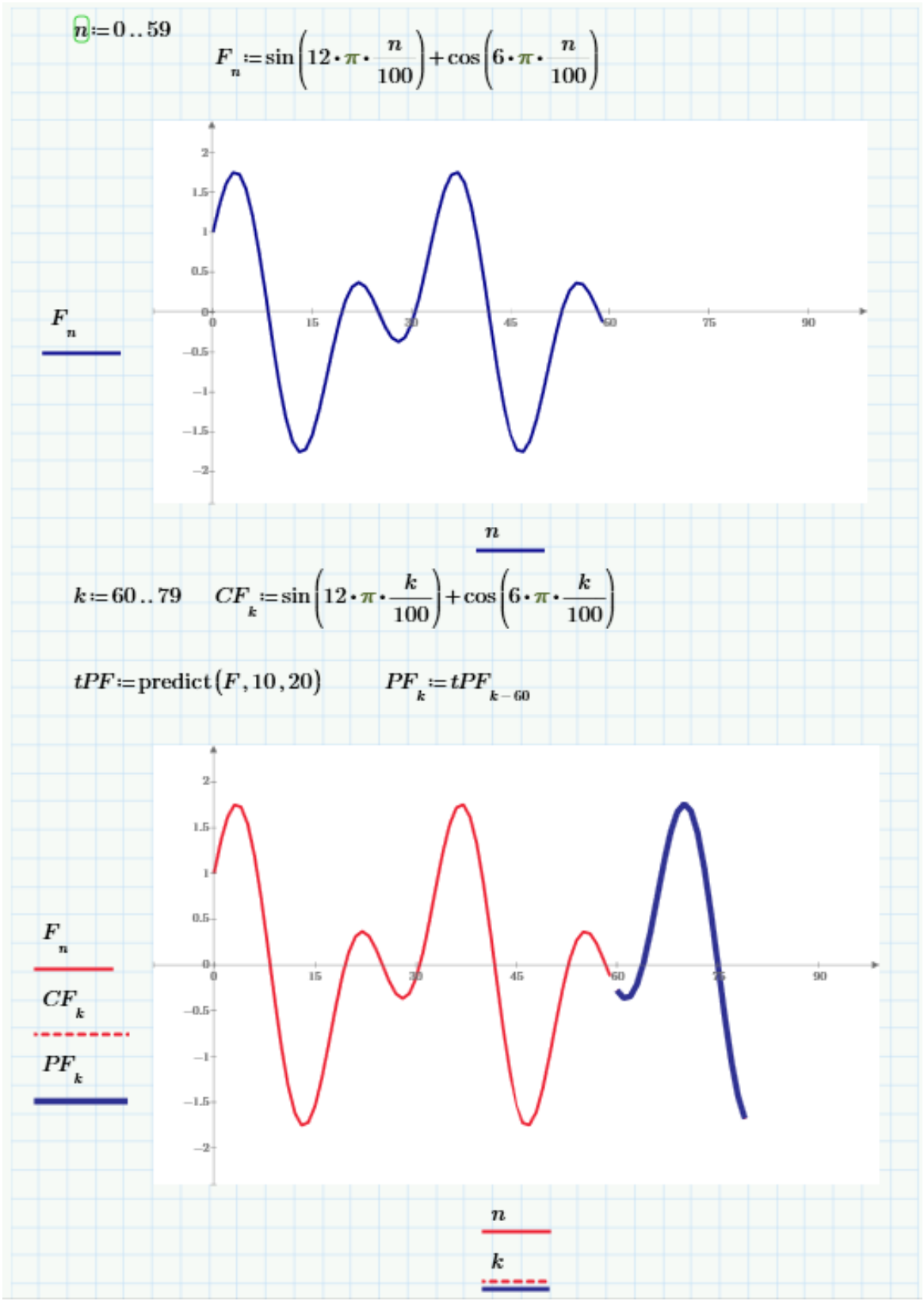


Рис. 7.10. Прогнозування періодичних даних (приклад 4)

РОЗДІЛ 8

РОЗВ'ЯЗАННЯ ЗВИЧАЙНИХ ДИФЕРЕНЦІЙНИХ РІВНЯНЬ

Здебільшого аналіз процесів у динамічних об'єктах різних галузей науки та техніки традиційно зводиться до обчислення розв'язку диференційних рівнянь. Значна кількість різних за видом диференційних рівнянь (або систем таких рівнянь) аналітично не розв'язуються, а у випадках, коли аналітичний розв'язок можливий, для його отримання потрібні ґрунтовні знання з відповідних розділів математики і відчутні витрати часу. Прямих універсальних засобів (спеціалізованих функцій) для аналітичного розв'язування диференційних рівнянь у пакеті MathCad Prime немає.

Тому особливої ваги у розв'язуванні звичайних диференційних рівнянь (ЗДР) та їх систем у пакеті MathCad Prime набувають саме числові методи. Для цього у *Engineering Notebook* існує низка вмонтованих функцій, які реалізовують різні методи (алгоритми) для числового інтегрування диференційних рівнянь та їх систем. Кожна з цих функцій має свою сферу застосування і є придатною (раціональною, доцільною) для обчислення розв'язку певних видів диференційних рівнянь та їхніх систем.

Розв'язати диференційне рівняння – означає отримати таку невідому функцію, яка стоїть під знаком похідної, що задовольняє рівняння в усіх точках вказаного інтервалу зміни її аргументів, разом з початковими умовами.

