МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

*Ю. О. Космінська*

# КОМП'ЮТЕРНІ ТЕХНОЛОГІЇ В ЕЛЕКТРОНІЦІ

Конспект лекцій

Суми
Сумський державний університет
2014

Комп'ютерні технології в електроніці : конспект лекцій / укладач Ю. О. Космінська. – Суми : Сумський державний університет, 2014. – 149 с.

Кафедра наноелектроніки

# Зміст

# Вступ

Системи комп’ютерної математики (СКМ) є зручними і потужними інструментами для розв'язування інженерних та наукових математичних задач. Це пов'язано не лише з можливістю проведення числових обрахунків та графічного подання результатів, а й із виконанням символьних викладок та перетворень. Беручи до уваги економічні реалії сучасного світу, зрозуміло, що всебічний комп'ютерний аналіз будь-якої кількісної задачі зберігає багато робочого часу та матеріальних ресурсів. СКМ дозволяють студентам, інженерам та науковцям працювати над усіма аспектами математичного моделювання: від аналітичного виведення та перетворення модельних рівнянь до розв'язання цих рівнянь у числовому або аналітичному вигляді, побудови графіків чи анімації результатів. Однією з найбільш потужних сучасних СКМ є система Maple. У цьому конспекті розглядається версія системи Maple 12, яка є розвитком класичних попередніх варіантів системи і дозволяє працювати як у класичному, так і в більш сучасному стандартному інтерфейсі і відкриває більше можливостей та зручностей для користувача. Структура конспекту відповідає аудиторному лекційному курсу і відповідно містить такі основні розділи: "Основні відомості про систему Maple 12", "Математичні обчислення", "Елементи програмування", "Графіка", "Робота з файлами. Взаємодія Maple з іншими програмами".

Для студентів Maple 12 допоможе істотному розумінню різноманітних фізичних процесів та явищ, які вони вивчають. Цьому буде сприяти і робота над курсовою та лабораторними роботами з тематик комп'ютерного моделювання фізичних процесів, передбачених робочою програмою курсу "Комп'ютерні технології в електроніці". Цей конспект містить опис необхідного інструментарію для їх виконання.

# 1. Основні відомості про систему Maple 12

## 1.1. Стисла характеристика системи

Система Maple 12 (далі Maple) являє собою популярний математичний пакет у вигляді потужної обчислювальної системи, призначеної для виконання математичних розрахунків різної складності.

Maple є прикладом системи комп'ютерної математики (СКМ), тобто це не просто інструмент обчислень на зразок звичайного калькулятора. СКМ – це система, що може виконувати числові розрахунки, реалізуючи при цьому точну арифметику, арифметику чисел із плаваючою точкою; символьні перетворення; графічно відображати результати. Крім того, до складу Maple входить досить розвинене середовище програмування та засоби підготовки електронних документів професійної якості.

Maple містить надійні та ефективні символьні та числові алгоритми для розв’язання великого спектра математичних задач. Наприклад, Maple вміє виконувати складні алгебраїчні перетворення, розв’язувати числово та аналітично рівняння, нерівності, їх системи, включаючи диференціальні рівняння; Maple розв’язує задачі лінійної та тензорної алгебри, теорії груп, комбінаторики, статистики, найрізноманітніші задачі диференціального та інтегрального числення і т. ін. При цьому важливо, що користувач відразу працює в потужному інтерактивному документі, де легко бачити власні розрахунки, їх результати, подавати їх у різних виглядах та супроводжувати текстовими коментарями і описами.

Основним, але не єдиним принципом роботи в системі є використання команд у діалоговому режимі, тобто робота за схемою «введення команди – отримання відповіді системи». Основою Maple є ядро системи, до якого входять базові функції та алгоритми символьних перетворень. Також є основна

бібліотека команд, операторів і функцій-процедур, які готові до використання при початковому завантаженні. Крім того, існують команди та функції, які містяться у спеціалізованих пакетах (Packages) для розв'язання спеціалізованих задач, що підключаються додатковою командою `with(name)`, де name – ім'я пакета.

## 1.2. Структура системи

Система Maple являє собою інтегровану програмну систему, тобто вона містить багато компонентів:

– мови програмування, до яких відносять: вхідну мову інтерактивного спілкування із системою, мову процедурного програмування (Maple-мову), мову реалізації системи – C;

– зручний редактор для підготовки та редагування програм;

– багатовіконний інтерфейс користувача з можливістю роботи в діалоговому режимі;

– потужну довідкову систему з багатьма прикладами;

– словник математичних та інженерних понять і термінів з алфавітною організацією, довідник наукових констант та властивостей хімічних елементів;

– ядро алгоритмів та правил перетворення математичних виразів;

– числовий та символьний програмні процесори;

– систему діагностики;

– бібліотеки вбудованих та додаткових функцій;

– пакети розширення;

– засоби підтримки деяких мов програмування та інтеграції з поширеними програмами.

## 1.3. Початок роботи

Існує декілька інтерфейсів користувача для роботи в системі Maple 12: стандартний, класичний, командний рядок, графічний калькулятор та меплет-програми. Запуск кожного з них

відбувається стандартно через меню **Пуск** або через відповідний значок на робочому столі Windows.

### 1.3.1. Стандартний інтерфейс (Standart Worksheet)

Стандартний інтерфейс використовується для отримання максимальних можливостей системи Maple. У даному стандартному робочому листі користувач може створювати «живі» електронні документи, в яких наводяться текстові блоки або коментарі, розрахунки та їх результати або повідомлення про помилки, при цьому в будь-який момент можна змінити значення параметрів та перерахувати результати. Розширені можливості форматування дозволяють створити бажану структуру та вигляд документа.

На рисунку 1.1 наведено приклад стандартного вікна системи. Його основними елементами є такі:

- головне меню 1;
- панель інструментів 2;
- палітри команд 3;
- контекстна панель 4;

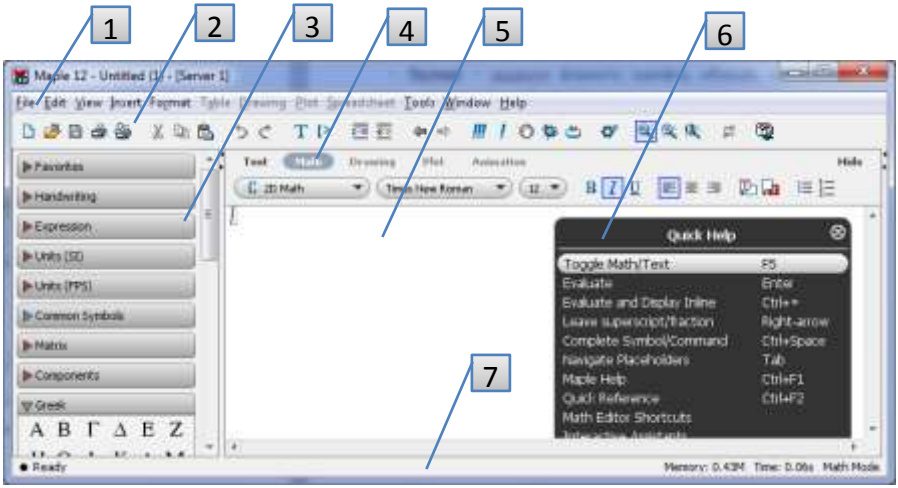



Рисунок 1.1 – Стандартне вікно Maple (Standart Worksheet)

– робоча область 5;
– віконце допомоги 6;
– рядок стану 7.

Крім зазначених вище елементів, існує ще контекстне меню, що викликається правим кліком у будь-якому місці робочого документа і зміст якого, як і контекстна панель 4, змінюється. Меню відкриває швидкий доступ до всіх можливих операцій, які можна застосувати для даного об’єкта.

Стандартний інтерфейс є головним для роботи в системі, тому в подальшому основна увага приділятиметься саме йому.

*1.3.1.1. Структура та можливості головного меню*

Головне меню містить найбільш повний набір команд керування системою. До нього входять такі пункти:

– **File** – робота з файлами, наприклад, створення, відкриття, збереження, експорт в інші формати, а також друкування документів;

– **Edit** – редагування тексту документа, наприклад, повтор дії, відміна дії, знайти/замінити, перехід за закладками та гіперпосиланнями, а також операції з даними буфера обміну Windows;

– **View** – настроювання вигляду інтерфейсу користувача, а саме: масштабування, відображення панелей інструментів, палітр-шаблонів та ін.;

– **Insert** – вставка в документ різних об’єктів, таких, як закладки, посилання, текст, мітки, таблиці, графіки, виконувані групи команд та ін.;

– **Format** – задання формату шрифту, абзаців, числових даних, перетворення форматів;

– **Table** – робота з об’єктами типу таблиця (меню доступне при активації об’єкта);

– **Drawing** – робота з рисунками та кресленнями (меню доступне при активації об’єкта);

– **Plot** – робота з графіками (меню доступне при активації об’єкта);

– **Spreadsheet** – робота з електронними таблицями (меню доступне при активації об’єкта);

– **Tools** – меню інструментів, до яких відносять помічники, підручники, шаблони розв’язання задач, підключення пакетів команд, опції настроювання системи та ін.;

– **Window** – класичні операції роботи з вікнами;

– **Help** – доступ до засобів потужної довідкової системи.

*1.3.1.2. Панель інструментів*

Ця панель (поз. 2 на рис. 1.1) містить кнопки найбільше використовуваних дій:

 – група кнопок для створення, відкриття, зберігання або друку робочого документа;

 – кнопки роботи з буфером обміну та повтору/відміни дії;

 – вставка з нового рядка відповідно текстового блоку типу plain text та знака запрошення введення команд (maple input);

 – кнопки організації документа у вигляді секцій та підсекцій;

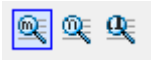
 – навігація між робочими документами за історією гіперпосилань;

 – виконати весь робочий документ або його виділену частину (аналогічно натисканню клавіші **Enter**);

 – зупинка або відлагодження поточної операції;

 – перезапуск системи Maple без закриття вікна, що приводить до очищення внутрішньої пам’яті системи (аналогічно команді **restart**);

 – редагування коду ініціалізації;

 – масштабування;

– переключення функції, яку виконує клавіша табуляції **Tab**: перехід між виконуваними групами команд (виконувана група – команди під одним знаком запрошення) або вставка табуляції в текст;

– виклик довідкової системи.

*1.3.1.3. Палітри команд*

Палітри команд (поз. 3 на рис. 1.1) є наборами шаблонів для вставки готових символів, виразів, операторів, матриць, активних компонентів, рукописного введення символів та ін. Використання шаблонів значно спрощує та прискорює роботу в системі, оскільки не потребує знання синтаксису тих чи інших команд. Для того щоб вставити відповідний об'єкт у документ, необхідно поставити курсор в потрібне місце документа та клікнути на кнопку на палітрі або просто перетягнути методом drag-and-drop.

**Приклад 1.1.** Розрахувати визначений інтеграл від функції sin(*x*) на відрізку [-π , π].

Для цього необхідно відкрити палітру Expression, клікнути або перетягнути в документ шаблон визначеного інтегралу, заповнити шаблон заданими значеннями, використовуючи при переході між елементами шаблону клавішу **Tab** (див. рис. 1.2).

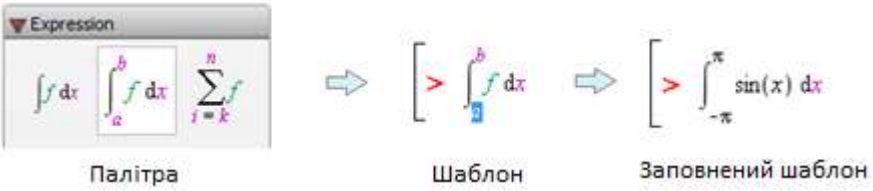



Рисунок 1.2 – Схема використання шаблонів

Еквівалентна команда має такий вигляд `[> int(sin(x), x=0..Pi);`

Всього існує 28 палітр. До них належать палітри для роботи із символами та шрифтами, палітри шаблонів різних виразів і палітри математичних операторів та операцій. Найбільш використовуваними палітрами є такі, як, наприклад:

– Expression (містить шаблони похідних, інтегралів, сум, добутків, різних математичних функцій);

– Greek (містить грецькі символи);

– Common Symbols (містить широковживані математичні символи і знаки);

– Units (SI) (містить шаблони одиниць вимірювання фізичних величин);

– Matrix (створення матриць) та ін.

Корисною є палітра «улюблених» шаблонів Favorites, до якої можна додати будь-який елемент, що користувач найчастіше використовує, правим кліком на цьому елементі.

Відображення набору потрібних палітр на екрані можна настроїти самостійно через меню **View → Palettes →** …

*1.3.1.4. Контекстна панель*

Вигляд контекстної панелі залежить від режимів роботи або від поточного об’єкта. Панель відкриває швидкий доступ до засобів форматування тексту, програми або коментарів, засобів створення креслень або роботи з графічними об’єктами.

### 1.3.2. Класичний інтерфейс (Classic Worksheet)

Класичний інтерфейс повторює інтерфейс старших версій системи і рекомендується до використання на старих персональних комп’ютерах з обмеженою пам’яттю.

На рисунку 1.3 наведений приклад класичного вікна системи з такими основними елементами:

– головним меню 1;

– панеллю інструментів 2;

– контекстним меню 3;

– робочою областю 4;

– рядком стану 5.

Розглянемо різницю між стандартним та класичним інтерфейсом. Вона полягає не лише в зовнішньому вигляді робочих вікон та представленні команд.

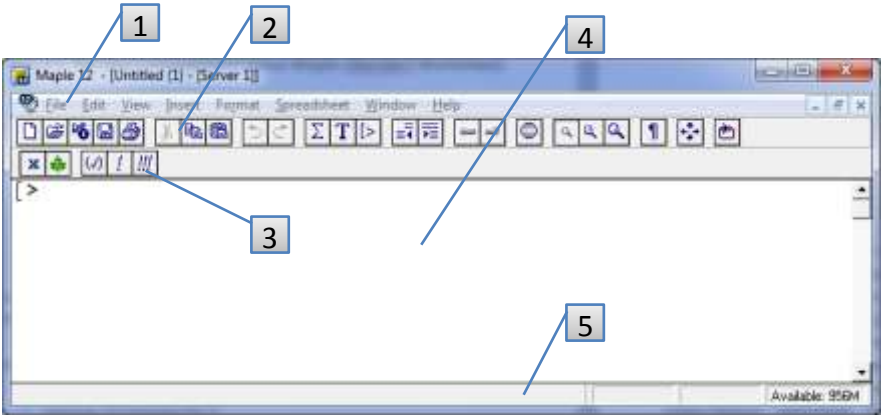



Рисунок 1.3 – Вигляд класичного вікна Maple (Classic Worksheet)

По-перше, розрізняються набори доступних функцій, операторів і процедур. По-друге, по-різному організовані обчислення і доступ до оперативної пам’яті. Так якщо завантажено декілька програм одночасно (тобто відкрито декілька вікон документів), то в стандартному режимі пам'ять розділена між тими програмами, що виконуються на даний момент, та тими, що просто завантажені. Тому ці документи поводять себе незалежно один від одного. Для класичного режиму навпаки: виділена системі пам'ять є спільною для всіх документів. Це дозволяє знизити затрати пам’яті, але всі визначення змінних, функцій користувача, процедур та інших об’єктів одного документа будуть діяти і для інших. Наприклад, якщо в одному документі виконали команду присвоєння значення a:=5, то в іншому документі змінна з таким самим ім’ям також набуде значення 5. Це необхідно знати для розуміння роботи maple-програм та їх налагодження.

### 1.3.3. Режим командного рядка (Command-line version)

Цей режим не має графічного інтерфейсу, зручного для візуального сприйняття користувачем. Використовується для розв’язування складних і великих за обсягом задач.

### 1.3.4. Графічний калькулятор (Maple Calculator)

Це є калькулятор, в якому додані функції графічного представлення результатів обчислень. Графіки можна масштабувати та форматувати.

### 1.3.5. Меплет-програми (Maplets applications)

Використання меплет-програм (або просто меплетів – maplets) передбачає роботу з вікнами, текстовими полями та іншими візуальними елементами за допомогою вказівника-миші. Всі обчислення та графічні побудові не потребують введення maple-команд.

### 1.3.6. Довідкова система Maple

Довідкова система є дуже потужним засобом для отримання інформації про синтаксис maple-команд та вивчення функціональних можливостей системи Maple. Доступ до довідкових ресурсів можливий через декілька інструментів. Головним із них є пункт головного меню **Help**. Позиції цього пункту меню певним чином відрізняються для стандартного та класичного інтерфейсів. Оскільки в даному курсі рекомендується працювати зі стандартним інтерфейсом, то далі розглядатимемо систему допомоги саме для нього.

У меню **Help** можна знайти декілька позицій, зміст яких коротко розкривається на рис. 1.4.

Так, для того щоб зайти в довідкову систему, необхідно:

- вибрати меню **Help → Maple Help** або
- натиснути комбінацію клавіш **Ctrl+F1**, або
- на панелі інструментів натиснути кнопку  .

Довідкова система відкривається в окремому вікні (рис. 1.5), ліва панель якого є навігатором за темами, а на правій панелі розкривається зміст потрібної довідкової сторінки.

Для кожної сторінки допомоги наводяться повна інформація про функції даної команди, її синтаксис, особливості застосування, приклади та гіперпосилання на сторінки споріднених тем. Як правило, наведені приклади ілюструють

багато випадків використання команди, тому, вивчаючи роботу певної команди, рекомендується звертати на них увагу. В Maple є також можливість відкрити сторінку допомоги як робочий документ. Для цього у вікні довідкової системи:

- обрати позицію меню **View → Open Page As Worksheet** або
- натиснути кнопку  на панелі інструментів.

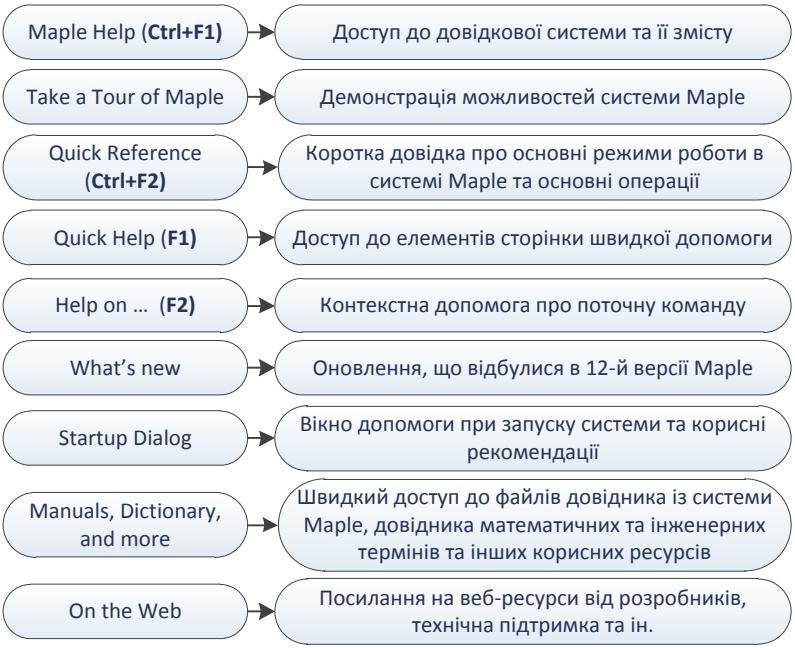



Рисунок 1.4 – Склад меню **Help**

Для *пошуку* необхідної інформації можна застосовувати інструменти пошуку за темами (Topic search) або за ключовими словами (Text Search), які знаходяться на лівій панелі вікна допомоги. Для швидкого пошуку необхідно в робочій області задати команду **?назва_команди**, наприклад **?plot**, та натиснути **Enter**.

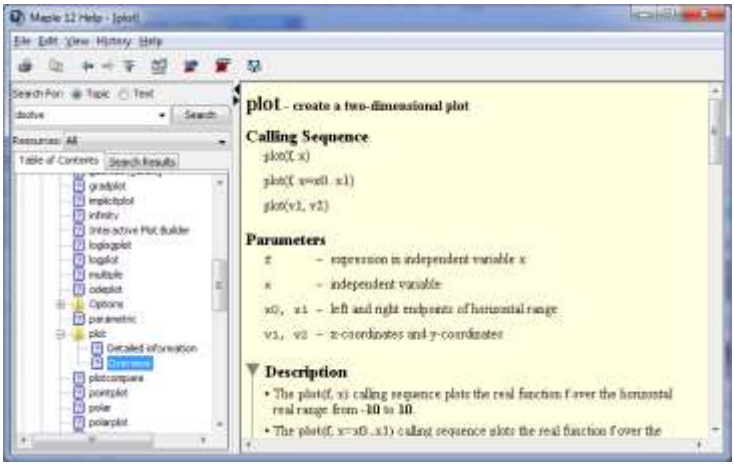



Рисунок 1.5 – Вигляд вікна довідкової системи, відкритої на сторінці функції **plot** для двовимірної графіки

Дуже корисною є функція *контекстного пошуку*. Працюючи в робочому документі або з довідковою системою, необхідно поставити курсор на слово, що цікавить, і натиснути клавішу **F2** або вибрати пункт меню **Help → Help on …**, після чого буде виконаний тематичний пошук по всій довідковій системі.

### 1.3.7. Робота в робочій області. Введення даних

Головним принципом взаємодії користувача та системи Maple є *діалог*. Користувач у просторі робочої області задає (вписує) команди, текст і т. ін. та натискає **Enter**. Натискання цієї клавіші доручає системі опрацювати даний запис. Система реагує і «відповідає» на запит.

Залежно від вигляду робочої області можна вибрати один із двох можливих режимів організації роботи в системі:

а) режим документа (Document Mode);

б) режим робочого листа (Worksheet Mode).

Обрати потрібний режим можна при створенні документа, наприклад, через головне меню: **FILE → NEW → Worksheet / Document**.

*Worksheet* – це режим інтерактивної роботи з maple-командами, який дозволяє більш функціонально їх використовувати, а також зручніше програмувати.

Для введення даних у Worksheet-режимі система видає знак-запрошення `[>`. Після введення команди закінчують рядок знаком `;` або `:` і далі потрібно натиснути **Enter**. Для обох знаків команда буде виконана, але результат буде виведений на екран лише після крапки з комою, наприклад:


Але зауважимо, що двокрапка або крапка з комою є *необхідними* лише в режимах типу 1-D Math input.

Вхідні дані можуть бути подані в математичному (Math) або текстовому (Text) вигляді. Перемикатися між ними можна завдяки кнопкам на 

контекстній панелі. Крім того, на цій панелі можна вибирати формат двовимірного або одновимірного подання вхідних даних. На наступній таблиці 1.1 показана різниця між зазначеними режимами.

Таблиця 1.1 – Порівняння математичного і текстового введень вхідних даних

| Режим \ Формат | 2D Input | Maple Input |
|---|---|---|
| Text Math | $> \sin^2\left(\frac{\pi}{3}\right)$ | `> sin^2(Pi/3)` |
| Text Math | > sin^2(Pi/3) | `> sin^2(Pi/3)` |

Режим *Document* – це режим, що використовується для швидких розрахунків. Він підтримує текстовий та математично орієнтований режими введення, двовимірне та одновимірне подання тексту. Для оформлення розрахунків у текстовому вигляді є зручна можливість отримувати результати не на

окремому рядку, а автоматично після знака «дорівнює». Для цього після введення виразу необхідно натиснути **Ctrl+=**. Порівняємо:

*Використання* **Enter:**

$$\frac{2}{5}+\frac{3}{8}$$

$$\frac{31}{40}$$

*Використання* **Ctrl+=**:

$$\frac{2}{5}+\frac{3}{8}=\frac{31}{40}$$

### 1.3.8. Види команд Maple

Бібліотека команд системи Maple містить великий набір команд, які можна поділити на *дві групи*:

- команди високого рівня;
- пакети команд.

*Команди високого рівня* є найбільш широковживаними, і вони готові до використання відразу після запуску системи. Вони мають вигляд (або «послідовність виклику») `command(arguments)`, де command – це ім'я команди, arguments – список її аргументів. Про синтаксис та використання кожної команди можна знайти вичерпну інформацію в системі допомоги.

*Пакети команд* – це міні-бібліотеки спеціалізованих команд, зібраних разом відповідно до розв'язуваних задач, наприклад, лінійної алгебри, роботи з диференціальними рівняннями та їх розв'язками і т. ін. Існує два способи звернутися до команди з певного пакета. Перший варіант – це послідовність виклику типу `package[command](arguments)`. Тобто зазначити ім'я пакета, далі в квадратних дужках – ім'я команди і далі в круглих дужках – список аргументів команди. У такому разі йде звертання лише до однієї команди з пакета. Якщо необхідно використовувати декілька команд з одного пакета, доцільно спочатку підключити весь пакет командою `with(package)`, а потім звертатися до кожної команди стандартним способом `command(arguments)`.

Повний список команд та пакетів можна знайти в довідковій системі на сторінці **index**. Зайти на цю сторінку можна виконавши команду `?index`.

## 1.4. Поняття функцій та операторів

### 1.4.1. Функції

Важливим поняттям у системі Maple є поняття функції. ***Функція*** повертає результат деякого перетворення вихідних даних, які називаються параметрами функції.

У системі Maple є безліч функцій, вбудованих в ядро, та спеціалізовані пакети. Для того щоб задати функцію у виразі, потрібно ввести її ім'я та список параметрів у круглих дужках. Наприклад, обчислення квадратного кореня або синуса:

```
> sqrt(121);
```

$$11$$

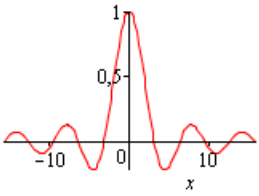
```
> sin(Pi/6);
```

$$\frac{1}{2}$$

В обох випадках маємо повернення значення функції у відповідь на звертання до неї. Ця ознака є *основною ознакою функції* Maple.

Прийнято називати функціями і ті команди, які повертають, наприклад, не числове значення, а результат іншого типу. Наприклад, для побудови графіків математичних функцій використовується maple-функція `plot`:

```
> plot(sin(x)/x, x=-5*Pi..5*Pi);
```



### 1.4.2. Оператори

***Оператори*** як об'єкти Maple самі по собі результат не повертають. Але спільно з функціями і операндами (константами або змінними) допомагають конструювати математичні вирази. Залежно від кількості операндів розрізняють бінарні, унарні, нульарні оператори, і, крім того, існує особливий вид оператора – функціональний. Розглянемо види операторів окремо.

*1.4.2.1. Бінарні оператори*

Бінарні оператори використовуються з двома операндами. Основні з них перелічені в табл. 1.2.

Таблиця 1.2 – Бінарні оператори

| Оператор | Виконувана операція | Оператор | Виконувана операція |
|---|---|---|---|
| + | Додавання | < | Менше |
| - | Віднімання | <= | Менше або дорівнює |
| * | Добуток | > | Більше |
| / | Ділення | >= | Більше або дорівнює |
| ^ | Піднесення до степеня | = | Дорівнює |
| mod | Остача від ділення | <> | Не дорівнює |
| $ | Оператор послідовності | and | Логічне і |
| @ | Оператор композиції функцій | or | Логічне або |
| @@ | Повторювана композиція | xor | Виключне або |
| \|\| | Оператор конкатенації | implies | Імплікація |
| . | Некомутативний добуток | union | Об'єднання множин |
| .. | Діапазон значень | subset | Підмножина |
| , | Розділювач виразів | intersect | Перетинання множин |
| := | Присвоювання | minus | Різниця множин |
| assuming | Обчислення за умови припущень | in | Відношення «бути членом множини» |
| :: | Об'явлення типу | &name | Нейтральний оператор |

**Приклад 1.2.** Деякі операції з використанням бінарних операторів:

```
> 33 mod 15; # остача від ділення
                              3
> x $ 3; # створення послідовності
                           2, 2, 2
> a||0; "calcu"||"lation"; # поєднання
                              a0
                        "calculation"
> sqrt(a^2) assuming a > 0; sqrt(a^2); # обчислення квадратного
                                       кореня за умови a > 0
> (sin@ln)(x); # створення складеної функції
                           sin(ln(x))
```

### *1.4.2.2. Унарні оператори*

Ці оператори відповідно використовуються з одним операндом. Якщо оператор стоїть перед операндом, то це *префіксний* оператор, якщо після операнда – *постфіксний*.

Таблиця 1.3 – Унарні оператори

| Оператор | Виконувана операція |
|---|---|
| + | Унарний плюс (префікс) |
| - | Унарний мінус (префікс) |
| ! | Факторіал (постфікс) |
| not | Логічне ні (префікс) |
| . | Десяткова точка (префікс або постфікс) |
| $ | Оператор послідовності (префікс) |
| &name | Нейтральний оператор (префікс) |

**Приклад 1.3.** Варіанти використання унарних операторів:

```
> + 1; -1; # унарні плюс, мінус
                              1
                             -1
> 5! # факторіал
                            120
> .25 # десяткова точка
                           0.25
> $ 1..5; # створення послідовності
                        1, 2, 3, 4, 5
```

### *1.4.2.3. Нульарні оператори*

Дані оператори мають нульове число аргументів. Таких операторів у Maple всього три (табл. 1.4).

Таблиця 1.4 – Нульарні оператори

| Оператор | Виконувана операція |
|---|---|
| % | Оператор ditto (дітто) – останній вираз |
| %% | Передостанній вираз |
| %%% | Другий передостанній вираз |

Ці оператори застосовуються, якщо необхідно використати останній результат обчислень (або відповідно передостанній та другий передостанній).

**Приклад 1.4.** Додавання трьох останніх результатів обчислень:

```
> x := 2; y := 3; z := x^y;
                        x := 2
                        y := 3
                        z := 8
> % + %% + %%%;
                          13
```

### *1.4.2.4. Функціональний оператор*

Функціональний оператор – це оператор «стрілка», який є засобом створення математичних функцій. Формою запису оператора є схема: *змінні -> вигляд функції*. Цей запис означає, що на змінні (або одну змінну), які є лівим операндом, відбувається певна дія, записана в правому операнді. Ця дія є функцією, якій можна присвоїти ім'я і в подальшому за ним звертатися до неї. Функціональний оператор є своєрідним аналогом процедури.

**Приклад 1.5.** Створити функцію однієї змінної $f_1(x) = \sin(x/3)$. Розрахувати її значення у точці $x = \pi$.

```
> f1 := x -> sin(x/3);
                        f1 := x -> sin(1/3 x)
```

```
> f1(π);
```

$$\frac{1}{2}\sqrt{3}$$

Як бачимо, після створення функції $f_1$ до неї можна звертатися за її ім'ям, зазначаючи в дужках необхідне значення аргументу.

**Приклад 1.6.** Створити функцію двох змінних $f_2(x,y)=\sqrt{x^2+y^2}$. Розрахувати її значення для $x = 3$, $y = 4$.

У разі функції багатьох змінних необхідно лівим операндом задати список змінних у круглих дужках:

```
> f2 := (x, y) → sqrt(x^2 + y^2);
```

$$f2 := (x, y) \to \sqrt{x^2 + y^2}$$

```
> f2(3, 4);
```

$$5$$

У Maple існує певна різниця між поняттями виразу та функції. Деякі команди системи по-різному реагують на дані об'єкти. Розглянемо різницю на такому прикладі.

**Приклад 1.7.** Нехай необхідно розрахувати значення математичного виразу $x^2 + 1$ для $x = 5$.

Варіант 1: а) створюємо вираз з ім'ям *a*:

```
> a := x^2 + 1;
```

$$a := x^2 + 1$$

б) тепер під *a* система буде розуміти $x^2 + 1$:

```
> a;
```

$$x^2 + 1$$

в) якщо підставити значення *x* через запис у круглих дужках, то система підстановку та обчислення проводити не буде:

```
> a(2);
```

$$x(2)^2 + 1$$

г) для обчислень у такому разі можна використати функцію **eval** – "розрахувати значення":

```
> eval(a, x = 5);
```

$$26$$

Варіант 2: а) створюємо функцію *a*(*x*) за допомогою функціонального оператора:

```
> a := x → x^2 + 1;
```

$$a := x \to x^2 + 1$$

б) ім'я функції $a(x)$ дуже зручно використовувати для подальших розрахунків:

```
> a(5);
```

$$26$$

Створити функціональний оператор можна також командою **`unapply(expr, vars)`**, де **`expr`** – це вираз, **`vars`** – змінна (або список змінних). У результаті виконання даної команди відбувається трансформація виразу в функцію. Для того щоб перейти назад від функції до виразу, застосовують команду **`apply(function, vars)`**.

**Приклад 1.8.** Перехід між виразом та функцією:

```
> a := x^2 + 1;
```

$$a := x^2 + 1$$

```
> f := unapply(a, x);
```

$$f := x \to x^2 + 1$$

```
> a := apply(f, x);
```

$$a := x^2 + 1$$

Корисно пам'ятати, що палітра **Expression** має два шаблони для швидкої вставки функції: $f := a \to y$ та $f := (a, b) \to z$.

## 1.5. Типи даних Maple

Система Maple, як і будь-яка інша система комп'ютерної математики, працює з *даними* та обробляє їх. Існує досить багато різних типів даних у Maple та засобів роботи з ними.

### 1.5.1. Прості типи даних

До простих типів даних відносять *числа* та *числові константи*:

- цілі числа (0, 1, 123, –456, …);
- раціональні числа (1/2, –123/4, …);
- дійсні числа з плаваючою точкою (0.12, –0.34, …);
- дійсні числа з мантисою та порядком (1.29E5, –3.5E-10);
- комплексні числа (2+3*I).

Для чисел з плаваючою точкою є важлива особливість. Кількістю цифр, що виводяться після десяткової точки, можна керувати, задаючи її значення системною змінною *Digits*:

```
> Digits := 4;
  ln(2.)
```

$$Digits := 4$$

$$0.6931$$

За замовчуванням *Digits* = 10.

Для задання комплексного числа в алгебраїчній формі використовується константа уявної одиниці – *I*. Для виведення дійсної частини комплексного числа використовується функція **`Re(число)`**, а уявної – функція **`Im(число)`**:

```
> x := 2 + 7 I;
  Re(x);
  Im(x);
```

$$x := 2 + 7\,I$$

$$2$$

$$7$$

### 1.5.2. Дані множинного типу

*1.5.2.1. Послідовності*

***Послідовність*** є фундаментальною структурою даних у Maple. Це ряд виразів, які записані через кому. Послідовностям можна присвоювати імена. Елементами послідовності можуть бути будь-які вирази – константи, змінні, функції тощо:

```
> P := x, y, 35, exp(x), 0;
```

$$P := x, y, 35, e^{x}, 0$$

Для того щоб звернутися до певного елемента послідовності, зазначають ім'я послідовності та індекс (номер) позиції елемента в квадратних дужках. Причому якщо рахувати позиції з початку послідовності, то індекс позитивний, якщо з кінця – негативний:

```
> P[2];
```

$$y$$

```
> P[-2];
```

$$e^{x}$$

Для того щоб вибрати декілька елементів підряд, зазначають діапазон індексів через дві крапки, наприклад:

```
> P[3..-2];
                              35, e^x
```

Цей принцип посилання на елементи поширюється на більшість структур даних.

*1.5.2.2. Набори*

***Набори*** *(або множини)* – це ряд будь-яких виразів, які створюються за допомогою фігурних дужок **{}**. Відмінною властивістю наборів є те, що Maple автоматично усуває елементи з однаковим значенням, а також упорядковує елементи за збільшенням значення (якщо числа) та за алфавітом (якщо символи та рядки):

```
> {a, b, d, 1, a, c, b, d, 3, b, 2};
                          {1, 2, 3, a, b, c, d}
```

*1.5.2.3. Списки*

***Списки*** – це упорядковані набори. Вони створюються в квадратних дужках **[]**. Особливістю списків є те, що їх елементи перетворюються та виводяться лише у тому порядку, в якому були задані:

```
> [2, 1, 3 + 4, 5];
                          [2, 1, 7, 5]
```

Елементами списку можуть бути також списки. Тому вони часто використовуються для створення векторів і матриць.

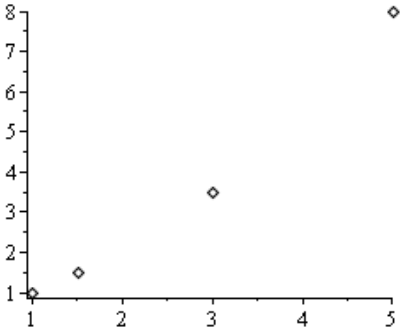
**Приклад 1.9.** Побудова точкового графіка

Нехай задані координати точок графіка у вигляді окремих списків абсцис і ординат: *X* = [1, 1.5, 3, 5], *Y* = [1, 1.5, 3.5, 8]. Для того щоб побудувати графік, необхідно створити для кожної точки список із пари координат, а потім об'єднати всі пари в один список. Це можна зробити таким чином:

```
> X := [1, 1.5, 3, 5]:
  Y := [1, 1.5, 3.5, 8]:
  p := (X, Y) -> [X, Y];
                        p := (X, Y) -> [X, Y]
> coordinaty := zip(p, X, Y);
          coordinaty := [[1, 1], [1.5, 1.5], [3, 3.5], [5, 8]]
```

Далі графік будується функцією **pointplot** з пакета **plots**:

```
> with(plots):
  pointplot(coordinaty);
```




Тут, як уже зазначалося в розділі 1.1, команда **`with`** підключає спеціалізований пакет функцій, до якого належить і необхідна нам функція побудови точкового графіка.

*1.5.2.4. Масиви*

***Масивами*** називаються узагальнені списки. Вони мають дві відмінні властивості, а саме:

а) розмірність масивів може бути більшою за одиницю;

б) індекси елементів – натуральні числа.

Масиви створюються командою **`Array`**. Синтаксис команди такий: **`Array(a..b, c..d, S)`**. Ця команда означає, що створюється масив з номерами рядків від **`a`** до **`b`**, номерами стовпців від **`c`** до **`d`** та значеннями в двовимірному списку **`S`**.

```
> a := Array(1..3, 1..3, [[c, d, e], ["String", infinity, π], [7, 8, 9]]);
```

$$a := \begin{bmatrix} c & d & e \\ \text{"String"} & \infty & \pi \\ 7 & 8 & 9 \end{bmatrix}$$

```
> a[1, 2];
```

$$d$$

У принципі, розмірність масивів не обмежена, і вони можуть бути багатовимірними. Повністю на екрані будуть відображатися ті, що мають розмірність 1 або 2 та мають не більше 10 індексів для кожної розмірності. Інакше масиви відображатимуться спеціальним замінником-покажчиком. Щоб розкрити такі масиви, необхідно двічі клікнути на замінникy.

*1.5.2.5. Таблиці*

***Таблиці*** – це розширення масивів із довільною індексацією. Тобто це масиви, для яких користувач може задавати будь-які індекси, не лише натуральні числа. Створюються таблиці командою **table**:

```
> T := table([a = α, b = β, c = [γ, δ]]);
```

$$T := table([c = [\gamma, \delta], b = \beta, a = \alpha])$$

```
> T[b];
```

$$\beta$$

Як бачимо, в наведеному випадку за індекси використані букви латинського алфавіту *a*, *b*, *c*.

*1.5.2.6. Вектори та матриці*

***Вектори та матриці*** – це специфічні об'єкти Maple, що використовуються для задач лінійної алгебри та векторного числення. Вектори є одновимірними структурами, а матриці – двовимірними. Для роботи з ними існує спеціальний пакет **LinearAlgebra**.

Щоб створити *вектор*, необхідно дані взяти в кутові дужки **<>**. Якщо потрібний вертикальний вектор, то дані розділяються комами, якщо горизонтальний – вертикальними рисками (|).

```
> V_vert := ⟨1, 2, 3⟩;
```

$$V_{vert} := \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

```
> V_horiz := ⟨1|2|3⟩;
```

$$V_{horiz} := \begin{bmatrix} 1 & 2 & 3 \end{bmatrix}$$

Для створення *матриць*, а також для виконання нескладних операцій із ними, використовується команда **Matrix**, синтаксис якої відрізняється залежно від задач. Для створення матриці з однаковими елементами як аргументи функції **Matrix** зазначаються лише індекси рядків і стовпців, а також значення елементів (якщо ненульові):

```
> M1 := Matrix(1..3, 1..2, 1);
```

$$M1 := \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 1 & 1 \end{bmatrix}$$

Для довільних значень елементів доцільно їх зазначати вкладеними списками:

```
> M2 := Matrix([[1, 2], [3, 4], [5, 6]]);
```

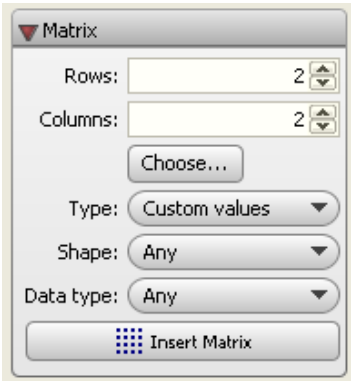
$$M2 := \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$$

За допомогою **Matrix** можна також будувати матриці, які є, наприклад, сумою або різницею двох інших:

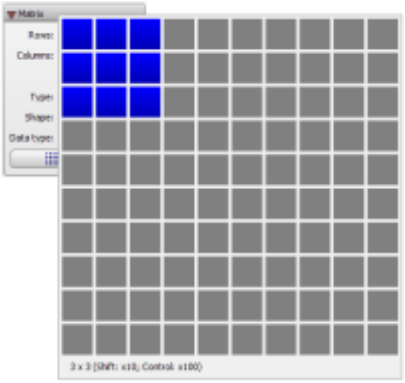
```
> M3 := Matrix(M1 + M2);
```

$$M3 := \begin{bmatrix} 2 & 3 \\ 4 & 5 \\ 6 & 7 \end{bmatrix}$$

Корисною можливістю для створення векторів і матриць Maple є палітра **Matrix**, вигляд якої показаний на рис. 1.6 а.




а



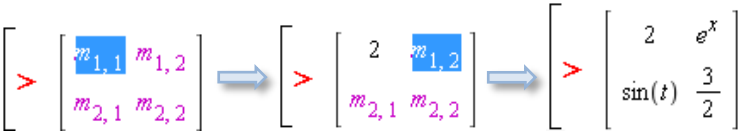
б

Рисунок 1.6 – Палітра Matrix для створення векторів та матриць (а) та її розширення для візуального вибору розміру матриці (б)

На рис. 1.6 а Rows – кількість рядків; Columns – кількість стовпців; Choose – клавіша візуального вибору розміру матриці (рис. 1.6 б), Type та Shape – вибір типу та форми матриці; Data type – вибір типу даних. Insert Matrix – кнопка вставки матриці в

робочий документ. Тип матриці та даних рекомендується зазначати для великих обсягів даних, щоб підвищити ефективність обчислень.

При вставці матриці в документ маємо невизначені елементи, в які далі вводяться дані. Зручно переходити між елементами клавішею **Tab**:



### 1.5.3. Рядки

***Рядки*** як тип даних являють собою ланцюжки символів, взяті в подвійні лапки (“ ”). Доступ до символів можна отримати посилаючись на ім’я рядка та номера символів у квадратних дужках:

```
> S := "This is a String";
                    S := "This is a String"
> S[11..16];
                          "String"
```

Для роботи з рядками призначений спеціальний пакет функцій **StringTools**. Він підключається стандартно командою `with(StringTools)`. Детальну інформацію про пакет можна отримати в системі допомоги, виконавши команду `?StringTools`.

**Приклад 1.9.** Створення списку слів із рядка:

```
> with(StringTools):
  Split("This is a String");
                ["This", "is", "a", "String"]
```

### 1.5.4. Константи

***Константи*** – це найпростіші іменовані об’єкти з наперед визначеним значенням. Константи можна поділити на такі групи:

а) числові константи;

б) рядкові константи;

в) константи, вбудовані в ядро Maple.