Така задача на початкові умови (*initial value problem*) називається задачею Коші і полягає у обчисленні розв'язку на основі інтегрування диференційного рівняння з урахуванням заданих початкових умов.

Для розв'язання диференційних рівнянь у часткових похідних у MathCad є можливість розв'язувати рівняння з двома незалежними змінними: одновимірні параболічні та гіперболічні рівняння, такі як рівняння теплопровідності, дифузії, хвильові рівняння, а також двомірні еліптичні рівняння (рівняння Пуассона та Лапласа).

Незважаючи на різні методи обчислення розв'язку, кожна з цих функцій вимагає, щоб були задані принаймні наступні значення, необхідні для обчислення розв'язку:

- початкові умови;
- набір точок, у яких потрібно обчислити розв'язок;
- диференційне рівняння, записане у певному спеціальному вигляді.

Щоб розв'язати ЗДР безпосередньо без створення блоку розв'язання, скористайтеся однією з вмонтованих функцій, які розв'язують системи ЗДР такого вигляду:

$$\frac{d}{dx}y = \begin{bmatrix} f_1(x,y) \\ f_2(x,y) \\ \cdots \\ f_n(x,y) \end{bmatrix},$$

де *y* – вектор невідомих функцій незалежної змінної *x*.

Щоб розв'язати ЗДР вищого порядку, перепишемо його як систему ЗДР першого порядку.

8.1. Вмонтовані функції для розв'язання ЗДР

Вмонтовані функції *Engineering Notebook* для розв'язання ЗДР поділяються на два типи: функції для жорстких систем та функції для нежорстких систем. Система ЗДР, записана у матричній формі як *y'=Ay*, називається жорсткою, якщо матриця *A* майже сингулярна. В іншому випадку система є нежорсткою.

Для розв'язання диференційних рівнянь із початковими умовами пакет *Engineering Notebook* має ряд вмонтованих функцій, які доступні у вкладці *Functions → Differential Equations* (рис. 8.1).

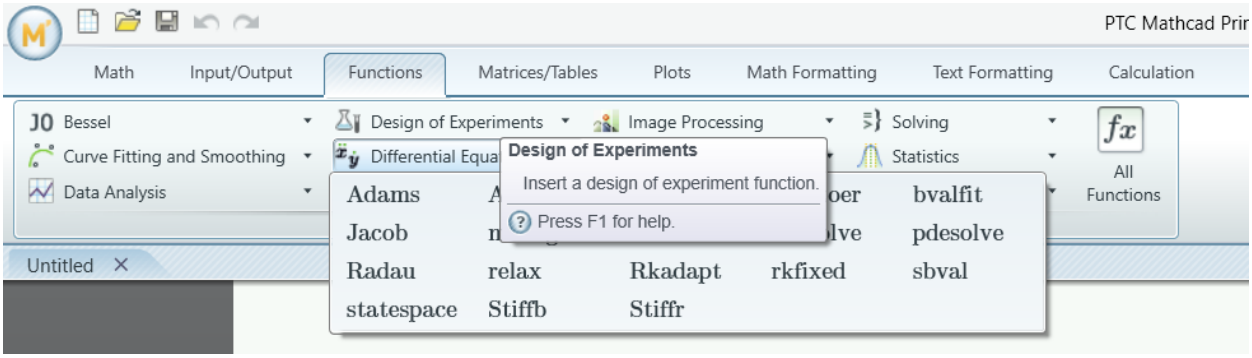


Рис. 8.1. Вибір функції для розв'язання диференційних рівнянь, які доступні у вкладці *Functions → Differential Equations*

Найбільш уживаними є наступні функції для розв'язання диференційних рівнянь:

- *Adams* – метод Адамса-Башфорда;
- *rkfixed, Rkadapt, Bulstoer* – метод Рунге-Кутти 4-го порядку з фіксованим та адаптивним розміром кроку, а також метод Бульстера для плавно змінних ЗДР;
- *BDF* – методи формул зворотного диференціювання;
- *Radau, Stiffb, Stiffr* – методи Radau, Bulirsch-Stoer і Rosenbrock для жорстких систем;
- *AdamsBDF* – визначає, чи є система жорсткою, чи нежорсткою та, відповідно, викликає *BDF* або *Adams*;
- *statespace* – системи лінійних звичайних диференціальних рівнянь першого порядку;
- *bvalfit, sbval* – крайові задачі, де не всі початкові умови відомі, перетворені на задачі з початковими значеннями за допомогою лінійної стрільби;
- *Odesolve* – функція, що розв'язує ЗДР блоковим способом.

Також існує безліч спеціальних генераторів поліномів та гіпергеометричних функцій, які розв'язують специфічні, поширені звичайні диференціальні рівняння.

Функції *Radau* та *Rkadaptu* використовують неоднорідні розміри кроків внутрішньо, коли вони розв'язують диференційне рівняння, додаючи більше кроків у областях більшої варіації розв'язку, але повертають розв'язок у кількості рівновіддалених точок, означеній у *intvls* .

Перевагою використання *Radau* є те, що не потрібні початкові дані Якобіана, хоча якщо *J* легкодоступний, його використання, як правило, підвищує точність.

8.2. Розв'язання звичайних диференційних рівнянь з використанням вмонтованої функції *rkfixed*

Функція *rkfixed* використовує для обчислення розв'язку чисельний метод Рунге-Кутта четвертого порядку. Звертання до функції *rkfixed* має вигляд:

rkfixed(y, x1, x2, n, D),

де *y* – вектор початкових умов розмірністю *n* (містить початкові значення);
n – кількість точок розв'язку на інтервалі інтегрування, що визначає *1+n* рядків результуючої матриці, яку повертає функція *rkfixed*. Значення для *n* вибирають з умови отримання бажаної точності та стійкості числового інтегрування;
x1, *x2* – початкова і кінцева точки інтервалу інтегрування;
D – функція-вектор, у кожному з *n* рядків якої записуються вирази перших похідних інтегральних функцій y_i (обчислюваних розв'язків), які є правими частинами початкових диференційних рівнянь, що записані у нормальній формі Коші. Ця вектор-функція оголошується як функція двох змінних *D(x, y)*, де *x* – незалежна змінна, за якою виконується інтегрування;
y – вектор з елементами y_i *(i = 0, 1, 2, ... , n)* – функціями, які описують праві частини диференційних рівнянь, що записані у нормальній формі Коші.

Приклад розв'язання задачі Коші методом Рунге-Кутта з фіксованим кроком на прикладі диференційного рівняння

$$\frac{d}{dx}y = \frac{y}{x} + x^2$$

за початкової умови *y(1) = 0* на відтинку [1, 5] із застосуванням *Solve Block* подано на рис. 8.2.

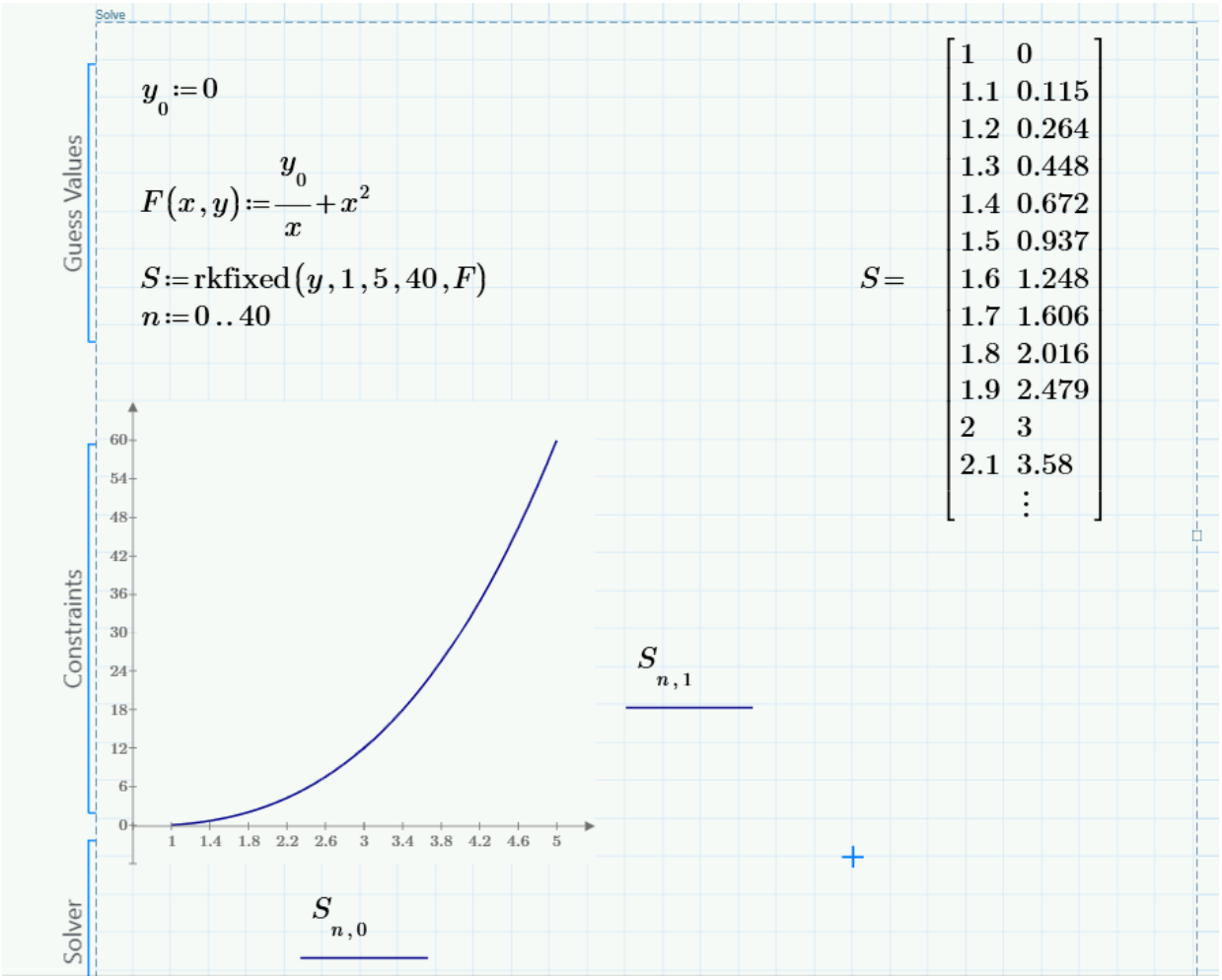


Рис. 8.2. Приклад розв’язання задачі Коші методом Рунге-Кутта з фіксованим кроком із застосуванням *Solve Block*

Описані вище і проілюстровані прикладами два основні способи розв’язування диференційних рівнянь та їхніх систем – за допомогою спеціалізованих функцій та *Solve Block* – мають приблизно однакові можливості, бо використовують вони однакові числові методи. Зокрема подання диференціального рівняння *n*-го порядку в *Solve Block* звичайною формою його записування або системи диференційних рівнянь, яка не зведена до нормальної форми Коші, є звичнішим та наочнішим для пересічного користувача.