*Числові константи* не мають окремого імені, тобто їх ім'я і є значенням. Наприклад, числа 2 та 5 у записі `> 2*5`.

*Рядкові константи* по суті є довільними ланцюжками символів, наприклад `"Maple"`, і служать значеннями рядкових змінних. Якщо в лапки взяті числа, то вони також є лише рядками і їх не можна використати в обчисленнях.

Константи, *вбудовані в ядро* Maple, мають заздалегідь визначені імена та значення. Їх імена не можна присвоїти будь-яким іншим об'єктам, а також не можна змінити їх значення. Тобто їх можна сприймати як визначені наперед глобальні змінні. Список констант наведений у таблиці 1.5.

Таблиця 1.5 – Вбудовані в ядро константи

| Ім'я | Значення | Опис |
|---|---|---|
| Pi (або π) | 3,1415… | Число π |
| gamma (або γ) | 0,57716… | Константа Ейлера |
| I | $\sqrt{-1}$ | Уявна одиниця |
| infinity | $\infty$ | Нескінченність |
| -infinity | $-\infty$ | Мінус нескінченність |
| Catalan | 0,91596… | Константа Каталана |
| true | «істина» | Означає, що логічне значення правильне |
| false | «хибність» | Означає, що логічне значення неправильне |
| FAIL | «помилка» | Невідомо, правильний чи неправильний логічний результат |

Серед зазначених констант немає виділеної для числа $e$ = 2,71828… . Щоб його задати в системі Maple, використовується експоненціальна функція `exp(1.0)`.

Крім наведених констант, в Maple 12 передбачене використання в обчисленнях і *наукових констант*, до яких відносять фізичні константи і властивості хімічних елементів та ізотопів. Для цього створений спеціалізований пакет

**ScientificConstants**. Детально робота з науковими константами описана в п. 2.3.3 і може бути знайдена самостійно в системі допомоги за командою ?**ScientificConstants**.

### 1.5.5. Змінні

***Змінними*** називаються ті об'єкти, значення яких змінюється по ходу виконання документа. Ім'я змінної повинне бути унікальним, складатися з букв та/або цифр, обов'язково починатися з букви. Не можна давати змінним імена, що повторюють ключові слова системи Maple, та включати до імені знаки операторів. Символ підкреслення (_) допускається. Для того щоб у режимі 2*D-Math* ввести цей символ, потрібно спочатку ввести обернений слеш, а потім підкреслення (_). Альтернативно, які ім'я змінної можна ввести будь-які символи в лівих одиничних лапках ``.

Використання змінних у математичних виразах та символьних розрахунках не потребує їх попереднього оголошення.

Присвоєння значення змінній відбувається за допомогою оператора `:=` . Змінні можуть бути різних типів, і тип визначається присвоєним значенням. Таким чином, є такі типи змінних:

- integer (цілочислові);
- rational (раціональні);
- float (дійсні з плаваючою точкою);
- complex (комплексні);
- string (рядкові);
- symbol (символьні);
- list (список).

Тип змінної можна перевірити за допомогою команди `type(var,type)`, де `var` – ім'я змінної, `type` – тип, на який проводиться перевірка. Команда вертає значення *true* або *false* («істина» або «хибність»).

**Приклад 1.10.** Перевірити, чи є число $a = 1{,}2$ дійсним і чи є число $\pi$ константою:

```
> a := 1.2 :
> type(a, float);
                                true
> type(Pi, constant);
                                true
```

Змінна, що не має присвоєного значення, займає менше пам’яті, ніж та, що має. Тому іноді корисно ліквідувати присвоєння. Як це зробити, ілюструє приклад 1.11.

**Приклад 1.11.** Ліквідація присвоєння значення змінної:

```
> x := 10;  # нехай змінній x присвоюється значення 10
                                x := 10
> x; # перевірка значення x
                                  10
> x := 'x'; # команда ліквідації значення
                                x := x
> x; # перевірка значення x
                                   x
```

Для того щоб ліквідувати *всі* попередні присвоєння, корисно використовувати команду **`restart`**. Ця команда очищує всю внутрішню пам'ять Maple, тобто ліквідує всі визначення та вивантажує всі пакети. Якщо під час роботи з документом є необхідність багатократно запускати виконання програми спочатку, рекомендується командою **`restart`** починати текст програми.

# 2. Математичні обчислення

## 2.1. Символьні та числові обчислення

### 2.1.1. Визначення

Система Maple має широкі можливості для символьних, тобто аналітичних, та числових обчислень.

***Символьні обчислення*** – це математичні перетворення виразів, що містять: а) символьні або абстрактні величини (змінні, функції, оператори); б) точні числа, наприклад, цілі числа, раціональні, $\pi$, $e$, і т. ін. Метою таких перетворень може бути отримання виразів у більш простій формі або встановлення співвідношень з іншими, більш зрозумілими, формулами.

**Приклад 1.12.** Спростити вираз *eq*:

```
> eq := cos(x)^5 + sin(x)^4 + 2·cos(x)^2 - 2·sin(x)^2 - cos(2·x);
```

$$eq := \cos(x)^5 + \sin(x)^4 + 2\cos(x)^2 - 2\sin(x)^2 - \cos(2x)$$

```
> simplify(eq);
```

$$\cos(x)^4 (\cos(x) + 1)$$

**Приклад 1.13**. Знайти похідну функції $\sin^2 x$:

```
> diff(sin^2(x), x)
```

$$2\sin(x)\cos(x)$$

**Приклад 1.14**. Розв'язати квадратне рівняння:

```
> solve(a·x^2 + b·x + c, x)
```

$$\frac{1}{2}\,\frac{-b+\sqrt{b^2-4ac}}{a},\ -\frac{1}{2}\,\frac{b+\sqrt{b^2-4ac}}{a}$$

***Числові обчислення*** – це маніпуляції з виразами в контексті лічильних операцій (тобто арифметичних) зі скінченною точністю. Вирази в такому разі містять конкретні числа. Причому точні числа замінюються на наближені числа з плаваючою точкою із заданою точністю. Наприклад, запис $\pi$ для Maple є точним, оскільки під іменем $\pi$ приховується майже нескінченний ланцюжок цифр (технічно обмежений лише кількістю порядку декількох сотень тисяч знаків). А в числових

обчисленнях точне число замінюється на число з плаваючою точкою, яке є наближеним, наприклад $\pi \approx 3{,}1415927$. Тому може накопичуватися похибка, значення якої може бути не менш важливим, ніж сам обчислений результат. Як правило, числові операції задіяні, якщо використовуються числа з десятковою точкою або команда **evalf**.

### 2.1.2. Точні та наближені обчислення

Система Maple проводить *точні* обчислення з такими величинами, як цілі числа, раціональні, математичні константи, матриці. Їм можна присвоїти певні імена, наприклад *x* або *y*, які поряд із математичними функціями типу cos(*x*) розглядатимуться системою як *символьні* об'єкти. Результатом символьних обчислень також буде точний результат, який за необхідності можна потім окремо перетворити на наближений.

Наприклад, результатом додаванням простих дробів є також простий дріб:

$$> \frac{11}{6}+\frac{2}{5}; \qquad \frac{67}{30}$$

$$> 1+\frac{\pi}{2}; \qquad 1+\frac{1}{2}\pi$$

Або при обчисленні значень, наприклад тригонометричних функцій, sin(1) є точним числом, а 0,8414709848 його округлене значення. Таким чином, Maple надає перевагу точному результату:

```
> sin(1)
                          sin(1)
> evalf(%);
                       0.8414709848
```

$$> \cos\left(\frac{\pi}{4}\right); \qquad \frac{1}{2}\sqrt{2}$$

Обчислення, що проводяться за участі чисел із плаваючою точкою, є *наближеними*. При цьому наявність саме десяткової точки сповіщає систему про використання в обчисленнях числових наближень. Відповідно результат також буде у вигляді числа з десятковою точкою. Наприклад, порівняйте два дроби і як їх сприймає система:

```
> 7/10;
```

$$\frac{7}{10}$$

```
> 7./10;
```

0.7000000000

Деякі команди Maple потребують роботи саме з наближеними значеннями. Наприклад, команда побудови графіка **plot** не зможе відобразити значення $\pi$, а 3,14 зможе. Для перетворення точних величин на наближені використовується команда **evalf**. За кількість знаків, що виводяться після десяткової коми, відповідає системна змінна **Digits** (див. п. 1.5.1).

### 2.1.3. Джерела похибок

Оскільки числові обчислення, тобто з числами з плаваючою точкою, є наближеними, то природно, що вони містять похибку. Основними джерелами похибки є такі:

а) округлення. Деякі величини не можуть бути точно подані у формі числа з десятковою точкою, тому округлення призводить до похибки: 2/3 ≈ 0,66666667;

б) накопичення малих похибок. Якщо проводиться велика кількість арифметичних операцій, то їх сумарна похибка може істотно перевищувати похибку одиничної операції;

в) віднімання дуже близьких величин. Розглянемо простий приклад розрахунку виразу ($x$ - tg$x$) при $x \approx 0$. Як відомо, тангенс наближається до нуля при наближенні до нуля його аргументу. Таким чином, будемо мати такий результат:

```
> x - tan(x) | x = 0.00001
```

0.

Так, отримуємо наближений результат. Відмітимо, що підставлення в даному вигляді виконане за допомогою відповідного шаблона в палітрі *Expression*. Інший швидкий засіб підставлення – команда контекстного меню *Evaluate at a point*.

Щоб знайти вагому різницю між $x$ та tg$x$, можна використати, наприклад, розвинення в ряд Тейлора (див. п. 2.4.6):

```
> res := taylor(x - tan(x), x);
```

$$res := -\frac{1}{3}x^3 - \frac{2}{15}x^5 + O(x^6)$$

```
> res | x = 0.00001
```

$$-3.333333333\ 10^{-16}$$

Отримуємо дуже мале число, але зі збільшеною точністю.

### 2.1.4. Перетворення виразів

Для виконання найбільш поширених дій над виразами в загальному (символьному) вигляді в системі Maple передбачені такі функції:

– **`simplify()`** – спростити вираз, а точніше – застосувати правила спрощення:

```
> expr := sin(x)^2 + ln(2*y) + cos(x)^2:
> simplify(expr);
```

$$1 + \ln(2) + \ln(y)$$

Якщо перетворювати лише частину доданків, то застосовуються опції:

```
> simplify(expr,'trig');
```

$$1 + \ln(2y)$$

```
> simplify(expr,'ln');
```

$$\sin(x)^2 + \ln(2) + \ln(y) + \cos(x)^2$$

– **`factor()`** – розкласти на множники:

```
> factor(x^6 - x^5 - 9*x^4 + x^3 + 20*x^2 + 12*x);
```

$$x\,(x-2)\,(x-3)\,(x+2)\,(x+1)^2$$

– **`expand()`** – розкрити дужки:

```
> expand((y - 3)(x + 1)^2(x + y));
```

$$y(x+1)(x+y)^2 - 6y(x+1)(x+y) + 9$$

```
> expand(sin(x + y));
```

$$\sin(x)\cos(y) + \cos(x)\sin(y)$$

– **`combine()`** – зібрати підвирази у вирази:

```
> combine(4*ln(x) - ln(y)) assuming x > 0, y > 0;
```

$$\ln\left(\frac{x^4}{y}\right)$$

– **convert()** – перетворити в іншу форму:

```
> convert(Pi,'degrees');
                                180 degrees
> convert(cos(x), exp);
                                1/2 e^(I x) + 1/2 e^(-I x)
> convert([a, b, c, d],'set');
                                {a, b, c, d}
```

– **normal()** – звести до нормального вигляду з розкладанням на множники:

```
> normal((x^2 - y^2)/(x - y)^3)
                                (x + y)/(x - y)^2
> normal((x^2 - y^2)/(x - y)^3,'expanded');
                                (x + y)/(x^2 - 2 x y + y^2)
```

– **sort()** – сортування елементів (упорядкування):

```
> sort([4, 3, 2.1, -4. - 43, 0]);
                                [-47., 0, 2.1, 3, 4]
> sort(-6 x y^2 - 3 x^2 y + 5 x + 1);
                                -3 x^2 y - 6 x y^2 + 5 x + 1
```

Більшість цих команд також доступна з контекстного меню.

### 2.1.5. Робота з частинами виразу

За необхідності «витягнути» частину компонентів будь-якого виразу для подальшої роботи з ними використовують такі команди Maple:

– **lhs()** – для добування лівої частини виразу;

– **rhs()** – для добування правої частини виразу:

```
> a := y = x + 1;
                                a := y = x + 1
> lhs(a);                                  > rhs(a);
                  y                                        x + 1
```

– **numer()** – добути числовик дробу;
– **denom()** – добути знаменник дробу:

```
> d := sin(x)/x
```

$$d := \frac{\sin(x)}{x}$$

```
> numer(d);
```

$$\sin(x)$$

```
> denom(d);
```

$$x$$

– **nops()** – визначити кількість операндів. Наприклад, визначити кількість доданків:

```
> a := 6*x^3 - x^2 + 7:
> nops(a);
```

$$3$$

або визначити кількість елементів списку:

```
> sols := [solve(a, x)];
```

$$sols := \left[-1, \frac{7}{12} + \frac{1}{12} I \sqrt{119}, \frac{7}{12} - \frac{1}{12} I \sqrt{119}\right]$$

```
> nops(sols);
```

$$3$$

– **op()** – добути операнд. Для цього як перший аргумент даної функції зазначається порядковий номер потрібного елемента:

```
> op(1, sols)
```

$$-1$$

– **indets()** – визначити невизначені елементи виразу, тобто ті, які не мають присвоєного значення:

```
> expr := x*sin(y) + y^2;
```

$$expr := x \sin(y) + y^2$$

```
> indets(expr);
```

$$\{x, y, \sin(y)\}$$

Якщо певній змінній присвоїти певне значення, то результат зміниться таким чином:

```
> x := 2:
  indets(expr);
```

$$\{y, \sin(y)\}$$

### 2.1.6. Обчислення виразів

*2.1.6.1. Підстановка значень*

Для того щоб знайти значення виразу при певних значеннях його підвиразів (або певних змінних), необхідно виконати підстановку. Це можна зробити декількома шляхами:

а) використати команду-шаблон на палітрі команд *Expression*:

$$f(x)\Big|_{x=a}$$

б) правою кнопкою миші або тачпаду клікнути на виразі і обрати пункт контекстного меню *Evaluate at a Point*. У такому разі Maple виконає підстановку за допомогою команди **`eval`**. Ця команда може виконувати підстановки значень як змінних, так і цілих операндів:

```
> eval( x - tan(x), [x = π/4])
```

$$\frac{1}{4}\pi - 1$$

```
> eval( x*y - tan(x*y), x·y = Pi/6 )
```

$$\frac{1}{6}\pi - \frac{1}{3}\sqrt{3}$$

в) за допомогою функції **`subs(s1, s2, …, sn, expr)`**, де **`si`** – вирази типу *x* = *y*, які підставляються у вираз **`expr`** :

```
> subs(c = a + b, [a, b, c]);
```

$$[a, b, a + b]$$

Цю функцію можна використовувати в комбінації з функціями **`eval`** або **`evalf`**:

```
> evalf(subs(x = 3, y = 4, a = sqrt(x^2 + y^2)));
```

$$a = 5.000000000$$

*2.1.6.2. Функції розрахунку значення виразу*

– **`eval(expr), eval(expr,[x=a,y=b])`** – обчислити точне значення виразу, в тому числі при заданих значеннях змінних *x* та *y* (див. також п. 2.1.6.1);

– **`evalf(expr)`** – обчислити значення виразу **`expr`** у вигляді числа з плаваючою точкою (тобто наближене числове значення) (див. також п. 2.1.6.1);

– **`evalc(expr)`** – обчислити вираз **`expr`** у вигляді комплексного числа;

– **`evalb(expr)`** – обчислити значення логічного виразу **`expr`** (із можливим результатом true, false, FAIL – див. п. 1.5.4).

*2.1.6.3. Задання математичних функцій у виразах*

Математичні вирази складаються з операторів, операндів та функцій. Система Maple має повний набір елементарних функцій, які можна включати в математичні вирази. Їх синтаксис та зміст наведені в таблиці 2.1.

Таблиця 2.1 – Математичні функції

| **Степеневі та логарифмічні функції** | |
|---|---|
| sqrt(x) – квадратний корінь | log10(x) – десятковий логарифм |
| exp(x) – експонента | log[a](x) – логарифм за заданою основою |
| ln(x) – натуральний логарифм | log(x) – загальний логарифм |
| **Тригонометричні функції** | |
| sin(*x*) – синус аргументу *x* | cot(*x*) – котангенс аргументу *x* |
| cos(*x*) – косинус аргументу *x* | arcsin(*x*) – арксинус аргументу *x* |
| tan(*x*) – тангенс аргументу *x* | arcos(*x*) – арккосинус аргументу *x* |
| **Гіперболічні функції** | |
| sinh(*x*) – синус гіперболічний | arcsinh(*x*) – арксинус гіперболічний |
| cosh(*x*) – косинус гіперболічний | arccosh(*x*) – арккосинус гіперболічний |
| **Функції з елементами порівняння** | |
| ceil(*x*) – знаходить найменше ціле, яке більше або дорівнює *x* | trunk(*x*) – ціле, округлене в напрямку *x* = 0 |
| floor(*x*) – знаходить найбільше ціле, яке менше або дорівнює *x* | round(*x*) – округлене значення числа (до цілого) |
| frac(*x*) – дробова частина числа | signum(*x*) $= \begin{cases} -1, \text{якщо } x < 0 \\ 1, \text{якщо } x > 0 \\ 0, \text{якщо } x = 0 \end{cases}$ – функція знака числа |
| **Функції роботи з комплексними числами** | |
| abs(*x*) – обчислює модуль дійсного або комплексного числа | polar(*x*) – задає комплексне число у полярному вигляді |
| argument(*x*) – знаходить аргумент комплексного числа | Re(*x*) – знаходить дійсну частину комплексного числа |
| conjugate(*x*) – знаходить комплексно спряжене число | Im(*x*) – знаходить уявну частину комплексного числа |
| **Спеціальні функції** | |
| Si – інтегральний синус | Ei – експоненціальний інтеграл |
| Ci – інтегральний косинус | Heavyside – функція Хевісайда |

Продовження таблиці 2.1

| | |
|---|---|
| Shi – інтегральний гіперболічний синус | euler – числа та поліноми Ейлера |
| Chi – інтегральний гіперболічний косинус | Gamma – *γ*-функція |
| Dirac – *δ*-функція Дірака | lnGAMMA – логарифмічна *γ*-функція |
| BesselI(J,K,Y) – функції Бесселя | Li – логарифмічний інтеграл |
| **Операції з цілими числами** | |
| ifactor(*a*) – розкладання на множники | igcd(*a,b*) – найбільший спільний дільник |
| iquo(*a,b*) – ціле від ділення *a* на *b* | icm(*a,b*) – найменше спільне кратне |
| irem(*a,b*) – залишок від ділення | isqrt(*a*) – ціле від кореня квадратного |

## 2.2. Розв'язування рівнянь, нерівностей та їх систем

### 2.2.1. Символьне розв'язування

Будь-яке рівняння в системі Maple можна задати у вигляді

$$f_1(x) = f_2(x) \text{ або } F(x) = 0$$

Аналогічним чином задаються нерівності за допомогою знаків відношення: >, <, >=, <=.

Рівнянню або нерівності можна присвоїти ім'я, в такому разі необхідно використовувати саме знак присвоєння, наприклад `eq:=x^2+x+2=0`.

Для точного розв'язання рівнянь, нерівностей та їх систем у символьному вигляді використовується функція `solve(eqns, vars)`, де `eqns` – рівняння, нерівність або їх система; `vars` – невідомі змінні, що підлягають визначенню. Якщо змінна лише одна, то Maple може автоматично розпізнати змінну і параметр `vars` можна опустити.

**Приклад 2.1.** Розв'язати рівняння $(x - a)^2 + bx = c$:

```
> eq := (x-a)^2+bx=c;
```

$$eq := (x-a)^2 + bx = c$$

```
> solve(eq, x);
```

$$a + \sqrt{c - bx},\ a - \sqrt{c - bx}$$

**Приклад 2.3.** Розв'язати системи рівнянь та нерівностей:

```
> solve({2*x - y = 1, x*y = 10}, [x, y])
```

$$\left[[x=-2, y=-5], \left[x=\frac{5}{2}, y=4\right]\right]$$

```
> solve({x + y < 4, y^2 = 9}, [x, y]);
```

$$[[y=3, x<1], [y=-3, x<7]]$$

Як видно із цих прикладів, при записі команди **solve** систему рівнянь або нерівностей необхідно задавати списком або набором, тобто через кому і брати в дужки {} або []. Перелік невідомих змінних, відносно яких розв'язується система, подається списком у квадратних дужках – [].

**Приклад 2.4.** Розв'язати тригонометричне рівняння sin(2*x*) + 1 = cos(2*x*):

```
> solve( sin(2*x) + 1 = cos(2*x) )
```

$$x=-\frac{1}{4}\pi, \quad x=0$$

Особливістю тригонометричних рівнянь є те, що їх розв'язок найчастіше повинен бути періодичним числом відповідно до періодичності тригонометричних функцій. Із прикладу 2.4 бачимо, що виводяться розв'язки рівняння лише на одному періоді (оскільки період функцій sin та cos дорівнює 2π, і в рівнянні аргументами sin та cos є 2*x*, то період відповідно 2π/2 = π). Для того щоб змусити функцію **solve** навести всі розв'язки, змінній оточення **_EnvAllSolutions** необхідно присвоїти значення **true**:

```
> _EnvAllSolutions := true;
```

$$\_EnvAllSolutions := true$$

```
> solve( sin(2*x) + 1 = cos(2*x) )
```

$$x=-\frac{1}{4}\pi+\pi\_Z1\sim, \quad x=\pi\_Z2\sim$$

В отриманих розв'язках бачимо, що період дійсно є π, але записаний за допомогою згенерованої системою Maple змінної *_Z1~*. Maple використовує змінні вигляду *_ZN~*, де *N* – порядковий номер, як цілі числа-посередники. Окрім цілих чисел, можуть вводитись автоматично числа типу *_B*, що свідчить про бінарну форму, тобто 0 або 1, та *_NN*, що означає

невід'ємне число. Позначка ~ свідчить про те, що ця величина є величиною з припущенням.

Часто трапляються математичні вирази, коли на їх частину накладаються певні умови (як, наприклад, підлогарифмовий вираз повинен бути додатним, ненульовий знаменник дробу і т. ін.). Перед розв'язуванням рівнянь, що містять такі вирази, рекомендовано це зазначати системі Maple. Це виконується командою **assume()**, що означає «припустимо, що…».

```
> # Припустимо, що x — додатна величина, і знайдемо корінь квадратний з x^2 :
> assume(x > 0) : sqrt(x^2)
                                    x~
> # Можна спитати Maple про прийняте припущення щодо x:
> about(x);
Originally x, renamed x~:
  is assumed to be: RealRange(Open(0),infinity)

> # Для того що зняти припущення, виконуємо таку команду:
> unassign('x');
> # І перевіряємо, що відомо про x:
> about(x);
x:
  nothing known about this object
```

### 2.2.2. Розв'язування в числовому вигляді

Для отримання числового розв'язку рівняння, нерівності або їх системи використовують функцію **fsolve**. Вона виводить розв'язок відразу у вигляді дійсних або комплексних чисел. Синтаксис даної функції такий: **fsolve(eqns, vars, options)**, де **eqns** – рівняння, нерівність або їх система, що підлягає розв'язанню; **vars** – список невідомих змінних, які необхідно знайти; **options** – додаткові необов'язкові опції.

У найпростішому випадку в функції **fsolve** достатньо задати лише рівняння, і система сама з'ясує невідому змінну та знайде її значення:

```
> fsolve(5*x^4 - 2*x^3 - 3*x^2 + 4*x - 5);
                    -1.209126017, 1.073321849
```

При такому записі будуть знайдені лише ті розв'язки, які є дійсними числами. Для того щоб змусити знайти комплексні розв'язки, необхідно задати опцію **`complex`**:

```
> fsolve( exp(x) = 2*x, x, complex)
                  0.7940236323 - 0.7701117505 I
```

Можна змусити Maple шукати розв'язки не на всій числовій шкалі, а лише на певному інтервалі. Тоді зазначається опція **`{var=a..b}`**:

```
> fsolve(exp(x) - sin(2*x), x, {x = -Pi..0})
                                -1.665880472
```

Локалізація пошуку коренів може бути корисною, якщо функції **`solve`** та **`fsolve`** у своєму звичайному вигляді коренів не знаходять.

### 2.2.3. Функція RootOf

Структури з функцією **`RootOf`** з'являються в розв'язках після виконання функції **`solve`**. Вони замінюють самі корені і означають, що система не може знайти точного символьного виразу для коренів і що розв'язок даний у неявному вигляді.

```
> a := solve( 5*x^4 - 2*x^3 - 3*x^2 + 4*x - 5 = 0);
a := RootOf(5*_Z^4 - 2*_Z^3 - 3*_Z^2 + 4*_Z - 5, index = 1),
  RootOf(5*_Z^4 - 2*_Z^3 - 3*_Z^2 + 4*_Z - 5, index = 2),
  RootOf(5*_Z^4 - 2*_Z^3 - 3*_Z^2 + 4*_Z - 5, index = 3),
  RootOf(5*_Z^4 - 2*_Z^3 - 3*_Z^2 + 4*_Z - 5, index = 4)
```

Параметр *index* упорядковує та пронумеровує розв'язки.

У даному випадку можна отримати числові значення коренів рівняння за допомогою функції **`evalf`**:

```
> evalf(a);
1.073321849, 0.2679020837 + 0.8359272005 I, -1.209126017,
  0.2679020837 - 0.8359272005 I
```

Якщо **`solve`** не знаходить корені, необхідно використовувати числовий спосіб розв'язування – **`fsolve`**.

### 2.2.4. Робота з отриманими розв'язками

Після отримання розв'язку за допомогою функцій solve та fsolve рекомендується *перевірити його правильність*. Для цього необхідно виконати підстановку кореня у вихідне рівняння (див. п. 2.1.6.1) та розрахувати значення лівої та правої частин рівняння за допомогою функції eval:

```
> eq := sin(x) = -cos(x);
                    eq := sin(x) = -cos(x)                    (1)
> solve(eq, x);
                              -1/4 π                          (2)
> eval(eq, x = (2));
                    -1/2 √2 = -1/2 √2                         (3)
```

У цьому прикладі використовується корисна можливість системи Maple 12 присвоювати отриманим результатам пронумеровані мітки. Для того щоб вставити посилання-мітку, необхідно виконати команду головного меню **Insert → Label**… (або комбінацію клавіш **Ctrl+L**) і у відкритому віконці зазначити потрібний номер мітки.

Отриманому розв'язку можна встановити *статус виразу* з метою зручної подальшої роботи з ним. Так, з наступного прикладу бачимо, що розв'язку можна присвоїти ім'я, прирівнюючи його до функції **`solve`**:

```
> eq := sin(x) = -cos(x);
                    eq := sin(x) = -cos(x)                    (1)
> a = solve(eq, x);
                              a = -1/4 π                      (2)
> a;
                               a                              (3)
```

Але просте звертання до розв'язку через ім'я в даному випадку не спрацьовує, система «не розуміє» значення. Необхідно виконати команду **`assign(a)`** для перетворення імені на вираз:

```
> assign(%);
> a;
```

$$-\frac{1}{4}\pi \qquad (4)$$

Ця дія еквівалентна команді **a:=solve()**.

Отриманому розв'язку можна встановити *статус функції* командою **unapply**. Якщо розв'язок отримано в загальному, символьному, вигляді, то його статус як функції дозволить у подальшому виконувати підстановки та розраховувати значення при конкретних параметрах. Це ілюструє такий приклад:

```
> # створюємо список із розв'язків квадратного рівняння:
> sols := [solve(a·x^2 + b·x + c, x)];
```

$$sols := \left[\frac{1}{2}\,\frac{-b+\sqrt{b^2-4\,a\,c}}{a},\ -\frac{1}{2}\,\frac{b+\sqrt{b^2-4\,a\,c}}{a}\right]$$

```
> # перетворюємо перший розв'язок (перший елемент списку) на функцію, яка
    залежатиме від a, b, c:
> f := unapply(sols[1], a, b, c);
```

$$f := (a,b,c) \to \frac{1}{2}\,\frac{-b+\sqrt{b^2-4\,a\,c}}{a}$$

```
> # розрахуємо значення цього розв'язку при a = 1, b = 2, c = 3:
> f(1, 2, 3);
```

$$-1+\frac{1}{2}\sqrt{-8}$$

### 2.2.5. Розв'язування звичайних диференціальних рівнянь

Система Maple дає широкі можливості для розв'язання диференціальних рівнянь. Існує багато засобів для розв'язання як окремих рівнянь, так і їх систем, як звичайних диференціальних рівнянь (ODE – ordinary differential equations), так і диференціальних рівнянь у часткових похідних (PDE – partial differential equations). Можна використовувати засоби розв'язку в символьному або числовому вигляді з урахуванням початкових значень, граничних або крайових умов.

Розглянемо основні засоби для розв'язання диференціальних рівнянь – помічник та команду **dsolve** (див. також п. 2.6).

*2.2.5.1. Використання помічника*

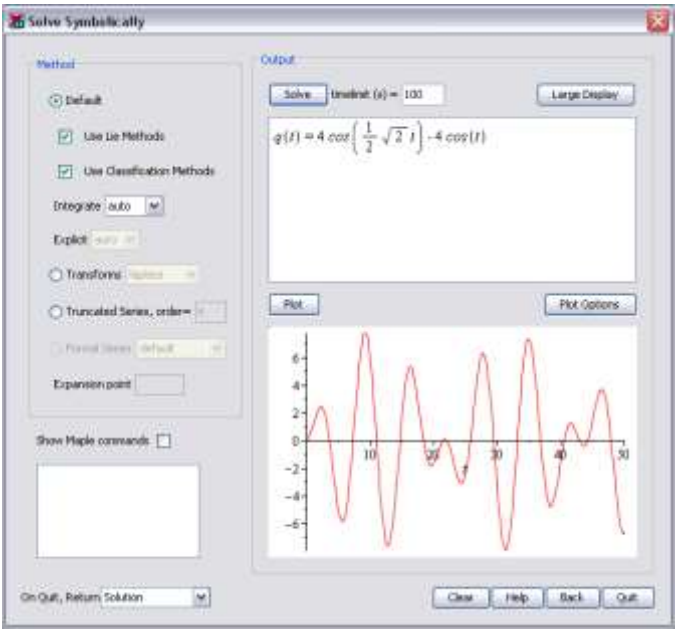
Помічник є інструментом для розв'язання ODE зі зручним інтерфейсом типу «point-and-click» («вказав та клацнув»). Він має назву **ODE Analyzer**, є аналогом команди `dsolve`, яку розглянемо наступним пунктом, та викликається з головного меню: **Tools → Assistants → ODE Analyzer**. Вигляд головного вікна помічника показаний на рис. 2.1. Воно містить три головні поля:

- Differential equations, в якому задається вигляд рівнянь для розв'язання;
- Conditions, в якому задаються початкові, граничні або крайові умови;
- Parameters, в якому задаються значення параметрів рівнянь, заданих символьно (система їх автоматично розпізнає).

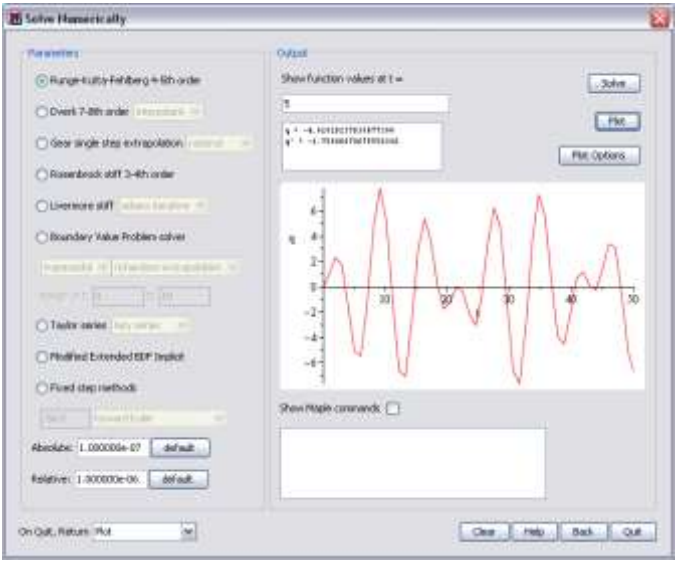
Зміст цих полів задається або редагується відповідними кнопками **Edit**.


Рисунок 2.1 – Вигляд головного вікна помічника **ODE Analyzer**

Після того як необхідні поля заповнені, необхідно обрати спосіб розв'язання – символьний або числовий – шляхом натискання на кнопку **Solve Symbolically** або **Solve Numerically** відповідно. Для кожного способу на екрані з'явиться своє віконце (рис. 2.2), в якому обирається метод розв'язання та виводиться сам розв'язок в аналітичному, числовому або графічному виглядах.




а




б

Рисунок 2.2 – Вікна символьного (а) та числового (б) розв’язувань диференціальних рівнянь

Елементи *символьного* вікна:

- Method – вибір методу розв'язання та за необхідності відповідних до методу параметрів;

- Output – поле для введення незалежної змінної, виведення аналітичного розв'язку, а при натисканні кнопки Plot – побудови графіка знайденої функції, параметри якого можна змінювати кнопкою Plot Options;

- Show Maple Commands – установлення даного прапорця призведе до виведення в лівому нижньому куті віконця Maple-команд, які відповідають діям даного помічника;

- On Quit, Return – вибір способу подання результату, що відобразиться в робочому документі після закриття віконця помічника.

Відмінні елементи *числового* вікна:

- Parameters – вибір числового методу розв'язання та за необхідності відповідних до методу параметрів; тут поля Absolute та Relative установлюють величину абсолютної та відносної похибок розрахунку;

- Output – поле для введення значення незалежної змінної та виведення розрахованих значень шуканої функції та її похідних у даній точці; при натисканні кнопки Plot будується графік знайденої функції, параметри якого можна змінювати кнопкою Plot Options.

Інші елементи збігаються з елементами символьного вікна.

Параметри, що за замовчуванням зазначені на полях Method або Parameters, є оптимальними для більшості задач.

*2.2.5.2. Використання команди* **`dsolve`**

Універсальною командою для розв'язання ODE та їх систем є команда **`dsolve`**, яка може знаходити аналітичні та числові розв'язки, а також розв'язки з розвиненням у ряд.

Синтаксис команди такий:

```
dsolve ({ODE, InCond}, {y(x)}, options)
```

Тут **`ODE`** – одне або декілька диференціальних рівнянь (в останньому випадку рівняння зазначаються списком **`[ ]`** або набором **`{ }`**);

**InCond** – початкові умови у вигляді **y(x0)=y0, D[y][x00]=y00** і т. ін.;

**y(x)** – шукана функція (одна чи список/набір функцій);

**options** – необов'язкові опції, що визначають метод розв'язання, наприклад: опція **series** – якщо використовується розвинення в ряд, та опція **method=laplace** – якщо використовується перетворення Лапласа; опція **numeric** – при числовому розв'язанні.

**Приклад 2.5.** Розв'язати диференціальне рівняння $\frac{d^2y}{dx^2} + 4\frac{dy}{dx} + 3y + 2 = 0$ без початкових умов та при $y(0) = 1, \frac{dy}{dx}(0) = 0$.

```
> ode1 := diff(y(x), x$2) + 4·diff(y(x), x) + 3·y(x) + 2;
```

$$ode1 := \frac{d^2}{dx^2} y(x) + 4\left(\frac{d}{dx} y(x)\right) + 3\,y(x) + 2$$

```
> dsolve(ode1);
```

$$y(x) = e^{-3x}\_C2 + e^{-x}\_C1 - \frac{2}{3}$$

```
> IC := D(y)(0) = 0, y(0) = 1;
```

$$IC := D(y)(0) = 0,\ y(0) = 1$$

```
> dsolve({ode1, IC}, y(x))
```

$$y(x) = -\frac{5}{6}e^{-3x} + \frac{5}{2}e^{-x} - \frac{2}{3}$$

Можемо бачити, що, якщо кількість початкових умов недостатня (або вони відсутні), система вводить до розв'язання власні невизначені константи, позначені як _C1, _C2. Для розв'язання задачі Коші необхідно включати початкові умови, в даному прикладі IC.

## 2.3. Обчислення з використанням одиниць вимірювання та наукових констант

Окрім перетворень із символьними та числовими величинами, Maple може проводити розрахунки з одиницями вимірювання. Велика бібліотека одиниць вимірювання, що підтримуються системою, а також засоби для їх використання при розрахунках знаходяться в пакеті **Units**. Пакет є гнучким, тобто за необхідності користувач може його доповнювати сам власними одиницями вимірювання. Крім того, через головне

меню **Tools → Tasks → Browse** можна знайти деякі готові шаблони задач, типових для випадків використання одиниць вимірювання.

### 2.3.1. Основні поняття про вимірювані величини та одиниці вимірювання

Вимірювані величини можна подати двома групами величин:

1) *основними величинами*, що обираються незалежними від інших так, що ніяка основна величина не може бути обрана через інші основні (такі, як довжина, маса, час, електричний струм, сила світла, термодинамічна температура, кількість речовини, інформація і т. ін.)

2) *похідними величинами*, що визначаються через комбінацію основних величин, наприклад, сила = маса · довжина/час$^2$.

Вираз, що пов'язує величину з основними, визначає її *розмірність*. Повний список величин або їх розмірностей, з якими працює Maple, можна знайти в довідковій системі, виконавши команду `Units[GetDimensions]()`.

Так, кожна величина, основна чи похідна, має власні одиниці вимірювання, які відповідно є основними або похідними. У таблиці 2.2 наведені приклади величин, їх вираження через основні розмірності та відповідні до них одиниці вимірювання.

Таблиця 2.2 – Деякі величини та їх одиниці вимірювання

| **Вимірювана величина** | **Відповідні основні розмірності** | **Приклад одиниць вимірювання** |
|---|---|---|
| Час | *Час* | Секунда, хвилина, година, день, місяць, рік |
| Енергія | $\frac{Довжина^2 \cdot маса}{час^2}$ | Джоуль, електрон-вольт, ерг, ват-година, калорія, британська теплова одиниця |
| Електричний потенціал | $\frac{Довжина^2 \cdot маса}{час^3 \cdot електричний\ струм}$ | Вольт, абвольт, статвольт |

Дізнатися повний перелік доступних одиниць вимірювання для даної величини можна в довідковій системі за допомогою команди `?Units/dimension` (dimension – назва величини або розмірності). Наприклад, для одиниць довжини команда має такий вигляд: `?Units/length`.

Одиниці вимірювання згруповані в різні системи одиниць, наприклад: СІ (SI) – міжнародна система одиниць, для якої незалежними одиницями є кілограм, метр, секунда, ампер, кельвін, моль та кандела; СГС (CGS) – система з трьома незалежними одиницями «сантиметр-грам-секунда»; ФФС (FPS) – Британська система «фут-фунт-секунда» та ін.

### 2.3.2. Дії з використанням одиниць вимірювання

*2.3.2.1. Перетворення одиниць вимірювання*

Для того щоб перетворити одні одиниці вимірювання на інші, використовується калькулятор одиниць. Він запускається в окремому віконці командою `?UnitsCalculator` або з головного меню **Tools → Assistants → Units Calculator**. Вигляд калькулятора наведено на рис. 2.3. У меню **Dimension**, що випадає, обирають величину (розмірність); в меню **From** – одиницю, яка перетворюється; в меню **To** – одиницю, на яку перетворюється дана одиниця; в полі **Convert** зазначають кількість, натискання кнопки **Perform Unit Conversion** починає розрахунок. Результат виводиться в полі **Result**.

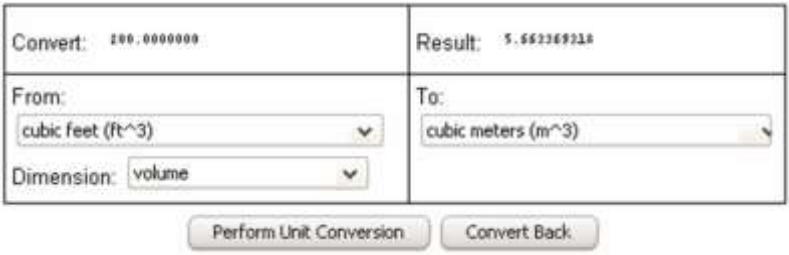



Рисунок 2.3 – Віконце калькулятора одиниць вимірювання (приклад – перетворення 200 футів кубічних на кубічні метри)

Перетворення одиниць вимірювання можна проводити в робочому документі з використанням еквівалентної команди Maple `convert()`. Ця команда є корисною для перетворення не лише одиниць вимірювання, а й будь-яких виразів (див. п. 2.1.4).

Перетворення температур має особливість. У Maple є можливість перетворити абсолютне значення температури за будь-якою шкалою, а також відносну зміну температур. Наступний приклад ілюструє синтаксис відповідних команд.

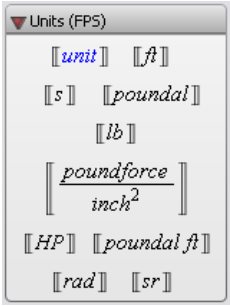
**Приклад 2.6:** а) перетворити 27 градусів температури за шкалою Цельсія на градуси Кельвіна; б) перетворити діапазон температур, що дорівнює 27 градусів Цельсія, на градуси Кельвіна; в) перетворити діапазон температур, що дорівнює 27 градусів Цельсія, на градуси за Фаренгейтом.

```
> convert(27.,'temperature','degC','kelvin');
                                  300.1500000      # а)
> convert(27.,'units','degC','kelvin');
                                  27.              # б)
> convert(27.,'units','degC','degF');
                                  48.60000000      # в)
```

### *2.3.2.2. Застосування одиниць вимірювання до виразу*

Для того щоб вставити одиниці вимірювання у вираз в робочому документі, необхідно використовувати спеціальні палітри **Units** (див. п. 1.3.1.3). Наприклад, **Units(SI)** для міжнародної системи одиниць СІ або **Units(FPS)** для системи ФФС. На рис. 2.4 а, б показано відповідно вигляд цих палітр.

а



б

Рисунок 2.4 – Палітри команд для вставки одиниць вимірювання

Для вставки будь-якої одиниці вимірювання достатньо на палітрі натиснути на її символ.

Зверніть увагу, що, якщо на палітрі немає символа необхідної одиниці вимірювання, спочатку вставляють у документ пусту позицію [[*unit*]], в яку потім вводять ім'я одиниці. Пуста позиція є на обох палітрах типу **Units**.

Необхідно пам'ятати, що якщо використовується одновимірний режим введення (1-D Math input), то значення величини та її одиниця вимірювання є *добутком*, а не цілісним об'єктом. Тобто, наприклад, швидкість 20 м/с повинна записуватись як `20*[[m]]/[[s]]`.

*2.3.2.3. Обчислення з використанням одиниць вимірювання*

Перед проведенням обчислень, в яких величини фігурують із власними одиницями вимірювання, необхідно підключити оточення Natural або Standard (останнє рекомендується) за допомогою команди `with(Units[Standard])`. Якщо цього не зробити, то можливі лише перетворення одиниць, а обчислення не дозволені.

**Приклад 2.7.** Швидкість руху об'єкта дається залежністю від часу у вигляді $t^2 + \sin t$. Обчислити переміщення та прискорення об'єкта.