Потрібно зазначити, що диференційне рівняння *n*-го порядку можна звести до системи диференційних рівнянь першого порядку і розв'язати її довільною із вказаних вище функцій *Engineering Notebook*. Щоправда, останнє вимагає попереднього виконання відповідних аналітичних перетворень для зведення системи диференційних рівнянь до нормальної форми Коші. Тому використання у кожному конкретному випадку вмонтованих функцій або *Solve Block* визначається переважно початковою формою подання диференційних рівнянь у конкретній задачі і певною мірою уподобаннями користувача.

Розглянемо приклад 1. Одержати числовий розв'язок диференційного рівняння

$$\frac{ty'e^t}{10} + y = 2 \cdot \mathrm{y} \cdot \ln(t) \cdot sin(t)$$

для *y(1)* = 5 на відтинку [1, 6] з кроком *h* = 0.25. Для отримання розв'язку використаємо функцію *rkfixed*. Результати розв'язання задачі (приклад 1) подано на рис. 8.3.

Приклад 2. Одержати розв'язок системи диференційних рівнянь

$$\begin{cases} \dfrac{dq}{dt} = ysin(2t) - zcos(q) \\ \dfrac{dz}{dy} = sin(q) + cos(z) + 2e^{0.2t} \end{cases}$$

Для початкових умов *y(0) = 1*, *z(0) = 0* на відтинку [*0*, *2*] з кроком *h = 0.1*. Розв'язок отримати з використанням двох функцій: *Bulstoer* та *Rkadapt* і порівняти розв'язки, які повертаються цими функціями, за точністю.

Усі описані вище внутрішні функції *Engineering Notebook* для числового інтегрування систем диференційних рівнянь вимагають їх зведення та подання у нормальній формі Коші.

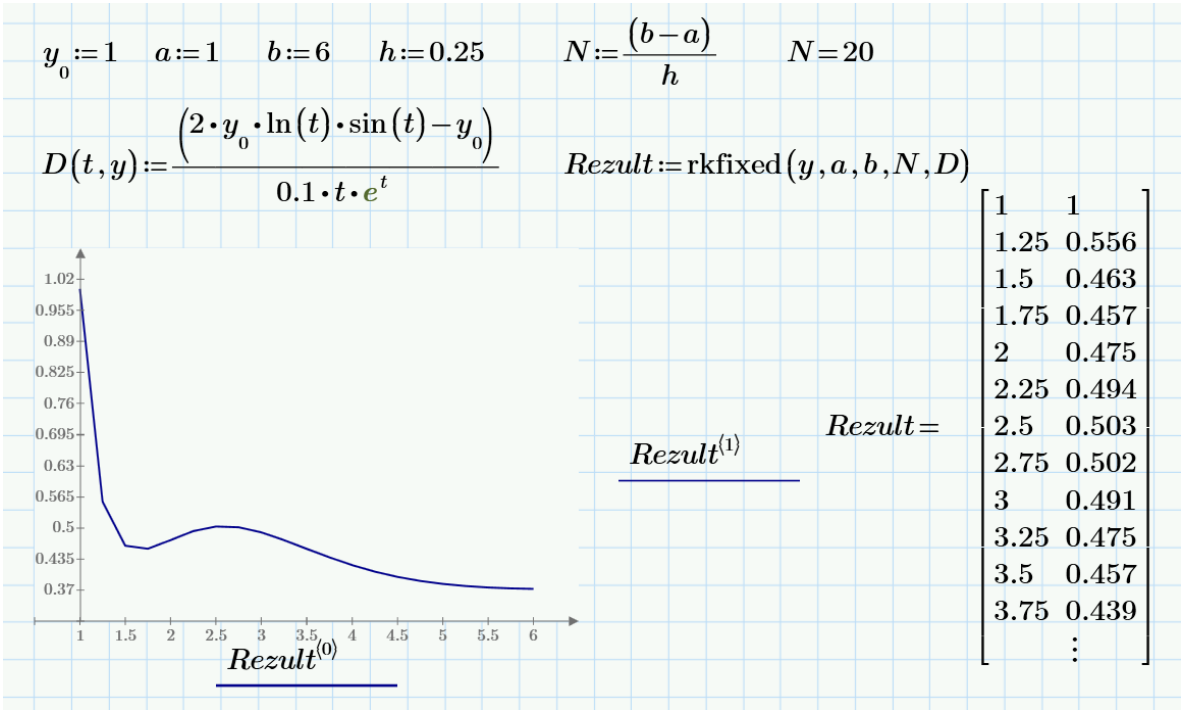


Рис. 8.3. Результати розв'язання задачі з використанням функції *rkfixed* (приклад 1)

Результати розв'язання задачі (приклад 2) подано на рис. 8.4. Як видно з рисунку, результати розв'язання системи диференційних рівнянь з використанням функцій *Bulstoer* та *Rkadapt* є практично ідентичними.