Завдання полягає в інтегруванні та диференціюванні заданої функції.

```
> with(Units[Standard]):
> (t^2 + sin(t))*[[m]]/[[s]];
```

$$\left(t^2 + \sin(t)\right) \left[\!\left[\frac{m}{s}\right]\!\right] \qquad (1)$$

```
> int((1), t[[s]]);
```

$$\left(\frac{1}{3}t^3 - \cos(t)\right) [\![m]\!] \qquad (2)$$

```
> diff((1), t[[s]]);
```

$$(2t + \cos(t)) \left[\!\left[\frac{m}{s^2}\right]\!\right] \qquad (3)$$

*2.3.2.4. Зміна поточної системи одиниць*

Під час проведення обчислень усі одиниці вимірювання автоматично зводяться до одиниць, що відповідають поточній

системі одиниць. Дізнатися про вид поточної системи можна за допомогою команди `Units[UsingSystem]()`.

Системою, що використовується за замовчуванням, є система СІ. За необхідності можна змінити систему одиниць командою `Units[UseSystem](назва)`, де в круглих дужках зазначається ім'я системи, наприклад `SI, FPS, CGS`.

**Приклад 2.8.** Розрахувати постійну швидкість, з якою автомобіль за 3 години проїхав відстань 132 милі.

```
> with(Units[Standard]):
> Units[UsingSystem]()
```

$$SI \qquad (1)$$

```
> 132[mile]
```

$$132\ [mi] \qquad (2)$$

```
> (2)/3[hour]
```

$$\frac{61468}{3125}\left[\frac{m}{s}\right] \qquad (3)$$

Як бачимо, при підключенні оточення Standard система автоматично зводить одиниці вимірювання до поточної системи одиниць (СІ).

Зауважимо, що, як і для будь-якого пакета, для команд пакета **Units** можна застосовувати послідовності виклику різних типів (див. п. 1.3.8).

*2.3.2.5. Розширення можливостей*

У разі необхідності можна самостійно розширити набір розмірностей, одиниць вимірювання та навіть створити власну систему одиниць. Нижче наводяться назви команд пакета Units, що розширюють можливості Maple щодо цього питання:

- додати основну або похідну розмірність: `?Units[AddDimension]`;
- додати основну одиницю вимірювання: `?Units[AddBaseUnit]`;
- додати похідну одиницю вимірювання: `?Units[AddUnit]`;
- додати свою систему одиниць: `?Units[AddSystem]`.

### 2.3.3. Наукові константи та властивості хімічних елементів

Система Maple підтримує розрахунки не лише з одиницями вимірювання, але й із науковими константами та величинами,

що характеризують властивості хімічних елементів та їх ізотопів. Як і у випадку з одиницями вимірювання, система і тут є гнучкою, тобто дає можливості легкого доступу до вбудованої довідникової бази констант та створення власних.

Пакет **ScientificConstants** містить усю довідникову інформацію та засоби роботи з нею. Він може бути корисним для розрахунків у багатьох галузях фізики, хімії, інженерної справи і т. ін. Так, усі величини, що містяться в даному пакеті, можна поділити на дві категорії:

а) фізичні константи (наприклад, швидкість світла, маса та заряд електрона та ін.);

б) властивості хімічних елементів та ізотопів (атомні маси, порядкові номери елементів та ін.).

Для кращого розуміння змісту отримуваних рівнянь та перевірки розмірності цей пакет зберігає всі величини разом з одиницями вимірювання.

Ознайомитися з повним списком фізичних констант можна виконавши команду **?ScientificConstants/PhysicalConstants**. Таблиця 2.3 ілюструє деякі константи. З таблиці можна бачити, що до кожної константи можна звертатися як за її ім’ям, так і за скороченим символом.

Таблиця 2.3 – Приклади звертань до деяких констант

| **Константа** | **Звертання за ім’ям** | **Звертання за символом** |
|---|---|---|
| Гравітаційна стала | Newtonian_constant_of_gravitation | G |
| Постійна Планка | Planck_constant | h |
| Елементарний заряд | elementary_charge | e |
| Борівський радіус | Bohr_radius | a[0] |
| Постійна Авогадро | Avogadro_constant | N[A] |

Розглянемо, яку інформацію можна отримати про константи і як це зробити. По-перше, для кожної константи зберігаються її ім'я (name), символ (symbol), значення (value), одиниці вимірювання (units), похибка (uncertainty). Здобути інформацію можна командою **`GetConstant(ім'я_константи)`**:

```
> with(ScientificConstants) :
> GetConstant('h');
Planck_constant, symbol = h, value = 6.62606876 10^-34,
  uncertainty = 5.2 10^-41, units = J s
```

Аналогічно, можна знайти повний список хімічних елементів, які підтримуються системою, за командою **?ScientificConstants/elements**. Для кожного елемента зберігаються його ім'я, атомний номер (1-112, 114, 116), хімічний символ, атомна вага (atomicweight), густина (density), спорідненість з електроном (electronaffinity), та ін. Кожен параметр зберігається з одиницями вимірювання та похибками. Дістати загальну інформацію про властивості елементів можна за командою **?ScientificConstants/properties**. Детальну інформацію про доступні ізотопи елемента та властивості кожного з них можна отримати за командами відповідно **`GetIsotopes()`** та **`GetElement()`**.

**Приклад 2.9.** Отримати перелік ізотопів кисню, властивості кисню та одного з його стабільних ізотопів $^{16}_{8}O$.

```
> with(ScientificConstants) :
  GetIsotopes('element'='O');
    O12, O13, O14, O15, O16, O17, O18, O19, O20, O21, O22, O23, O24, O25, O26
> GetElement('O')
8, symbol = O, name = oxygen, names = {oxygen}, ionizationenergy = [value
    = 13.6181, uncertainty = undefined, units = eV], meltingpoint = [value = 54.36,
    uncertainty = undefined, units = K], density = [value = proc( ) ... end proc,
    uncertainty = undefined, units = L], atomicweight = [value = 15.9994, uncertainty
    = 0.0003, units = amu], electronaffinity = [value = 1.4611103, uncertainty
    = 7. 10^-7, units = eV], electronegativity = [value = 3.44, uncertainty
    = undefined, units = 1], boilingpoint = [value = 90.20, uncertainty = undefined,
    units = K]
> GetElement('O[16]') :
```

(Властивості ізотопу $^{16}_{8}O$ в розгорнутому вигляді не показані).

Для того щоб використовувати константи або властивості елементів/ізотопів, необхідно спочатку *створити* відповідний *об'єкт*. Об'єкт-константа створюється командою `Constant('ім'я')`. Наприклад, для гравітаційної сталої:

```
> with(ScientificConstants) :
> G := Constant('G');
                        G := Constant(G)
```

Тепер об'єкт з ім'ям *G* має значення, що дорівнює гравітаційний сталій:

```
> evalf(G);
                      6.673 10^-11
```

Якщо необхідно дізнатися одиниці вимірювання:

```
> GetUnit(G);
                        [[ m^3 / (kg s^2) ]]
```

Дізнатися про похибку:

```
> GetError(G)
                        1.0 10^-13
```

Щоб створити об'єкт, значення якого дорівнює величині певної властивості хімічного елемента, використовують команду `Element('ім'я елемента', ім'я властивості)`. Наприклад, для атомної ваги гелію:

```
> Prop := Element('He', atomicweight);
                    Prop := Element(He, atomicweight)
```

Тепер об'єкт з ім'ям *Prop* має значення, що дорівнює атомній вазі гелію:

```
> evalf(Prop);
                    6.646481526 10^-27
```

Щоб отримати значення властивості в одиницях вимірювання:

```
> evalf(Element('He', atomicweight, units));
                    6.646481526 10^-27 [[kg]]
```

## 2.4. Операції та функції математичного аналізу

Область задач математичного аналізу широко покривається засобами системи Maple. За допомогою нескладних maple-функцій можна обчислювати похідні, інтеграли, проводити аналіз функцій і т. д. Шаблони розв'язання деяких задач (task templates) можна знайти в головному меню **Tools → Tasks → Browse**. У даному підрозділі ми розглянемо ключові maple-команди, більшість яких використовується в шаблонах задач або доступна через контекстні меню.

### 2.4.1. Обчислення границь функцій

Для обчислення границі функції $\lim_{x\to a}\left(f\left(x\right)\right)$, тобто величини, до якої функція *f*(*x*) наближається при наближенні її незалежного аргументу *x* до значення *a*, можна використовувати або команди, або готовий шаблон. Палітра шаблонів команд Expression містить шаблон  . Після його вставки в документ необхідно підставити власну функцію, незалежну змінну, значення точки *a* і натиснути **Enter**. Цей шаблон еквівалентний команді `> limit(function, x=a, direction)`, де необов'язковий параметр *direction* означає напрямок наближення до точки *a:* справа – '*right*' чи зліва – '*left*'. За замовчуванням, коли не заданий *direction*, система шукає границю при наближенні з обох боків.

**Приклад 2.10.** Робота функції `limit`. Необхідність уточнення напрямку наближення на прикладі функції $f\left(x\right)=1/x$:

```
> limit(1/x, x=0);
                        undefined

> limit(1/x, x=0, 'right');
                            ∞

> limit(1/x, x=0, 'left');
                           -∞
```

При записі команди `limit` для нескінченності використовують або слово infinity (також -*infinity*), або символ із палітри шаблонів *Common Symbols*.

Якщо ви працюєте в класичному інтерфейсі або в стандартному інтерфейсі в режимі 1-D Math Input (у цей режим можна перейти так: **Format → Convert to → 1-D Math Input**), то майже для всіх maple-функцій математичного аналізу є їх абсолютний аналог, що лише записується з великої літери. Такі аналоги являють собою заданий вираз у звичному математичному вигляді. Наприклад, порівняйте для функції limit:

```
> Limit(1/x, x = infinity);
```

$$\lim_{x \to \infty} \frac{1}{x}$$

```
> limit(1/x, x = infinity);
```

$$0$$

### 2.4.2. Обчислення похідних. Функція diff

Існує декілька шляхів обчислення похідних. Найпростіший із них – використання шаблонів $\frac{d}{dx} f$ або $\frac{\partial}{\partial x} f$ (останній для частинної похідної) із палітри команд *Expression*.

Основною Maple-командою, еквівалентом якої є ці шаблони, є команда **`diff(function, x1, x2, …, xn)`**, де **`x1, x2, …, xn`** – послідовність незалежних змінних функції function, за якими шукають похідну.

Нехай function – функція однієї змінної. Якщо потрібно знайти похідну першого порядку, то використовують запис **`diff(f(x))`**. Для похідної вищих порядків, наприклад порядку 3, використовують запис з оператором послідовності $: **`diff(f(x), x$3)`**, що еквівалентно **`diff(f(x), x, x, x)`** або **`diff(diff(diff(f(x), x), x), x)`**.

Нехай function – функція декількох змінних. Тоді шукають частинні похідні. Наприклад, для обчислення похідної вигляду $\frac{\partial^5 f(x,y)}{\partial x^3 \partial y^2}$ використовують запис diff (*f*(*x, y*), *x*$3, y$2).

Для режиму 1-D Math Input також має місце аналогічна функція Diff, що лише відображає зручний математичний вигляд похідної, але не знаходить її:

```
> diff(sin(x), x);
```
$$\cos(x)$$
```
> Diff(sin(x), x);
```
$$\frac{d}{dx}\sin(x)$$

### 2.4.3. Обчислення похідних. Диференціальний оператор D

Для створення функцій із похідними можна використовувати спеціальний оператор – *диференціальний оператор D*. Він дозволяє створювати більш компактні вирази, ніж функція diff. Оператор має два формати запису:

`D(f)`, якщо `f` – функція одного аргументу;

`D[i](f)`, якщо `f` – функція деяких аргументів;

де `f` – ім'я функції або її вираз; `i` – додатне ціле число, вираз або їх послідовність, за якими проводиться диференціювання.

Для того щоб створити похідну, необхідно даним оператором подіяти на певну змінну (змінні). За своїм змістом запис `D(f)(x)` еквівалентний запису `diff(f(x), x)`.

**Приклад 2.11.** Знайти похідну функції $e^x$ та *sin(x)*:

```
> D(exp+sin);
```
$$\exp + \cos$$
```
> D(exp+sin)(x);
```
$$e^x + \cos(x)$$
```
> D(exp+sin)(0);
```
$$2$$

**Приклад 2.12.** Знайти частинну похідну функції трьох змінних за кожною змінною:

```
> f:= (x,y,z) -> x*exp(y)+ln(z);
```
$$f := (x,y,z) \to x\,e^y + \ln(z)$$
```
> D[1](f);
```
$$(x,y,z) \to e^y$$
```
> D[2](f);
```
$$(x,y,z) \to x\,e^y$$
```
> D[3](f);
```
$$(x,y,z) \to \frac{1}{z}$$
```
> D[](f);
```
$$f$$

Як бачимо з останнього рядка прикладу, якщо квадратні дужки залишають пустими, то похідна не шукається.

Для знаходження похідної вищих порядків використовується запис вигляду `(D@@n)(f)`, де `n` – порядок похідної. Наприклад, два наступні вирази є еквівалентними:

```
> D(D(f));
```
$$(D^{(2)})(f)$$
```
> (D@@2)(f);
```
$$(D^{(2)})(f)$$

Уточнимо різницю між диференціальним оператором D та функцією diff. Оператор D знаходить похідні від операторів, а функція diff – від виразів. Отже, аргументом і результатом D є функціональні оператори, а аргументом і результатом diff є вирази. Поняття функціонального оператора ми розглядали в п. 1.4.2.4. Існує також спеціальна функція **unapply**, що перетворює вираз на функцію (функціональний оператор):

**Приклад 2.13.** Перетворення виразу *a* на функціональний оператор *f*:

```
> a := exp(x) + sin(x);
```

$$a := e^x + \sin(x)$$

```
> f := unapply(a,x);
```

$$f := x \to e^x + \sin(x)$$

```
> f(Pi);
```

$$e^{\pi}$$

## 2.4.4. Обчислення інтегралів

У системі Maple можна обчислювати невизначені та визначені інтеграли. Найпростіший спосіб обчислення – використання шаблонів із палітри команд Expression, а саме $\int f\,dx$ для невизначених інтегралів та $\int_a^b f\,dx$ – для визначених. Ці шаблони використовують відповідно команду **int(f,x)** або **int(f, x=a..b)**. Для режиму 1-D Math Input можна використовувати запис цієї команди з великої літери для візуалізації виразу.

**Приклад 2.14.** Знайти визначений (на інтервалі –π..π) та невизначений інтеграли від функції *tg(x)*:

Невизначений інтеграл:

```
> int(tan(x), x);
```

$$\ln(\cos(x))$$

Визначений інтеграл:

```
> int(tan(x), x = Pi..Pi);
```

$$undefined$$

Знаходження визначеного інтеграла ускладнюється тим, що тангенс має розриви в точках $\pi/2 \pm \pi n$. У подібних випадках потрібно у функції **int** зазначати опцію **'continuous'**, що

задасть "не шукати точки розриву". У результаті інтеграл буде обчислений:

```
> int(tan(x), x=-π..π,'continuous');
                              0
```

Якщо Maple не може знайти інтеграл, то необхідно розвинути підінтегральну функцію в ряд (див. п. 2.4.6).

### 2.4.5. Обчислення сум та добутків послідовностей

Суму послідовності вигляду

$$\sum_{k=m}^{n} f(k) = f(m) + f(m+1) + \ldots + f(n-1) + f(n)$$

обчислюють функцією **`sum(f, k=m..n)`**, де f – вираз членів ряду, сумування відбувається для параметра k, що змінюється від m до n.

Добуток вигляду

$$\prod_{k=m}^{n} f(k) = f(m) \cdot f(m+1) \cdot \ldots \cdot f(n-1) \cdot f(n)$$

обчислюють функцією **`product(f, k=m..n)`** .

Обидві функції мають аналоги, що записуються з великої літери, **`Sum`** та **`Product`** для візуального подання операції. Також вставити дані операції в робочий документ можна через шаблони команд  та  палітри *Expression*.

### 2.4.6. Розвинення функції в ряд

Задача розвинення функцій у ряд може бути розв'язана декількома засобами Maple.

Широко використовується розвинення в ряд Тейлора, що має вигляд

$$f(x) = f(x = x_0) + \frac{1}{1!}\frac{\partial^1 f}{\partial x^1}\bigg|_{x=x_0} (x - x_0)^1 + \frac{1}{2!}\frac{\partial^2 f}{\partial x^2}\bigg|_{x=x_0} (x - x_0)^2 + \ldots$$

Для розвинення в ряд Тейлора виразу expr за змінною x поблизу точки x = a використовується функція **taylor(expr, x=a, n)**, де n – натуральне число, що визначає порядок розкладання. Якщо n не заданий, то порядок розвинення береться таким, що дорівнює значенню системної змінної **Order** (за замовчуванням **Order** = 6). Порядок n означає, що максимальний степінь доданків ряду буде n – 1.

Щоб отримати розвинення в ряд Маклорена вигляду

$$f(x) = f(x=0) + \frac{1}{1!}\frac{\partial^1 f}{\partial x^1}\bigg|_{x=0} x^1 + \frac{1}{2!}\frac{\partial^2 f}{\partial x^2}\bigg|_{x=0} x^2 + \ldots,$$

використовують функцію **taylor** при x = 0.

Загальною функцією для розкладання в ряд є **series(expr, x=a, n)**. Часто дана функції дозволяє отримати розкладання в ряд, коли функція **taylor** не спрацьовує, наприклад, якщо вираз має точки розриву:

```
> taylor (tan(x),x=Pi/2);
Error, does not have a taylor expansion, try series()

> series (tan(x),x=Pi/2);
```

$$-\left(x-\frac{\pi}{2}\right)^{-1}+\frac{1}{3}\left(x-\frac{\pi}{2}\right)+\frac{1}{45}\left(x-\frac{\pi}{2}\right)^{3}+\frac{2}{945}\left(x-\frac{\pi}{2}\right)^{5}+O\left(\left(x-\frac{\pi}{2}\right)^{6}\right)$$

Цей приклад показує, що, оскільки *tgx* має розриву точці *π/2*, функція **taylor** видає помилку, а **series** знаходить результат. Крім того, в отриманому ряді доданок вигляду O(x) є залишковим членом.

Розглянемо більш складний випадок розвинення в ряд функції за декількома змінними. Цю задачу розв'язує функція **mtaylor (expr, v, n, w)**. Її аргументами є: expr – вираз, який розвивають; v – список змінних; n – порядок розвинення; w – список натуральних чисел, які визначають вагу кожної змінної при розвиненні. Вага визначає, що доданки ряду зі степенем даної змінної, більшим за n/w, відкидаються.

**Приклад 2.15.** Порівняємо результати розвинення в ряд функції sin(x + y) при різних значеннях параметра ваги w для змінних x та y:

```
> mtaylor(sin(x + y), [x, y], 4, [1, 1]);
```

$$x + y - \frac{1}{6}x^3 - \frac{1}{2}yx^2 - \frac{1}{2}y^2x - \frac{1}{6}y^3$$

```
> mtaylor(sin(x + y), [x, y], 4, [2, 1]);
```

$$y + x - \frac{1}{6}y^3$$

```
> mtaylor(sin(x + y), [x, y], 4, [1, 2]);
```

$$x + y - \frac{1}{6}x^3$$

Для деяких специфічних розвинень є власні функції, як, наприклад, **`asympt(expr, x, n)`** для асимптотичного розвинення при $x \to \infty$, **`poisson(expr, x)`** для розвинення тригонометричних функцій у ряд Фур'є.

Отримане розвинення має тип series. Деякі команди Maple «не розуміють» такого типу даних, тому необхідно ряд перетворювати на інший тип, наприклад на polynom (поліном).

**Приклад 2.16.** Побудувати графік розвинення в ряд функції sin(*x*) поблизу точки *x* = 0.

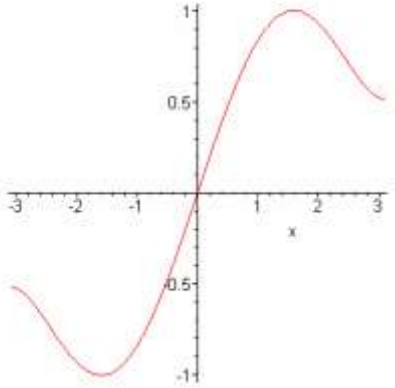
```
> s1:=taylor (sin(x),x=0);
```

$$s1 := x - \frac{1}{6}x^3 + \frac{1}{120}x^5 + O(x^6)$$

```
> s2:=convert(s1,polynom);
```

$$s2 := x - \frac{1}{6}x^3 + \frac{1}{120}x^5$$

```
> plot(s2,x=-Pi..Pi);
```



### 2.4.7. Інтерполяція та апроксимація функцій і даних

Якщо деяка функція $y(x)$ представлена низкою табличних даних $y_i(x_i)$, то *інтерполяція* – це обчислення значень $y(x)$ при $x$, заданому в інтервалі між точками $x_i$. За межами мінімального та максимального із заданих значень x, тобто при $x < x_{min}$ та $x > x_{max}$, обчислення значень функції називається екстраполяцією.

*Апроксимацією* в системах комп'ютерної математики називають отримання наближених значень певного виразу. *Апроксимація функції* – отримання конкретної функції, значення якої з певною точністю відповідають вихідній залежності. Як правило, віддають перевагу знаходженню однієї залежності, що наближає задану послідовність вузлових точок. Для цього часто використовують степеневі многочлени (поліноми).

*2.4.7.1. Апроксимація функцій, заданих аналітично*

Нехай заданий аналітичний вираз певної функції $y = y(x)$ і необхідно знайти наближений спрощений аналітичний вираз цієї функції. У такому разі вдаються до розвинення вихідної функції в степеневий ряд, що можливо робити поблизу деякого значення змінної $x$. Розглянемо цю процедуру на такому прикладі.