Приклад 3. Розв'язання системи ЗДР першого порядку з використанням *Solve Block* і функції *Odesolve*.

Функція *Odesolve(vf, b, [intvls])* визначає функцію, що є розв'язком системи звичайних диференціальних рівнянь, яка залежить від початкового наближення та граничних умов. Елементи рівнянь, які містять похідні вищого порядку, повинні бути лінійними, а кількість початкових та граничних умов має дорівнювати порядку рівнянь.

Аргументи: *vf* – функція або вектор-стовпець, що містить функції в тому порядку, як вони задані у *Solve Block*.

При визначенні функцій у векторі *vf* необхідно вказувати перелік аргументів. Наприклад, якщо обчислюється розв'язок для функцій *f(t)* і *g(t), b* може бути більшою або меншою від початкового значення, заданого у *Solve Block*; *intvls*

(необов'язковий) – ціле число, кількість інтервалів дискретизації, що використовується під час інтерполювання функції розв'язку $\begin{bmatrix} f(t) \\ g(t) \end{bmatrix}$.

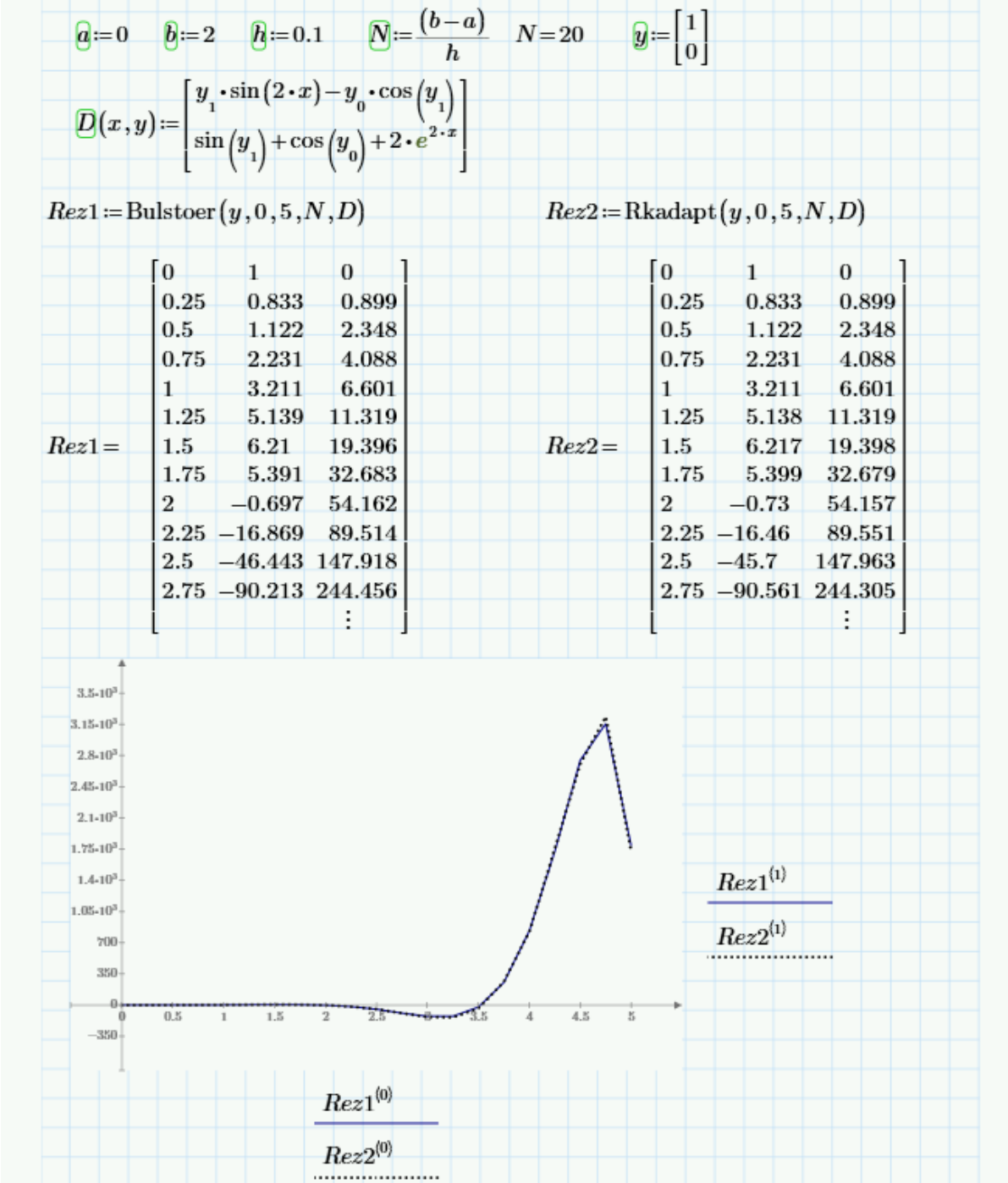


Рис. 8.4. Результати розв'язання задачі з використанням функцій *Bulstoer* та *Rkadapt* (приклад 2)

Кількість точок розв'язків дорівнює кількості інтервалів + 1. За замовчуванням *intvls* має значення 1000. Значення за замовчуванням змінної

intvls досить велике, щоб отримати точний інтерпольований розв'язок, але ще можна збільшити це значення, щоб у розв'язку були відтворені дрібні деталі.

У цьому випадку функції *Odesolve* потрібно зберігати більше точок інтерполювання, що може призвести до збільшення часу обчислення. При розв'язанні ЗДР для великого інтервалу встановіть значення *intvls* більшим, ніж за замовчуванням.

У довільному випадку кінцеві точки, які використовуються у граничних умовах, повинні відповідати кінцевим точкам, вказаним у функції *Odesolve*. *Engineering Notebook* перевіряє правильність типу та кількості умов і генерує помилку при виявленні невідповідності. Результат обчислень функції *Odesolve* необхідно присвоїти або імені функції, або вектору імен функцій, без аргументів. Нагадаємо, що функцію *Odesolve* можна використовувати тільки у *Solve Block.* Щоб отримати розв'язки системи звичайних диференційних рівнянь, виконаємо таку послідовність дій:

1. Вкажіть кінцеву точку інтервалу розв'язків $T1:=10$.

2. Визначте завдання, використовуючи похідні та набір початкових наближень.

3. Побудуйте графік розв'язків на одному інтервалі у спільній області побудови (рис.8.5).

Приклад 4. Використання функції *Rkadapt*. Використовуйте функцію *Rkadapt* для розв'язання системи диференційних рівнянь, яка використана у попередньому прикладі. Використаємо таку послідовність команд для розв'язання задачі:

1. Вкажіть функцію, яка визначає вектор значень похідної у довільній точці розв'язання (t, Y).

2. Вкажіть додаткові аргументи для розв'язання ЗДР функцією *Rkadapt*: a – початкове значення незалежної змінної; b – вектор початкових значень функції; c – кількість значень розв'язків на інтервалі [*t0*, *t1*].

3. Скористайтеся функцією *Rkadapt* для обчислення матриці розв'язків.

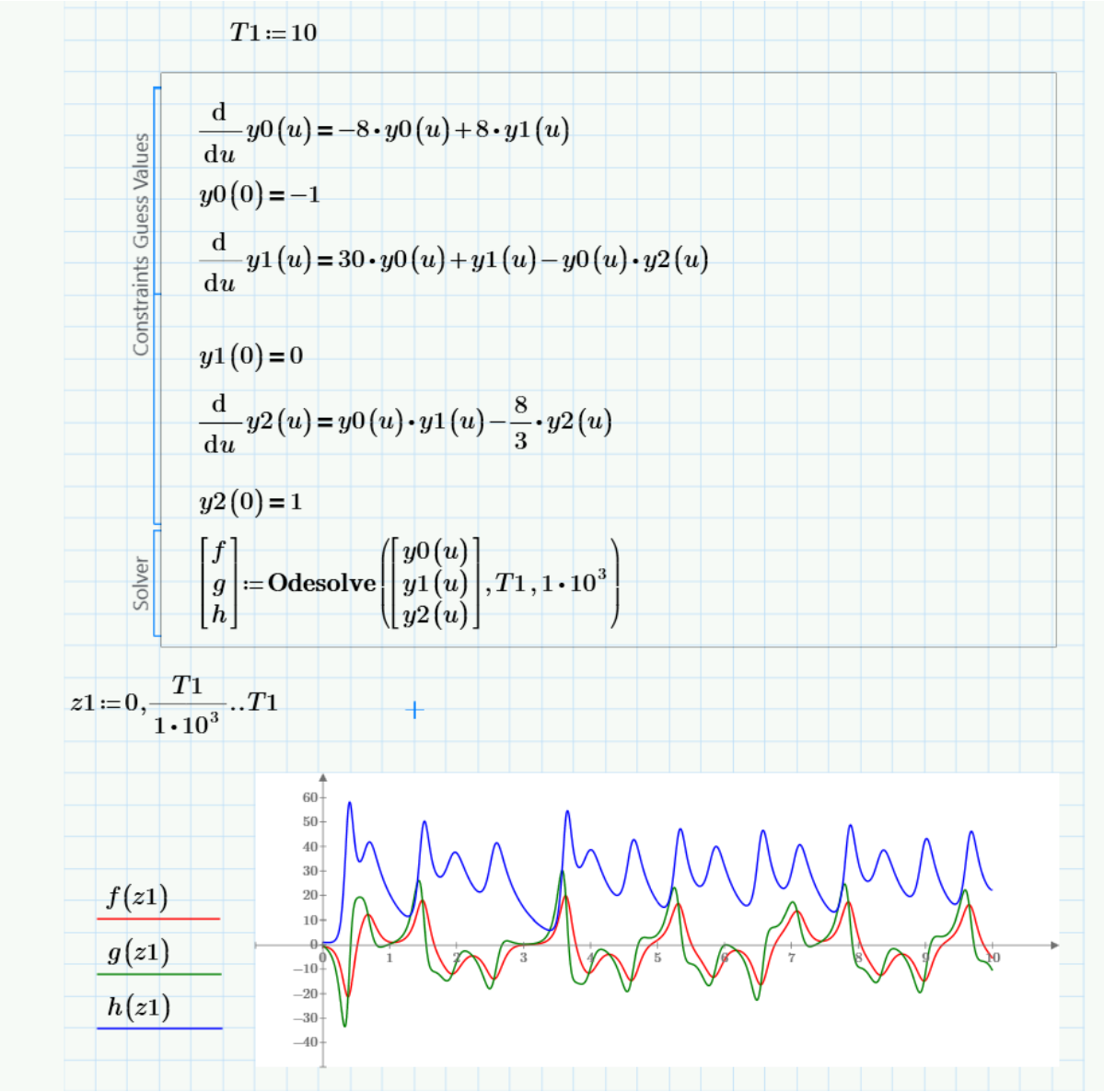


Рис. 8.5. Приклад розв'язання системи ЗДР першого порядку з використанням *Solve Block* і функції *Odesolve*

4. Отримайте значення незалежної змінної (стовпець з індексом 0).