**Приклад 2.17**. Отримати наближений вираз функції $exp(x)$.

Першим кроком розвинемо $exp(x)$ в ряд степеня 5 поблизу точки $x = 0$:

```
> p1 := taylor(exp(x), x = 0);
```

$$p1 := 1 + x + \frac{1}{2}x^2 + \frac{1}{6}x^3 + \frac{1}{24}x^4 + \frac{1}{120}x^5 + O(x^6)$$

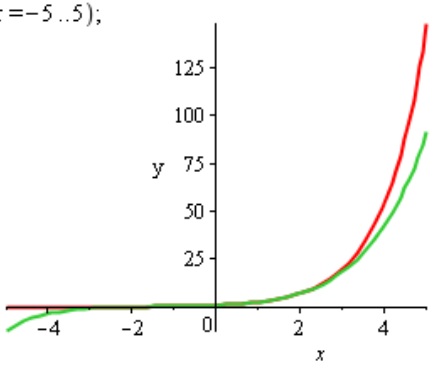
Доданок $O(x^6)$ є невідомим залишковим членом і його необхідно позбутися. Перетворимо отриманий ряд на поліном:

```
> p2 := convert(p1, polynom);
```

$$p2 := 1 + x + \frac{1}{2}x^2 + \frac{1}{6}x^3 + \frac{1}{24}x^4 + \frac{1}{120}x^5$$

Порівняємо вихідну функцію та отримане наближення графічно:

```
> plot([exp(x), p2], x=-5..5);
```




Як бачимо, обидві функції добре збігаються поблизу точки розвинення $x = 0$.

*2.4.7.2. Поліноміальна інтерполяція табличних даних*

Нехай дані деякої залежності $y(x)$ задані векторами $X$ та $Y$ її дискретних значень:

| $X$ | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | … | $x_n$ |
|---|---|---|---|---|---|---|---|
| $Y$ | $y_1$ | $y_2$ | $y_3$ | $y_4$ | $y_5$ | … | $y_n$ |

.

Тоді можна отримати інтерполяційний степеневий многочлен, або поліном, за допомогою команди **`interp(X, Y, var)`**, де var – ім'я змінної, яке буде фігурувати в поліномі. Зверніть увагу, що степінь полінома залежить від кількості заданих точок: якщо їх кількість $N$, то степінь полінома буде $N$ – 1.

**Приклад 2.18**. Функція задана таблично:

| $X$ | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| $Y$ | 0,5 | 1 | 2 | 2 | 1 |

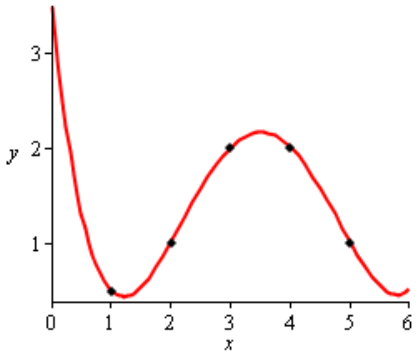
.

Знайти вираз інтерполяційного полінома. Розрахувати значення функції (полінома) в точці $x$=1,5, яка не належить до табличних даних. Побудувати графік полінома та вихідних точкових даних в одних координатах.

```
> X := [1, 2, 3, 4, 5] : Y := [0.5, 1, 2, 2, 1] :
  Digits := 4 :
  f1 := interp(X, Y, x);
```

$$f1 := 0.06250\,x^4 - 0.8750\,x^3 + 3.938\,x^2 - 6.125\,x + 3.500$$

```
> f2 := proc(a) RETURN(subs(x = a, f1)) end proc:
> f2(1.5);
                                  0.535
> p1 := plot(f1, x = 0..6):
  p2 := plots[pointplot](X, Y):
  plots[display](p1, p2);
```




У даному прикладі зверніть також увагу, що поліном перетворюється на процедуру, для того щоб у подальшому можна було обчислювати його значення для будь-якого *x* та будувати графік. Для побудови графіків створено два графічні об'єкти з іменами *p*1 (графік полінома) та *p*2 (точковий графік `pointplot` вихідних даних), які в подальшому виводяться на екран одночасно в одній системі координат функцією `display`. Функції `pointplot` та `display` належать до пакета команд **plots**.

*2.4.7.3. Сплайн-інтерполяція та апроксимація*

Зі збільшенням степеня апроксимувальних поліномів точність поліноміальної апроксимації значно зменшується. У такому разі використовують відрізки поліномів невисокого степеня, або сплайни. Це збільшує точність апроксимації, але додає інший недолік, а саме відсутність спільного виразу для всієї кривої.

Сплайн-апроксимація виконується командою `Spline(X, Y, var, degree = n)`, яка належить до пакета команд **CurveFitting**. X та Y – одновимірні вектори однакового розміру, які мають значення координат вузлових точок вихідної функції; var – ім'я змінної, що буде використовуватися в сплайні; n – необов'язковий параметр, який задає вид (степінь) сплайна: 1

або linear, 2 або quadratic, 3 або cubic, 4 або quartic. За замовчуванням, якщо опція degree = n відсутня, використовується кубічний сплайн (cubic).

**Приклад 2.19.** Задана така залежність:

| $x$ | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| $y$ | 0 | 1 | 4 | 3 | 2 | 1 |

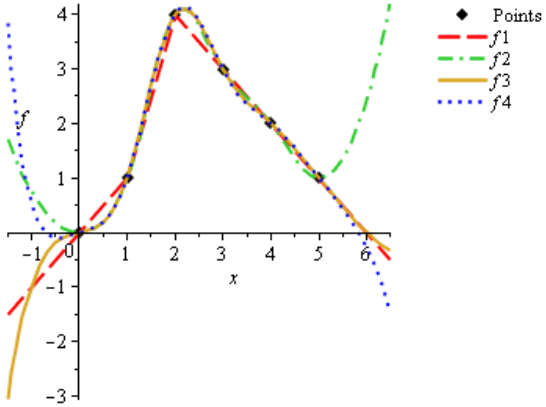
.

Побудувати апроксимацію цієї залежності чотирма видами сплайна. Зобразити результати графічно.

```
> with(CurveFitting):
> X := [0, 1, 2, 3, 4, 5]:
  Y := [0, 1, 4, 3, 2, 1]:
> f1 := Spline(X, Y, x, degree = 1);
```

$$f1 := \begin{cases} x & x < 1 \\ -2 + 3x & x < 2 \\ 6 - x & x < 3 \\ 6 - x & x < 4 \\ 6 - x & otherwise \end{cases}$$

```
> f2 := Spline(X, Y, x, degree = 2):
  f3 := Spline(X, Y, x, degree = 3):
  f4 := Spline(X, Y, x, degree = 4):
  p1 := plots[pointplot](X, Y):
> p2 := plot([f1, f2, f3, f4], x = -1.5..6.5);
  plots[display](p1, p2);
```




Тут для сплайнів порядку 2, 3, 4 опущені математичні вирази. Бачимо, що в межах інтервалу координат вузлових точок сплайни з різною точністю, але задовільно описують задану залежність. За межами крайніх точок існує дуже істотний розбіг в екстраполяції даних.

*2.4.7.4. Апроксимація методом найменших квадратів*

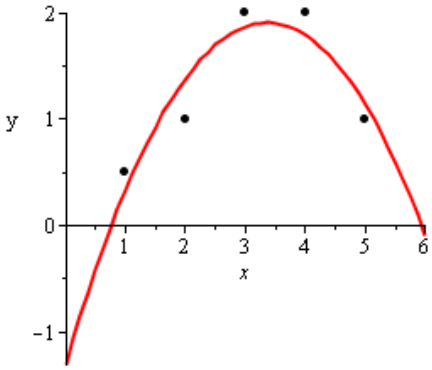
Суть апроксимації цим методом полягає у статистичній обробці всіх даних, виходячи з мінімуму середньоквадратичної похибки. Для цього використовується команда **`LeastSquares`** з пакета **CurveFitting**, синтаксис якої такий: **`LeastSquares(data, var, option)`**, де data – вихідні дані, задані координатами точок у вигляді списків, масивів або векторів, наприклад: [[$x_1$, $y_1$], [$x_2$, $y_2$], …] чи [$x_1$, $x_2$, …], [$y_1$, $y_2$, …]; var – ім'я змінної; option – опція, яка задає вигляд функції у вигляді *curve = expression*. Розглянемо приклад:

```
> X := [1, 2, 3, 4, 5] : Y := [0.5, 1, 2, 2, 1] :
  with(CurveFitting) :
  LeastSquares(X, Y, x, curve = a·x^2 + b·x + c);
```

$$-1.300 + 1.914\,x - 0.2857\,x^2$$

Тут ми використовуємо апроксимальну функцію у вигляді квадратичної параболи. Нижче порівняємо візуально вихідні дані з отриманою функцією.

```
> p1 := plot(f, x = 0..6) :
  p2 := plots[pointplot](X, Y) :
  plots[display](p1, p2);
```




## 2.4.8. Дослідження аналітичних функцій

Аналітичні функції широко використовуються у фізиці та математиці. Розглянемо аналіз функцій на неперервність, екстремуми, максимальне та мінімальне значення на відрізку.

Для розв'язування таких задач можна використовувати декілька шляхів, наприклад, через пошук похідних за правилами математичного аналізу або безпосередньо за допомогою команд Maple.

Для *пошуку точок екстремуму* функції є команда **`extrema(expr, constraints, vars, 'v')`**. Її аргументами є: expr – вираз, що досліджується; constraints – обмеження; vars – незалежна змінна або список незалежних змінних; v - ім'я змінної, якій буде функцією **`extrema`** присвоєне значення абсциси точки екстремуму. Сама функція **`extrema`** повертає значення ординати точки екстремуму. Обмеження constraints являють собою певні вирази або рівняння у вигляді $f(x) = 0$. У найпростішому випадку замість обмежень зазначають пусті фігурні дужки {}. Наприклад:

```
> extrema(sin(x)^2, { }, x, 's');
                                  {0, 1}
> s;
                          {{x = 0}, {x = 1/2 π}}
```

Так, у даному випадку маємо дві точки екстремуму з координатами (0; 0) та (π/2; 1). Причому досліджувана функція $\sin^2 x$ є періодичною з періодом π, і точки екстремуму знайдені для одного періоду $x \in [0;\pi]$.

Якщо необхідно *дослідити функцію на максимальне або мінімальне значення* на певному відрізку, то потрібно мати на увазі, що ці значення не завжди збігаються з екстремальними точками. Наприклад, для тієї ж функції $\sin^2 x$ на відрізку [2; 4] максимальним буде значення в точці А (2; 0,83), а мінімальним буде значення в точці екстремуму В (π; 0) (див. рис. 2.5).

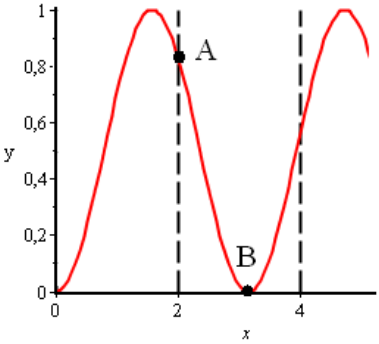



Рисунок 2.5 – Графік функції $y = \sin^2 x$

Для пошуку максимального значення функції на відрізку використовують команду **`minimize(expr, vars, ranges)`**, а для пошуку максимального значення – аналогічну команду **`maximize(expr, vars, ranges)`**, де expr – вираз, що досліджується; vars – змінні; ranges – відрізок значень змінної у вигляді $x = a..b$ (за замовчуванням $x = -\infty..\infty$). Наприклад:

```
> maximize(sin(x)^2, x = 2..4);
                              sin(2)^2
> evalf(%, 2);
                                0.83
> minimize(sin(x)^2, x = 2..4);
                                0
```

Для *аналізу функції на неперервність* використовують такі дві команди Maple: **`iscont`** та **`discont`**. Перша з них відповідає на запитання "чи є функція неперервною на певному відрізку?" Тому її результатом є логічне значення true (так) або false (ні). У разі, якщо функція перервна, тобто відповідь false, можна визначити координати точок розриву за допомогою другої команди **`discont`**.

Синтаксис цих команд є таким: **`iscont(expr, ranges, 'closed')`** та **`discont(expr, var)`**, де expr – вираз, що досліджується; ranges – відрізок значень змінної у вигляді $x = a..b$, необов'язкова опція 'closed' свідчить про те, що крайові

точки відрізку також перевіряються; var – ім'я змінної. При цьому команда **discont** шукає точки розриву для всіх дійсних значень аргументу var. Наприклад:

```
> f := 1/((x-1)*(x+2)):
> iscont(f, x=-3..3, 'closed');
                                  false
> discont(f, x);
                                 {-2, 1}
```

## 2.5. Робота з матрицями та векторами

### 2.5.1. Створення векторів та матриць

Задачі лінійної алгебри є одними з найпоширеніших у техніці, науці та освіті. Розглянемо у цьому підрозділі основні засоби Maple для роботи з такими структурами даних, як вектори та матриці, а також розглянемо їх використання для розв'язання систем лінійних рівнянь.

***Матрицею*** розміром m×n будемо називати прямокутну двовимірну таблицю, що містить m рядків та n стовпчиків елементів, кожен з яких може бути числом, константою, змінною, символьним або математичним виразом.

***Вектором*** будемо називати одновимірну матрицю, яка містить лише один рядок або один стовпчик елементів.

Створення векторів та матриць детально описано в п. 1.5.2.6.

Розглянемо процес створення матриці розміром більше ніж 10×10 за допомогою палітри шаблонів **Matrix** (п. 1.5.2.6). Після введення кількості рядків та стовпчиків у шаблон, наприклад 11×3, в робочому документі отримаємо вставку об'єкта, який має назву placeholder та заміщує матрицю:

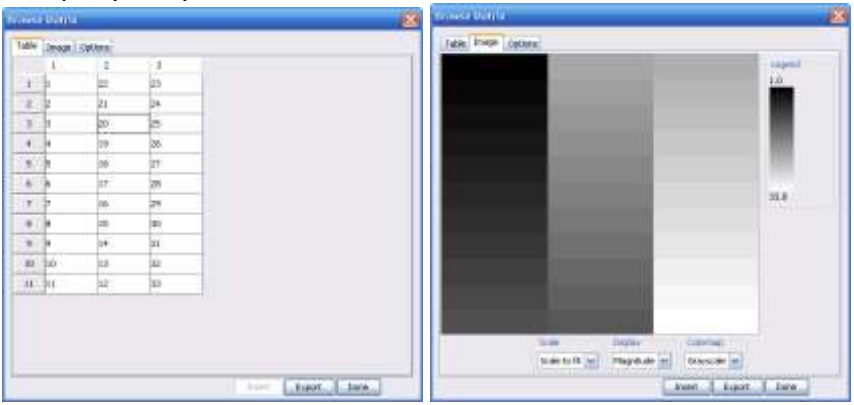
```
> [ 11 x 3 Matrix
  Data Type: anything
  Storage: rectangular
  Order: Fortran_order ]:
```

Подвійний клік на заміщувальному елементі відкриває віконце перегляду та редагування змісту матриці (див. рис. 2.6) – Browse Matrix. Це віконце містить три вкладки:

- Table – для введення даних;
- Image – для відображення даних у вигляді картинки;
- Options – для транспонування матриці.

Вкладка Table має вигляд електронної таблиці (див. рис. 2.6 а), яку необхідно заповнити даними та можна експортувати до Excel.

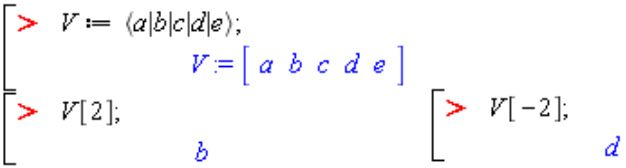
Вкладка Image відображає дані у вигляді кольорових комірок (див. рис. 2.6 б), де числова величина даних відповідає інтенсивності кольору обраної кольорової шкали. Кнопка Insert дозволяє вставити цю картинку в робочий документ. Меню Scale задає масштаб відображення, меню Display задає тип відображення картинки: magnitude – інтенсивність кольору відповідатиме значенню, structure – відображення нульових та ненульових елементів; в меню Colormap можна вибрати кольорову гаму.



а б

Рисунок 2.6 – Типовий вигляд вікна для перегляду та редагування матриць: а – вкладка Table; б – вкладка Image

Для звертання до елемента вектора, необхідно зазначити його ім’я та в квадратних дужках – індекс. Причому якщо рахувати з кінця, то індекс від'ємний:




Для створення нового вектора з визначеного раніше необхідно задати список або інтервал індексів потрібних елементів:


Аналогічні правила стосуються доступу і до елементів матриць із тією відмінністю лише, що зазначаються два індекси: перший індекс – індекс рядка, а другий – стовпчика, наприклад, M[1,2].

Для великих задач рекомендовано збільшувати ефективність обчислень шляхом створення векторів та матриць із властивостями, наприклад зазначати тип даних або тип матриці.

### 2.5.2. Обчислення з векторами та матрицями. Пакет LinearAlgebra

Усі обчислення з даними об'єктами можна проводити за допомогою елементарних операцій, контекстного меню та команд спеціалізованого пакета **LinearAlgebra**.

Для векторів і матриць застосовують такі елементарні операції:

- додавання (оператор +);
- віднімання (оператор -);
- скалярне множення (оператор *, що перетворюється на · );
- некомутативне множення (оператор . );
- піднесення до степеня (оператор ^);
- транспонування (оператор ^%T).

Для матричної алгебри немає спеціального оператора ділення. Його можна замінити множенням на обернену матрицю, тобто матрицю в степені –1.

**Приклад 2.20.** Операції з матрицями.

$$> \ a := \begin{bmatrix} 2 & 3 \\ 4 & 5 \end{bmatrix} : b := \begin{bmatrix} 5 & 7 \\ 8 & 9 \end{bmatrix} :$$

Скалярне множення:

$$> \ a \cdot 5;$$

$$\begin{bmatrix} 10 & 15 \\ 20 & 25 \end{bmatrix}$$

Некомутативне множення:

$$> \ a.b;$$

$$\begin{bmatrix} 34 & 41 \\ 60 & 73 \end{bmatrix}$$

Транспонування:

$$> \ a^{\%T};$$

$$\begin{bmatrix} 2 & 4 \\ 3 & 5 \end{bmatrix}$$

Піднесення до степеня:

$$> \ b^{-1};$$

$$\begin{bmatrix} -\frac{9}{11} & \frac{7}{11} \\ \frac{8}{11} & -\frac{5}{11} \end{bmatrix}$$

Пакет **LinearAlgebra** містить команди для різноманітних дій із векторами та матрицями, наприклад, створення та виконання перетворень, стандартних операцій та розв'язування задач лінійної алгебри. Більшість команд цього пакета є специфічними і детального розгляду в даному курсі не потребують. У таблиці 2.4 наведені вибрані та найбільш використовувані команди.

Розглянемо декілька прикладів використання команд пакета **LinearAlgebra**.

**Приклад 2.21.** Створення матриці 2×3 з випадковими елементами.

```
> with(LinearAlgebra):
> R := RandomMatrix(2, 3);
```

$$R := \begin{bmatrix} 99 & 44 & -31 \\ 29 & 92 & 67 \end{bmatrix}$$

У наступних прикладах вважатимемо, що пакет **LinearAlgebra** вже підключено.

Таблиця 2.4 – Вибрані команди пакета **LinearAlgebra**

| | |
|---|---|
| **Basis** | Повертає базис для векторного простору |
| **CharacteristicPolynomial** | Будує характеристичний поліном матриці |
| **CrossProduct** | Обчислює векторний добуток двох векторів |
| **DeleteRow** | Видаляє рядки матриці |
| **Determinant** | Обчислює детермінант матриці |
| **Dimension** | Обчислює розмірність матриці чи вектора |
| **DotProduct** | Обчислює скалярний добуток двох векторів |
| **Eigenvalues** | Обчислює власне значення матриці (розв’язки характеристичного рівняння) |
| **Eigenvectors** | Обчислює власні вектори матриці |
| **GaussianElimination** | Розв'язання системи лінійних алгебраїчних рівнянь (СЛАР) методом виключення Гауса |
| **LeastSquares** | Розв'язання СЛАР методом найменших квадратів |
| **LinearSolve** | Обчислює апроксимацію $A \cdot x = B$ методом найменших квадратів |
| **Map** | Застосовує певну операцію до виразу |
| **MatrixInverse** | Знаходить обернену матрицю для квадратної матриці |
| **MatrixScalarMultiply** | Знаходить добуток матриці та скаляра |
| **RandomMatrix** | Створює випадкову матрицю |
| **ReducedRowEchelonForm** | Розв'язання СЛАР методом Жордана–Гауса |
| **SubMatrix, SubVector** | Добуває підматрицю (вектор) із даної матриці |
| **Transpose** | Транспонує матрицю або вектор |

**Приклад 2.22.** Створення матриці 2×2, елементи якої залежать від власних індексів рядка та стовпчика, наприклад степеневою залежністю:

```
> f := (i,j) -> x^(i+j);
```

$$f := (i,j) \to x^{i+j}$$

```
> A := Matrix(2, 2, f);
```

$$A := \begin{bmatrix} x^2 & x^3 \\ x^3 & x^4 \end{bmatrix}$$

**Приклад 2.23.** Створення матриці 2×2 та диференціювання її елементів.

```
> B := Matrix([[3, exp(x)], [sin(x), x^n]]);
```

$$B := \begin{bmatrix} 3 & e^x \\ \sin(x) & x^n \end{bmatrix}$$

```
> map(diff, B, x);
```

$$\begin{bmatrix} 0 & e^x \ln(e) \\ \cos(x) & \dfrac{x^n n}{x} \end{bmatrix}$$

**Приклад 2.24.** Розв'язування системи лінійних алгебраїчних рівнянь матричним методом.

Рівняння можна подати у матричному вигляді як ***M · X = V***. Матриця *M* та вектор *V* задані, необхідно знайти вектор розв'язків *X*.

```
> M := Matrix([[0, 3, 1], [7, -10, -2], [1, 2, 4]]);
```

$$M := \begin{bmatrix} 0 & 3 & 1 \\ 7 & -10 & -2 \\ 1 & 2 & 4 \end{bmatrix}$$

```
> V := <5, -8, 13>;
```

$$V := \begin{bmatrix} 5 \\ -8 \\ 13 \end{bmatrix}$$

```
> x := LinearSolve(M, V);
```

$$x := \begin{bmatrix} \frac{8}{11} \\ \frac{17}{22} \\ \frac{59}{22} \end{bmatrix}$$

## 2.6. Робота з диференціальними рівняннями

Диференціальні рівняння покладені в основу математичних моделей різноманітних фізичних систем, процесів та пристроїв. Система Maple має потужні засоби для розв'язування диференціальних рівнянь і роботи з ними та їх розв'язками. При цьому система дозволяє проводити розв'язування як у символьному, так і в числовому вигляді, для окремих рівнянь та

їх систем, для звичайних рівнянь (ordinary differential equations ODE) та рівнянь у частинних похідних (partial differential equations PDE). Крім того, система має спеціальні засоби та пакети команд для виконання специфічних задач, пов'язаних із диференціальними рівняннями.