5. Отримайте значення першої функції розв'язку (стовпець з індексом 1).

6. Отримайте значення другої функції розв'язку (стовпець з індексом 2).

7. Отримайте значення третьої функції розв'язку (стовпець з індексом 3).

8. Побудуйте графіки всіх трьох розв'язків у спільній області.

Приклад використання функції *Rkadapt* для розв'язання системи диференційних рівнянь (приклад 4) подано на рис. 8.6.

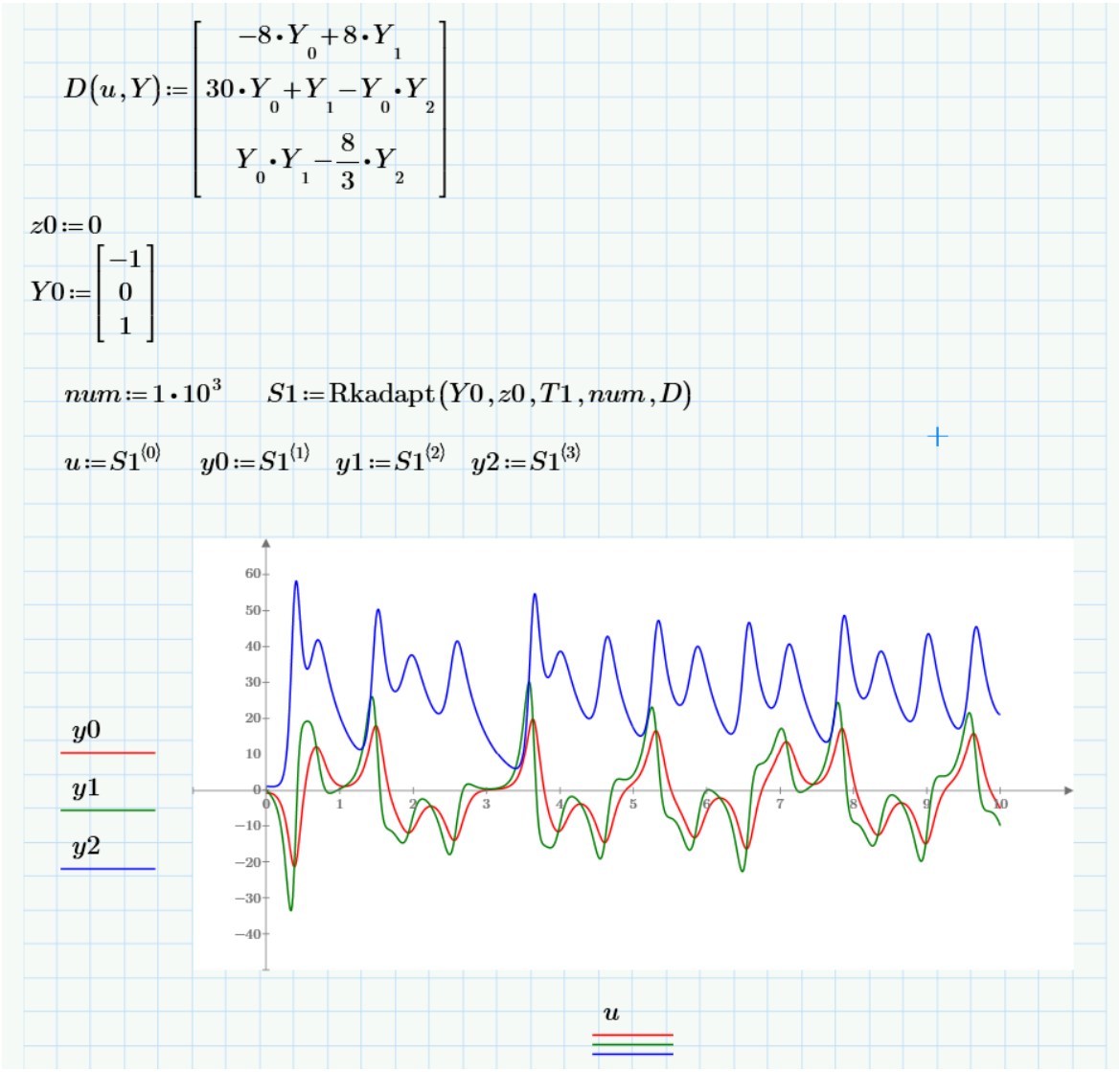


Рис. 8.6. Приклад використання функції *Rkadapt* для розв’язання системи диференційних рівнянь (приклад 4)