### 2.6.1. Символьне розв'язування диференціальних рівнянь та їх систем. Перевірка розв'язків

Існують два головні засоби Maple для розв'язування диференціальних рівнянь – це команда **dsolve** та помічник **ODE Analyzer.** Використання цих засобів описане в п. 2.2.5. У цьому підрозділі згадаємо використання команди **dsolve** на прикладах.

**Приклад 2.25.** Дано ODE. Необхідно його розв'язати без та з урахуванням початкових умов, а також із використанням методу розвинення в ряд та перетворення Лапласа.

```
> de := diff(y(x), x$2) = 2*y(x) + 1;
```

$$de := \frac{d^2}{dx^2} y(x) = 2\,y(x) + 1$$

```
> dsolve(de);
```

$$y(x) = e^{\sqrt{2}\,x}\_C2 + e^{-\sqrt{2}\,x}\_C1 - \frac{1}{2}$$

```
> Incond := y(0) = 1, D(y)(0) = 2;
```

$$Incond := y(0) = 1,\ D(y)(0) = 2$$

```
> dsolve({de, Incond});
```

$$y(x) = e^{\sqrt{2}\,x}\left(\frac{1}{2}\sqrt{2} + \frac{3}{4}\right) + e^{-\sqrt{2}\,x}\left(\frac{3}{4} - \frac{1}{2}\sqrt{2}\right) - \frac{1}{2}$$

Часто за неможливості отримати розв'язок використовують опцію розвинення в ряд:

```
> dsolve({de, Incond}, y(x), series);
```

$$y(x) = 1 + 2\,x + \frac{3}{2}x^2 + \frac{2}{3}x^3 + \frac{1}{4}x^4 + \frac{1}{15}x^5 + O\left(x^6\right)$$

або опцію перетворення Лапласа[1)]:

```
> dsolve({de, Incond}, y(x), method = laplace);
```

$$y(x) = -\frac{1}{2} + \frac{3}{2}\cosh\left(\sqrt{2}\,x\right) + \sqrt{2}\,\sinh\left(\sqrt{2}\,x\right)$$

Перевірити результати розв'язання можна за допомогою команди **`odetest(solution, ODE, y(x))`**:

```
> solution := dsolve({de, Incond}, y(x));
```

$$solution := y(x) = e^{\sqrt{2}\,x}\left(\frac{1}{2}\sqrt{2} + \frac{3}{4}\right) + e^{-\sqrt{2}\,x}\left(\frac{3}{4} - \frac{1}{2}\sqrt{2}\right) - \frac{1}{2}$$

```
> odetest(solution, de, y(x));
```

$$0$$

Якщо команда **`odetest`** повертає значення 0, то це означає, що розв'язок правильний.

Для розв'язування системи диференціальних рівнянь до команди **`dsolve`** необхідно включити систему рівнянь у фігурних дужках та необов'язково набір шуканих функцій також у фігурних дужках. Команда буде мати такий вигляд: **`dsolve({desystem, Incond}, {x(t), y(t)})`**.

**Приклад 2.26.** Розв'язати систему двох лінійних ODE з урахуванням початкових умов $x(1) = 1$, $y(0) = 0$:

$$\begin{cases} \dfrac{dx(t)}{dt} = y(t) \\ \dfrac{dy(t)}{dt} = -2x(t) \end{cases}$$

[1)] Перетворення Лапласа – це інтегральне перетворення, яке пов'язує функцію $F(s)$ комплексної змінної (зображення) з функцією $f(x)$ дійсної змінної (оригінал). Його переваги полягають у тому, що багатьом співвідношенням та операціям над оригіналами відповідають простіші співвідношення над їх зображенням. Наприклад, лінійні диференціальні рівняння стають алгебраїчними.

$$> \ desystem := \frac{d}{dt}x(t) = y(t),\ \frac{d}{dt}y(t) = -2\cdot x(t);$$

$$desystem := \frac{d}{dt}x(t) = y(t),\ \frac{d}{dt}y(t) = -2\,x(t)$$

$$> \ Incond := x(1) = 1,\ y(0) = 0$$

$$Incond := x(1) = 1,\ y(0) = 0$$

$$> \ dsolve(\{desystem, Incond\});$$

$$\left\{x(t) = \frac{\cos\left(\sqrt{2}\,t\right)}{\cos\left(\sqrt{2}\right)},\ y(t) = -\frac{\sqrt{2}\,\sin\left(\sqrt{2}\,t\right)}{\cos\left(\sqrt{2}\right)}\right\}$$

### 2.6.2. Числове розв'язування диференціальних рівнянь та їх систем

Загалом нелінійні диференціальні рівняння не мають розв'язку в аналітичному вигляді. Тоді їх потрібно розв'язувати числовими методами. Вони зручні і в тому разі, коли розв'язок потрібно подати числами або побудувати його графік.

Числове розв'язування диференціальних рівнянь відбувається також за допомогою команди **`dsolve`**, але із опцією **`numeric`**. Загальний вигляд команди при цьому буде таким: **`dsolve(ode_system, numeric, method=rkf45, vars, options)`**, де **`ode_system`** – диференціальне рівняння або система рівнянь; **`method=rkf45`** – опція визначення числового методу розв'язання, який буде використовувати команда **`dsolve`**; **`options`** – опції обраного числового методу.

Серед доступних числових методів основні такі:

– **classical** (одна з восьми версій класичного методу, що використовується за замовчуванням);

– **rkf45** (метод Рунге–Кутта–Фельберга 4–5-го порядків);

– **dverk78** (неперервний метод Рунге–Кутта порядків 7–8);

– **taylorseries** (метод розвинення в ряд Тейлора).

Для кожного методу існують свої опції **`options`**. Наприклад, для методу **rkf45** опції такі:

– **maxfun** (задає максимальну кількість обчислень правої частини рівняння, за замовчуванням дорівнює 30 000, для відключення цього ліміту записати **maxfun=0**);
– **abserr** (задає бажану абсолютну похибку розв'язку);
– **relerr** (задає бажану відносну похибку розв'язку);
– **initstep** (задає початковий крок);
– **range** (задає область значень незалежної змінної, для якої потрібно шукати розв'язок);
– **output** (задає тип результату);
– **stepsize** (задає статичний крок).

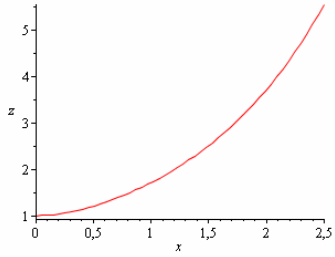
**Приклад 2.27.** Розв'язування системи двох ODE числовим методом.

```
> sys := diff(y(x),x) = 2*z(x) - y(x) - x, diff(z(x),x) = y(x) :
> functions := y(x), z(x) :
  Incond := y(0) = 0, z(0) = 1 :
> F := dsolve( {sys, Incond}, {functions}, numeric );
                        F := proc(x_rkf45) ... end proc
> F(2);
  [x = 2., y(x) = 2.94775576123021744, z(x) = 3.72064997248696061]
```

Зверніть увагу, що при числовому розв'язуванні команда **dsolve** створює процедуру (в даному прикладі з ім'ям *F*), яка обчислює значення шуканих функцій для заданого значення аргументу.

Нижче показано, як побудувати графік отриманих розв'язків на прикладі розв'язування *z(x)* за допомогою функції **odeplot** з пакета команд **plots**.

```
> plots[odeplot](F, [x, z(x)], 0..2.5, labels = [x, z]);
```




З прикладу можна бачити, що створена процедура *F* видає результат списком, елементами якого є значення змінної та значення шуканих функцій у вигляді рівностей. Існує особлива опція **output=listprocedure** команди **dsolve**, при якій буде

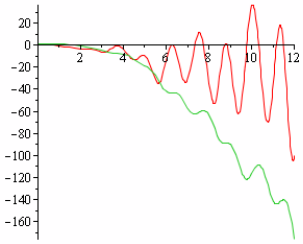
створено не одну процедуру для всіх функцій одразу, а список процедур для обчислення кожної функції окремо.

**Приклад 2.28.** Розглянути розв'язання системи двох диференціальних рівнянь із використанням опції **output=listprocedure**:

```
> sys := D(y)(x) = 2*z(x)*sin(5*x) - y(x)*cos(2*x) - x,
  D(z)(x) = y(x) :
> functions := y(x), z(x) :
  Incond := y(0) = 0, z(0) = 1 :
> F := dsolve({sys, Incond}, {functions}, numeric, output
    = listprocedure);
      F = [x = proc(x) ... end proc, y(x) = proc(x) ... end proc, z(x) = proc(x) ... end proc]
> Y := subs(F, y(x)) :
> Z := subs(F, z(x)) :
```

Тепер можна звертатися до кожного результату окремо. Функція **subs** тут перетворює список кінцевих даних на вектори розв'язку *Y* та *Z*. Так, побудуємо графіки двох знайдених функцій *y(x)* та *z(x)*:

```
> plot({Y, Z}, 0..12);
```




### 2.6.3. Інструментальний пакет розв'язування диференціальних рівнянь DEtools

Пакет **DEtools** містить специфічні засоби для аналітичного та числового розв'язування диференціальних рівнянь та їх систем, візуалізації розв'язку різними способами.

Звертатися до команд пакета можна двома стандартними способами:

```
> DEtools[command](arguments);
> command(arguments).
```

Розглянемо найбільш важливі функції із цього пакета.

– **autonomous** (тестує диференціальне рівняння на предмет автономності, тобто випадку, коли незалежна змінна до рівняння не входить в явному вигляді);
– **convertsys** (перетворює систему диференціальних рівнянь в систему першого порядку);
– **reduceOrder** (забезпечує зниження порядку диференціального рівняння);
– **regularsp** (знаходить особливі точки неавтономного лінійного диференціального рівняння першого порядку);
– **varparam** (розв'язує диференціальне рівняння або систему методом варіації параметрів);
– **de2diffop** (перетворює диференціальне рівняння на диференціальний оператор);
– **diffop2de** (перетворює диференціальний оператор на диференціальне рівняння);
– **DEplot** (будує 2D-розв'язок рівняння або системи);
– **DEplot3d** (будує 3D-розв'язок системи рівнянь);
– **dfieldplot** (будує поле напрямків);
– **PDEplot** (будує розв'язок диференціального рівняння в частинних похідних першого порядку);
– **phaseportrait** (будує фазовий портрет системи диференціальних рівнянь).

Функції з цього пакета для візуалізації розв'язків розглянемо в наступному пункті.

### 2.6.4. Графічна візуалізація розв'язків диференціальних рівнянь

*2.6.4.1. Функція* ***plots[odeplot]***

Для звичайного графічного 2D- або 3D-подання результатів розв'язання диференціальних рівнянь використовується функція **odeplot** з пакета **plots**. Її синтаксис є такий: **odeplot(s, vars, r, o)**, де s – результат роботи команди **dsolve(numeric)**; vars – змінні; r – параметр, що задає границі розв'язку, наприклад a..b; o – необов'язкові додаткові опції.

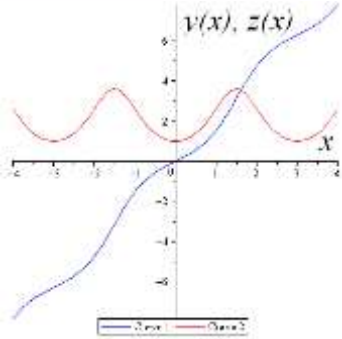
Звичайний розв'язок, як правило, більш наглядний, ніж фазові портрети. Але для спеціалістів (наприклад, у теорії коливань) фазовий портрет дає більше інформації. Він більш трудомісткий для побудови, але система Maple може будувати і фазові портрети.

**Приклад 2.29.** Побудувати графіки функцій $y(x)$ та $z(x)$, які є розв'язками системи диференціальних рівнянь, в звичайному вигляді та у вигляді фазового портрета:

```
> with(plots):
> sys := diff(y(x),x) = z(x), diff(z(x),x) = 3*sin(y(x)):
> functions := {y(x), z(x)}:
  Incond := y(0) = 0, z(0) = 1:
> p := dsolve({sys, Incond}, functions, numeric):
```

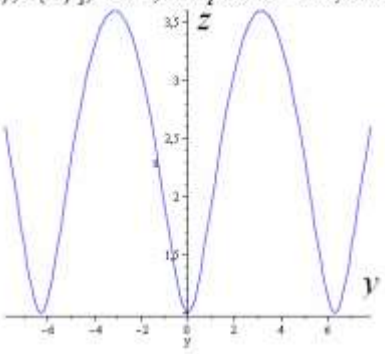
Звичайний графік:

```
> odeplot(p, [[x, y(x)], [x, z(x)]],
  -4..4, numpoints = 100, color = [blue, red]);
```




Фазовий портрет:

```
> odeplot(p, [y(x), z(x)], -4..4, numpoints = 100, color = [blue, red]);
```



*2.6.4.2. Функція* ***DEtools[DEplot]***

Розглянемо роботу команди **DEplot** із пакета **DEtools**. Вона числово розв'язує диференціальні рівняння та їх системи при одній незалежній змінній та будує графік розв'язку. При цьому використовується метод Рунге–Кутта 4-го порядку, а графічні побудови являють собою або криві, або векторні поля напрямків. Синтаксис функції такий: **DEplot (sde, vars, trange, inits, xrange, yrange, options)**, де sde – диференціальне рівняння або їх система у фігурних або квадратних дужках; vars – залежні змінні у фігурних або квадратних дужках; trange – область зміни незалежної змінної; inits – початкові умови для розв'язання; xrange та yrange – необов'язкові параметри, що задають область зміни для першої та другої залежної змінної; options – опції у вигляді keyword=value («ключове слово = значення»). Опції, які можуть використовуватися з функцією **DEplot**, наведено в таблиці 2.5.

Таблиця 2.5 – Опції функції **DEplot** із пакета **DEtools** у форматі **keyword**=value («ключове слово = значення»)

| Ключове слово | Значення | Зміст опції |
|---|---|---|
| *1* | *2* | *3* |
| **animatecurves** | true або false | Створює анімацію фазової траєкторії в часі |
| **animatefield** | true або false | Створює анімацію поля напрямків |
| **arrows** | 'small', 'smalltwo', 'medium', 'mediumfill', 'large', 'curve', 'comet', 'line', або 'none' | Задає тип стрілки векторного поля |
| **color** | name, RGB або HUE | Задає колір стрілок одним із 5 способів: ім'я кольору (name), цифрове позначення за шкалою RGB або HUE, математичним виразом або процедурою (про колір див. п. 4.1.2) |

Продовження таблиці 2.5

| *1* | *2* | *3* |
|---|---|---|
| **dirfield** | [int,int],int або [ [int,int],[ ],…] | Задає координати точок, куди поміщати стрілки (сіткою або кожну стрілку окремо) |
| **iterations** | int | Представляє метод для зменшення кроку stepsize при фіксованій кількості точок (int – натуральне число) |
| **linecolor** | name | Задає колір лінії |
| **numframes** | int | Вводить кількість кадрів при анімації |
| **numpoints** | int | Задає кількість точок, якими будується графік |
| **numsteps** | int | Задає кількість кроків при обчисленнях (використовується далі опцією stepsize) |
| **obsrange** | true або false | Задає, чи прибирати стрілки, що виходять за межі побудови (наприклад, при асимптотичній поведінці в анімації) |
| **scene** | [name, name] | Вказує імена залежних змінних, для яких будується графік |
| **size** | magnitude або float | Задає розмір стрілок, який визначається або пропорційно величині поля, або заданим числом типу float (за замовчуванням size = 1.0) |
| **stepsize** | real | Задає розмір кроку для числового методу обчислення розв'язку рівняння (за замовчуванням $stepsize = {}^{a-b}\!/_{numsteps}$ при trange=a..b) |

**Приклад 2.30.** Побудувати графіки розв'язку рівнянь моделі Лоткі – Вольтерра у вигляді векторного поля напрямків та фазового портрета.

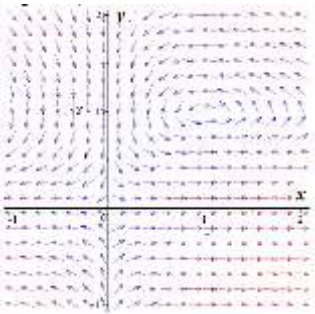
Нехай рівняння моделі мають вигляд:

$$\frac{dx}{dt} = x(t)(1 - y(t)),$$

$$\frac{dy}{dt} = 0,3y(t)(x(t) - 1).$$

Побудуємо векторне поле:

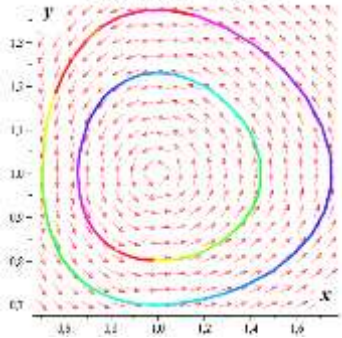
```
> with(DEtools):
> sys := [ d/dt x(t) = x(t)·(1 − y(t)), d/dt y(t) = 0.3·y(t)·(x(t) − 1) ]:
> DEplot(sys, [x(t), y(t)], t = −2..2, x = −1..2, y = −1..2, arrows = medium, color
    = magnitude);
```



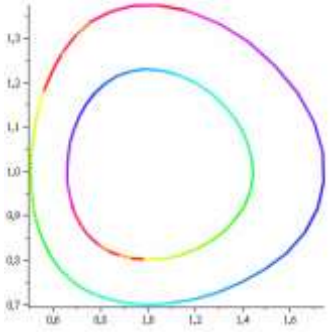
Побудуємо фазовий портрет. Для цього функції DEplot буде необхідно «знати» початкові координати фазових траєкторій:

```
> DEplot( sys, [x(t), y(t)], t = −7..7, [[x(0) = 1.2, y(0) = 1.2], [x(0) = 1, y(0)
    = .7]], arrows = medium, linecolor = t/10 );
```

При arrows = medium



При arrows = none



Ці два графіки на вигляд відрізняються значенням опції arrows. Так, зліва виводиться одночасно і поле напрямків, і фазові траєкторії, справа – лише фазові траєкторії. Якщо задати опцію animatecurves = true, то побачимо

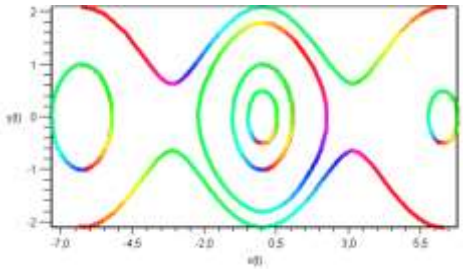
«мультик» про те, як малюються фазові траєкторії, починаючи від їх початкових координат.

### *2.6.4.3. Функція* `DEtools[DEplot3d]`

Ця функція будує розв'язок системи диференціальних рівнянь у вигляді просторових кривих, наприклад ліній однакового рівня або просто кривих у просторі. Поле напрямків ця функція не будує. При цьому повинна бути лише одна незалежна змінна.

**Приклад 2.31.** Розглянемо коливання маятника, які описуються координатою $x(t)$ та швидкістю $y(t)$ залежно від часу $t$.

```
> with(DEtools):
> DEplot3d({diff(x(t),t)=y(t), diff(y(t),t)=-sin(x(t))}, [x(t),y(t)], t=0..10, [[x(0)=0, y(0)=0.5], [x(0)=0, y(0)=1], [x(0)=0, y(0)=1.8], [x(0)=-2*Pi, y(0)=1], [x(0)=2*Pi, y(0)=0.5], [x(0)=-2*Pi, y(0)=2.1], [x(0)=2*Pi, y(0)=-2.1]], stepsize=0.2, orientation=[0,90], linecolor=sin(t)-t);
```



У даному разі маємо тривимірний графік, який можна розглядати під різними кутами, для чого використовуємо опцію orientation, наприклад orientation=[0,90], або інструмент інтерактивного повертання графіка  .

### *2.6.4.4. Інші функції графіки пакета* **DEtools**

Серед інших функцій для побудови розв'язків диференціальних рівнянь розглянемо `dfieldplot` та `phaseportrait`.

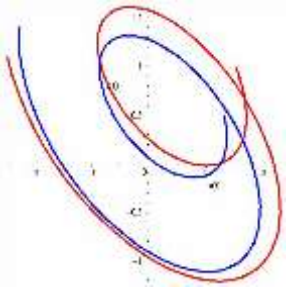
Команда `dfieldplot` використовується для побудови векторного поля напрямків за результатами розв'язку системи двох автономних диференціальних рівнянь першого порядку або одного диференціального рівняння першого порядку. При цьому вони повинні мати лише одну незалежну змінну.

Команда `phaseportrait` використовується для побудови фазового портрета системи диференціальних рівнянь першого

порядку або одного диференціального рівняння вищого порядку. Ця команда використовує числові методи розв'язання, тому обов'язково необхідно зазначати початкові умови. Для рівнянь першого порядку команда може побудувати й поле напрямків.