ЛІТЕРАТУРА

1. Литвинов А. Л. Чисельні методи: теорія і практика : навчальний посібник. Харків : ХНУМГ ім. О. М. Бекетова, 2022. 166 с.
2. Розрахунки і програмування у системі Mathcad Prime : навчальний посібник для студентів хімічних спеціальностей / Л. В. Соловей, Н. М. Мірошніченко, А. М. Миронов, М. В. Ільченко. Харків : НТУ «ХПІ», 2022. 184 с.
3. Волонтир Л. О., Зелінська О. В., Потапова Н. А., Чіков І. А. Чисельні методи : навчальний посібник. Вінниця : ВНАУ, 2020. 322 с.
4. Гончаров О. А., Васильєва Л. В., Юнда А. М. Чисельні методи розв'язання прикладних задач : навчальний посібник. Суми : Сумський державний університет, 2020. 142 с.
5. Гребенюк С. М., Гоменюк С. І. Чисельні методи розв'язання механічних задач : навчальний посібник для здобувачів третього освітньо-наукового рівня спеціальності «Прикладна математика» освітньо-наукової програми. «Прикладна математика». Запоріжжя : ЗНУ, 2022. 80 с.
6. Чисельні методи : конспект лекцій / О. В. Шебаніна, С. І. Тищенко, І. І. Хилько, В. О. Крайній. Миколаїв : МНАУ, 2023. 100 с.
7. URL: https://support.ptc.com/help/engineering_notebook/r11.0/en/index.html#page/PTC_Mathcad_Help/ode_solvers.html#wwID0EQ4T5
8. Практикум із чисельних методів : навч. посіб. / Н. І. Полтораченко, С. А. Теренчук, Ю. Н. Убайдуллаєв. Київ : КНУБА, 2023. 160 с.
9. Ремез Н. С., Кисельов В. Б., Дичко А. О., Мінаєва Ю. Ю. Чисельні методи розв'язання технічних задач : підруч. для ЗВО. Одеса : Гельветика, 2022. 185 с.
10. Андруник В. А., Висоцька В. А., Пасічник В. В., Чиркун Л. Б., Чиркун Л. В. Чисельні методи : навчальний посібник. Львів : Видавництво «Новий Світ-2000», 2020. 470 с.
11. Король І. Ю., Тютюнникова Г. С. Алгоритми та методи обчислень : навчальний посібник для студентів 3-го курсу інженерно-технічного

факультету, спеціальності «Комп’ютерна інженерія». Ужгород : видавництво ПП «АУТДОР-ШАРК», 2021. 124 с.

12. Абакумова О. О. Чисельні методи : лабораторний практикум. КПІ ім. І. Сікорського, 2020. 95 с.
13. Гребенюк С. М., Гоменюк С. І. Чисельні методи розв’язання механічних задач : навчальний посібник для здобувачів третього освітньо-наукового рівня спеціальності «Прикладна математика» освітньо-наукової програми. «Прикладна математика» . Запоріжжя : ЗНУ, 2022. 80 с.
14. Вища математика. Числові методи : методичні рекомендації до самостійної роботи для студентів технічних спеціальностей / уклад. : І. О. Ластівка, В. К. Репета, О. Д. Глухов. К. : НАУ, 2020. 56 с.
15. Рудий Т. В., Паранчук Я. С., Сеник В. В. Алгоритмізація та програмування : навчальний посібник. Частина 1. Структурне програмування. Львів : Львівський державний університет внутрішніх справ, 2023. 240 с.
16. Рудий Т. В., Паранчук Я. С., Сеник В. В. Алгоритмізація та програмування : навчальний посібник. Частина 2. Модульне програмування. Львів : Львівський державний університет внутрішніх справ, 2024. 172 с.
17. Ришковець Ю. В., Висоцька В. А. Алгоритмізація та програмування. Частина 1 : навчальний посібник. Львів : Видавництво «Новий Світ-200», 2021. 337 с.
18. Ришковець Ю. В., Висоцька В. А. Алгоритмізація та програмування. Частина 2 : навчальний посібник. Львів : Видавництво «Новий Світ-200», 2021. 315 с.
19. Ментинський С. М., Пелех Я. М. Основи програмування на С++ : навчальний посібник з курсу «Основи інформатики і програмування, частина 2» спеціальності 105 – «Прикладна фізика та наноматеріали» для першого (бакалаврського) рівня освіти. Львів : Галицька Видавнича Спілка, 2021. 256 с.
20. Алгоритмізація та програмування : спеціальність 122 «Комп’ютерні науки» / авт. Ю. С. Процик, Т. С. Самотій, М. В. Левкович. Львів : НЛТУ України, 2017. URL: http://vee.nltu.edu.ua/course/view.php?id=3

НАВЧАЛЬНЕ ВИДАННЯ

Рудий Тарас Володимирович,
кандидат технічних наук, доцент
(розділи 5–8)
Зачек Олег Ігорович,
кандидат технічних наук, доцент
(розділи 1–4)

Чисельні методи

Навчальний посібник

Редагування *Андріана Кузьмич-Походенко*
Макетування *Галина Шушняк*
Друк *Назарій Ганущак*

Підписано до друку 30.12.2025.
Формат 60×84/8. Умовн. друк. арк. 21,39.
Тираж 70 прим. Зам № 108-25.

Львівський державний університет внутрішніх справ
Україна, 79007, м. Львів, вул. Городоцька, 26.

Свідоцтво про внесення суб'єкта видавничої справи до державного реєстру
видавців, виготівників і розповсюджувачів видавничої продукції
ДК № 2541 від 26 червня 2006 р.