**Приклад 2.31.** Побудова фазового портрета системи трьох неавтономних диференціальних рівнянь першого порядку:

```
> sde := D(x)(t) = y(t) - z(t),
         D(y)(t) = z(t) - x(t),
         D(z)(t) = x(t) - y(t)·2 :
  vars := x(t), y(t), z(t) :
  inits := [x(0) = 1, y(0) = 0, z(0) = 1.5],
           [x(0) = 0.5, y(0) = 0, z(0) = 1.3] :
  phaseportrait( [sde], [vars], t = 0..5, [inits],
                 stepsize = 0.05, scene = [z(t), x(t)],
                 linecolour = [red, blue],
                 method = classical[foreuler])
```



Як бачимо, кількість рівнянь та залежних змінних може бути більшою, ніж два. Тоді для виведення фазового портрета у 2D-варіанті використовується опція scene у вигляді scene=[x(t),z(t)].

### 2.6.5. Розв'язування диференціальних рівнянь у частинних похідних

Для простого розв'язування диференціальних рівнянь у частинних похідних та їх систем служить команда **`pdsolve`**. Для отримання символьного результату використовують синтаксис команди **`psdolve(pde, function, options)`**, а для числового – **`psdolve(pde, function, numeric, options)`**, в яких pde – рівняння або система; function – ім'я залежної змінної (змінних), які підлягають визначенню; options – додаткові параметри побудови; numeric – опція, яка визначає саме числове розв'язування.

Альтернативно можна використовувати засоби спеціалізованого пакета **PDEtools**, який містить команди поглибленої роботи з диференціальними рівняннями у частинних похідних, а саме: різні перетворення, підстановки, види представлення розв'язків.

# 3. Елементи програмування

## 3.1. Засоби програмування

У цьому розділі розглядаються такі засоби програмування на мові Maple, як умовні вирази, цикли, процедури та ін. Взагалі всі інтерактивні засоби обчислень у Maple можна вважати частиною мови Maple, але в цьому розділі увага приділяється саме типовим засобам створення програм. На відміну від звичайних мов програмування написання коду на мові Maple є набагато простішим в основному завдяки інтерактивності та використанню вже готових функцій та шаблонів задач.

Розробники системи рекомендують під час програмування працювати в режимі Worksheet та 1D-Math input.

### 3.1.1. Умовні вирази

Можна запрограмувати систему на виконання певних дій лише тоді, коли задовольняється певна умова (набір умов). Це реалізується за допомогою умовних конструкцій **if**.

Є три способи задання умов:

1) **if** <умова 1> **then** <елементи 1> **end if**;
2) **if** <умова 1> **then** <элементи 1>
   **elif** <умова 2> then <элементи 2>
   **elif** <умова 3> then <элементи 3>
   **else** < елементи 4>
   **end if**;
3) **`if`** (<умова>, <елемент_правда>,
   <елемент_хибність>) **end if**.

У першому варіанті дії <елементи 1> будуть виконуватися, лише якщо задовольняється <умова 1>.

У другому варіанті умови розгалужуються за допомогою **elif** – «інакше якщо». Якщо жодна з <умов 1,2,3> не задовольняється, будуть виконуватись <елементи 4>.

Третій варіант є лаконічним записом, де <елемент_правда> виконуватиметься за правильної <умови>, а <елемент_ хибність> – за неправильної <умові>.

Умови можна задавати використовуючи:

– оператори порівняння <, <=, =, >=, >, <>;
– логічні оператори and, or, not;
– логічні імена true, false, FAIL.

**Приклад 3.1.** Розглянемо три варіанти запису умовних виразів:

```
> a := 1:
  b := 2:
  if b < a then a else b end if;
                                  2
```

```
> a := 3:
> if a = 1 then print(first)
    elif a = 2 then print(second)
    elif a = 3 then print(third)
  end if
                                  third
```

```
> a := 1:
  b := 2:
  3·(4 + `if`(b < a, a, b));
                                  18
```

### 3.1.2. Конструкції циклу

Цикли потрібні, якщо необхідно організувати повторюване виконання певної послідовності команд. Виконувати ці повтори можна трьома шляхами (відповідно є три типи циклів):

1) поки змінна-лічильник не перевищує заданого ліміту (конструкція **for/from**);

2) для кожного операнда заданого виразу (конструкція **for/in**);

3) поки виконується логічна умова (конструкція **while**).

Цикл **for/from** має такий синтаксис:

**> for** <лічильник> **from** <початкове значення> **by** <крок> **to** <кінцеве значення> **do** <елементи> **end do**.

У цій конструкції <елементи> будуть виконуватися для кожного послідовного значення змінної-лічильника, яке задане початковим, кінцевим значеннями та кроком його зміни. За замовчуванням <початкове значення> = 1, <крок> = 1, <кінцеве значення> = ∞.

Цикл **for/in** має такий синтаксис:

**> for** <змінна> **in** <вираз> **do** <елементи> **end do**.

У даному варіанті <елементи> будуть виконуватися при тих значеннях <змінної>, які безпосередньо визначаються <виразом>.

Цикл **while** має такий синтаксис:

> **while** <умовний вираз> **do** <елементи> **end do**.

У даному разі <елементи> виконуються доти, поки виконується умова в <умовному виразі>.

Можна також комбінувати конструкції циклів між собою певним чином: **> for/from/while** та **> for/in/while**.

**Приклад 3.2.** Порівняємо роботу різних конструкцій циклів.

```
> for n from 1 by 1 to 5 do
  evalf(sqrt(n));
  end do;
                    1.
                1.414213562
                1.732050808
                    2.
                2.236067977
```

Тут лічильник *n* по черзі набуває значень 1, 2, 3, 4, 5, та одночасно це значення використовується для обчислення квадратного кореня на кожному кроці циклу.

```
> L := [23.4, 87.2, 43.0, 99.7]:
  for i in L do
  evalf(sin(i * Pi / 180));
  end do;
                0.3971478907
                0.9988061374
                0.6819983602
                0.9857034690
```

Це є приклад циклу for/in, де список *L* задає всі значення, яких по черзі набуває лічильник *i*. Відповідно ці значення лічильника *i* також використовуються всередині циклу для обчислень.

```
> x := 0;
  for i from 11 by 2 while i < 100
  do x := x + i end do;
```

```
x := 0
x := 11
x := 24
x := 2279
x := 2376
x := 2475
```

Цей цикл є мішаним, типу for/from + while. Тобто лічильник *i* послідовно набуває значень 11, 13, 15, 17, …, 99, і, як тільки *i* = 101, виконання обчислень припиняється.

### 3.1.3. Оператори пропуску та переривання

Часто є необхідним пропустити певний крок циклу. Для цього використовують оператор **next**.

Якщо ж потрібно повністю перервати виконання циклу або іншого елемента програми, наприклад процедури, використовують оператор **break**.

**Приклад 3.3.** Порівняємо роботу операторів next та break.

```
> for i in [1, 2, 3, -4, 5] do
    if i = -4 then next
    else print(i) end if end do;
```

```
1
2
3
5
```

```
> for i in [1, 2, 3, -4, 5] do
    if i = -4 then break
    else print(i) end if end do;
```

```
1
2
3
```

Як видно, для одних і тих самих умов циклу оператор next лише пропускає крок, коли лічильник *i* = -4, а цикл далі виконується для наступного *i*. У разі оператора break цикл повністю переривається та для наступних *i* надалі не виконується.

### 3.1.4. Процедури

***Процедури*** – це модулі програми, які мають самостійне значення. Вони дозволяють виконувати цілий ряд команд усього лише одним звертанням до процедури через її ім'я. Процедури використовують, наприклад, якщо необхідно часто виконувати один набір дій.

Загальна форма задання процедури є такою:

```
>name := proc (var1, var2, …) local <vars>; global <vars>;
        options <options>; description <string>;
        expressions;
        end proc .
```

Розберемо детально структуру процедури:

– *name* – це є ім'я процедури, за яким до неї звертаються в подальшому;

– початок і кінець процедури зазначаються як **proc()** та **end proc**;

– **proc()** або **proc (var$_1$,var$_2$,…)** є оголошенням процедури, в круглих дужках за необхідності задають список формальних параметрів процедури;

– **local** <vars> вводить локальні параметри (необов'язково);

– **global** <vars> вводить глобальні параметри (необов'язково);

– **options** <options> є заданням опцій, які детально розглянуті нижче (необов'язково);

– **description** <string> є оголошенням коментарів, на цьому закінчується заголовок процедури (необов'язково);

– *expressions* є тілом процедури, тобто послідовністю команд, що реалізують зміст процедури.

**Приклад 3.4.** Розглянемо нескладну процедуру, яка обчислює модуль комплексного числа.

Задамо процедуру з ім'ям *modc* таким чином:

```
> modc := proc(z) evalf(sqrt(Re(z)^2 + Im(z)^2)) end proc;
      modc := proc(z) evalf(sqrt(Re(z)^2 + Im(z)^2)) end proc
```

Тут *z* – це комплексне число, модуль якого обчислюється, і воно є формальним параметром процедури. Значення числа *z* задається ззовні процедури. Так, наприклад, щоб обчислити модуль для *z = 3 + 4i*:

```
> modc(3. + 4. I);
                              5.000000000
```

*Рекомендується* при оголошенні процедури розміщувати її структурні елементи на робочій сторінці візуально зручно. Так, для даної процедури:

```
> modc := proc(z)
          evalf(sqrt(Re(z)^2 + Im(z)^2))
          end proc:
```

Якщо тіло процедури містить не одну, а декілька команд, то, як правило, процедура повертає значення останнього виконаного оператора. У будь-якому разі виконання процедури можна перервати оператором **Return(value)**, де value – значення, яке поверне процедура. Value також може бути пустим, тоді використовують оператор запис **Return()**.

Аналогією процедури, що складається лише з однієї дії, є форма задання функції через функціональний оператор →.

У загальному випадку необхідним при заданні процедури є лише послідовність операторів, інші параметри можуть бути відсутніми. Однак часто задання додаткових параметрів може бути потрібним.

***Локальними параметрами*** називаються змінні, що використовуються всередині процедури. Якщо їх імена збігаються з іменами зовнішніх змінних, то конфлікту між ними не виникає. ***Глобальними параметрами*** називаються ті параметри, які використовуються і процедурою, і зовнішньою програмою. Рекомендується їх оголошувати в заголовку процедури.

Для раніше створеної процедури можна вивести на екран її вміст за допомогою таких команд:

```
> interface('verboseproc'=2):
> print(name_of_proc);
```

Далі розглянемо, які саме ***опції***, або розширювальні ключі, використовують у процедурах:

1) **`remember`** – створює в пам'яті таблицю значень процедури, для того щоб при звертанні до неї не виконувати обчислення заново, а брати вже готові значення з таблиці. Ця опція прискорює виконання процедури.

```
> f := proc(x) options remember; x^3; end proc:
> f(2); f(3);
                                8
                                27
```

2) **`system`** – як правило, використовується разом із **`remember`** і надає процедурі статусу системної, для якої може бути видалена таблиця значень. Видалення відбувається під час процедури «збирання сміття», тобто видалення всіх даних, на які немає жодного посилання, за допомогою команди **`gc()`** – garbage collection (англ.).

3) **`builtin`** – використовується для інформування користувача про те, що процедура є вбудованою в ядро системи Maple. Створити власну вбудовану процедуру неможливо.

```
> print(map);
                    proc( ) option builtin = map; end proc
```

4) **operator, arrow** – надає процедурі статусу функціонального оператора, що може спрощувати поводження з нею. Так, двома еквівалентними записами будуть такі, наприклад:

```
> f := proc(x) option operator, x^2 - 1; end proc;
                    f := proc(x) option operator, x^2 - 1 end proc
> f := x -> x^2 - 1;
                                f := x -> x^2 - 1
```

5) **trace** – задає виведення налагоджувальної інформації, коли буде видаватися поетапне виконання процедури.

6) **Copyright** – забороняє відображати оператори тіла процедури. Для відміни захисту потрібно виконати команду **interface('verboseproc'=2)**.

Після опцій у заголовку можна задавати коментар у вигляді рядка **description** '…' для опису процедури. На виконання процедури коментар не впливає.

### 3.1.5. Модулі

Такі об'єкти Maple, як процедури, пов'язують послідовність декількох команд з однією командою, і процедури повертають одне значення. Можна побудувати більш складні програмні структури – ***модулі***. Вони дозволяють зв'язати дані з процедурами, які ці дані оброблюють. Особливістю модулів є те, що вони можуть експортувати значення змінних. Тобто змінні, які були створені всередині модуля, будуть доступні і ззовні. Наприклад, більшість пакетів команд Maple були виконані як модулі, і кожна команда може бути використана поза модулем.

Детально ознайомитися з модулями можна на сторінці довідкової системи Maple, виконавши команду **?module**.

## 3.2. Генерація випадкових чисел

Для генерації випадкових чисел і роботи з ними існує декілька різних засобів – окремі функції та спеціалізований

пакет команд **RandomTools**. Розглянемо функції генерації випадкових чисел.

### 3.2.1. Функції rand, randomize. Псевдовипадкові числа

Функція `rand()` без аргументів генерує випадкове ціле невід'ємне число з 12 знаків, наприклад 395718860534.

Функція `rand(a..b)` із заданням числового діапазону служить для генерації випадкового цілого числа з цього діапазону. При цьому спочатку генерується процедура, а потім при викликанні цієї процедури генерується число.

```
> numb := rand(1..10);
numb := proc( )
  proc( )
    option builtin = RandNumberInterface;

  end proc(6, 10, 4) + 1
end proc
> numb( );
                    7
```

Функція `rand(n)` з одним аргументом аналогічна функції `rand(0..n-1)`.

**Приклад 3.5.** Згенерувати випадкове дробове число від 0 до 1 за допомогою функції `rand`.

```
> evalf( rand( ) / 10^12, 3 );
                    0.396
```

Насправді всі генератори випадкових чисел виробляють ***псевдовипадкові*** числа. Річ у тому, що в системі Maple є вбудована послідовність чисел, з якої і беруться ті числа, що генеруються немов би випадково. Тому всі команди генерації, що використовуються, будуть впливати одна на іншу. Так, якщо виконати команду rand() після restart, то будемо мати одне й те саме випадкове число:

```
> restart : rand( );
    395718860534                    (1)
> restart : rand( );
    395718860534                    (2)
```

Для того щоб змінити цю послідовність, необхідно глобальній змінній **seed** (початкове число послідовності) призначити будь-яке ціле додатне число, або для цього використати функцію `randomize`:

– `randomize(n)` установлює значення змінної **seed** таким, що дорівнює n;

– `randomize()` встановлює значення змінної **seed** на основі системного годинника.

### 3.2.2. Генерація із заданим розподілом

Для генерації випадкових чисел із заданим розподілом використовується підпакет **random** із пакета **stats**. Загальний синтаксис команди такий:

```
> with(stats):
  random[distribution](quantity, uniform, method); або
> stats[random, distribution](quantity, uniform, method);
```

Розшифруємо елементи, що входять до цієї команди:

– `distribution` – вигляд розподілу (див. таблицю 3.1);

– `quantity` – ціле додатне число, яке дорівнює кількості чисел, що генеруються (за замовчуванням =1);

– `uniform` – генерація рівномірно розподілених чисел одночасно (за замовчуванням ‘default’);

– `method` – один із таких методів генерації: ‘auto’ (за замовчуванням), ‘inverse’, ‘builtin’.

Таблиця 3.1 – Види розподілів `distribution`

| `distribution` | Вид розподілу |
|---|---|
| **binomiald** | Дискретний біноміальний розподіл |
| **discreteuniform** | Дискретний рівномірний розподіл |
| **empirica** | Дискретний емпіричний розподіл |
| **poisson** | Дискретний розподіл Пуассона |
| **beta** | Неперервний бета-розподіл |
| **cauchi** | Неперервний розподіл Коші |
| **exponential** | Експоненціальний розподіл |
| **normald** | Нормальний (гаусів) розподіл |
| **uniform** | Неперервний рівномірний розподіл |

### 3.2.3. Пакет RandomTools

Пакет **RandomTools** має розширений набір інструментів для генерації випадкових чисел та роботи з випадковими об'єктами.

Цей пакет містить декілька команд та підпакетів. Підпакети служать для реалізації різних псевдовипадкових генераторів, які мають різні властивості, тому користувач може сам обрати бажаний варіант. Докладно дізнатися про роботу команд та підпакетів можна в довідковій системі, виконавши команду `>?RandomTools`.

Функція цього пакета `RandomTools[Generate](flavor)` генерує випадкові числа, кількість та властивості яких задані параметром `flavor`. Для того щоб мати уявлення про широкі можливості генерації чисел цієї функції, параметр `flavor` описаний у таблиці 3.2.

Таблиця 3.2 – Параметр `flavor` функції `Generate` пакета **RandomTools**

| Значення параметра | Опис |
|---|---|
| *1* | *2* |
| **choose(collection)** | Генерація елемента з колекції **collection**, якою можуть бути набір, список, матриця, вектор, масив або таблиця |
| **complex** | Генерація комплексного випадкового числа |
| **distribution** | Генерація із заданим розподілом |
| **exprseq** | Генерація послідовності випадкових об'єктів |
| **float** | Генерація чисел із плаваючою точкою |
| **integer** | Генерація цілих чисел |
| **list** | Генерація списку випадкових об'єктів |
| **listlist** | Генерація подвійного списку випадкових об'єктів (тобто списку списків) |
| **negative** | Генерація від'ємного раціонального числа |
| **negint** | Генерація від'ємного цілого числа |
| **nonnegint** | Генерація невід'ємного цілого числа |
| **nonposint** | Генерація непозитивного цілого числа |

Продовження таблиці 3.2

| *1* | *2* |
|---|---|
| **nonpositive** | Генерація непозитивного раціонального числа |
| **nonzero** | Генерація ненульового раціонального числа |
| **nonzeroint** | Генерація ненульового цілого числа |
| **polynom** | Генерація полінома з випадковими коефіцієнтами (за замовчуванням степеня 5) |
| **posint** | Генерація позитивного цілого числа |
| **positive** | Генерація позитивного раціонального числа |
| **rational** | Генерація раціонального числа |
| **set** | Генерація набору випадкових об’єктів |
| **truefalse** | Генерація одного з двох логічних значень: **true** або **false** (правда чи хибність) |

Кожен параметр може мати складну структуру відповідно до задач генерації випадкових об’єктів (див. довідкову систему `?RandomTools`).

**Приклад 3.6.** Розглянемо генерацію числа з плаваючою точкою, цілого числа, списку з трьох чисел із плаваючою точкою.

```
> restart:
  with(RandomTools):
> Generate(float);
                    0.001715876729
> Generate(integer);
                    -306860183579
> Generate(list(float, 3))
  [5.424170465 10^-8, 7.235196453 10^-9, 9.988963610 10^-9]
```

Окрім `Generate`, іншими командами пакета **RandomTools** є такі:

– `AddFlavor` – створення власного параметру генерації `flavor` для функції `Generate`;

– `GetFlavor` – повертає опис параметра `flavor` у вигляді процедури;

– `GetFlavors` – повертає імена всіх відомих параметрів `flavor`;

– **GetState** – повертає внутрішній стан генератору псевдовипадкових чисел, який використовується функцією **Generate**;

– **HasFlavor** – перевіряє, чи відомий такий параметр **flavor**;

– **RemoveFlavor** – видаляє даний вид параметра **flavor** зі списку, який розпізнається функцією **Generate**;

– **SetState** – установлює вид (стан) генератора псевдовипадкових чисел.

## 3.3. Створення та використання меплет-програм

### 3.3.1. Поняття про меплети

***Меплет-програми***, або ***меплети*** (***maplet*** – англ.), – це графічний інтерфейс користувача, який дає інтерактивний доступ до обчислювальних операцій системи Maple за допомогою кнопок (buttons), текстових областей (text regions), смуг прокручування (slider bars) та інших активних візуальних елементів.

Можна створювати власні меплети або використовувати вбудовані, які покривають багато академічних та спеціалізованих задач.

Меплети можна зберігати окремим файлом із розширенням *.maplet та запускати із середовища Windows. Також меплети викликаються з відкритих maple-документів.

Для того щоб редагувати код меплета, файл необхідно відкрити через головне меню системи Maple File→Open, обрати Maplet зі списку Files of type.

Для створення меплетів та роботи з ними є спеціалізований пакет команд **Maplets**. Він має три підпакети **Maplets[Tools]** **Maplets[Utilities]** та **Maplets[Elements]**.

### 3.3.2. Способи створення меплетів

*3.3.2.1. Командний спосіб*

Створити меплет можна використанням команд у робочому документі. Загалом цей спосіб містить три головні кроки:

1) підключити пакет **Maplets[Elements]**;

2) за допомогою команди цього пакета `Maplet` створити меплет;

3) за допомогою команди `Maplets[Display]` вивести меплет на екран.

**Приклад 3.7.** Створення найпростішого меплета.

```
> with(Maplets[Elements]):
> Mymaplet := Maplet([["My first Maplet"]]):
> Maplets[Display](Mymaplet):
```

Тут *Mymaplet* – ім'я меплета. Віконце створеного меплета буде виглядати так:


*3.3.2.2. Maplet-Builder*

Набагато зручніший спосіб створення меплетів – використання інтерактивного помічника – ***меплет-конструктора*** (рис. 3.1). Його запуск проводиться з головного меню **Tools → Assistants → Maplet Builder**.

У його вдалому графічному інтерфейсі можна за допомогою мишки та операцій «drag-and-drop» легко перетягувати потрібні елементи в робочу область, тобто компонувати віконце меплета, а також легко встановлювати необхідні властивості елементів, і таким чином визначати завдання, які будуть виконуватись елементами меплета. Для нескладних меплетів – це кращий спосіб створення.

На рис. 3.1 показаний вигляд вікна конструктора Maplet Builder. Воно ділиться на чотири області-панелі. Панель палітр елементів 1, або Palette Pane, містить палітри-набори

структурних елементів меплетів, відсортованих за категоріями. Панель компонування 2, або Layout Pane, є областю для компонування структури вікна меплета та розміщення в ньому елементів. Панель властивостей 3, або Properties Pane, служить для визначення властивостей кожного елемента. Панель команд 4, або Command Pane, відображає можливі команди та відповідні дії.

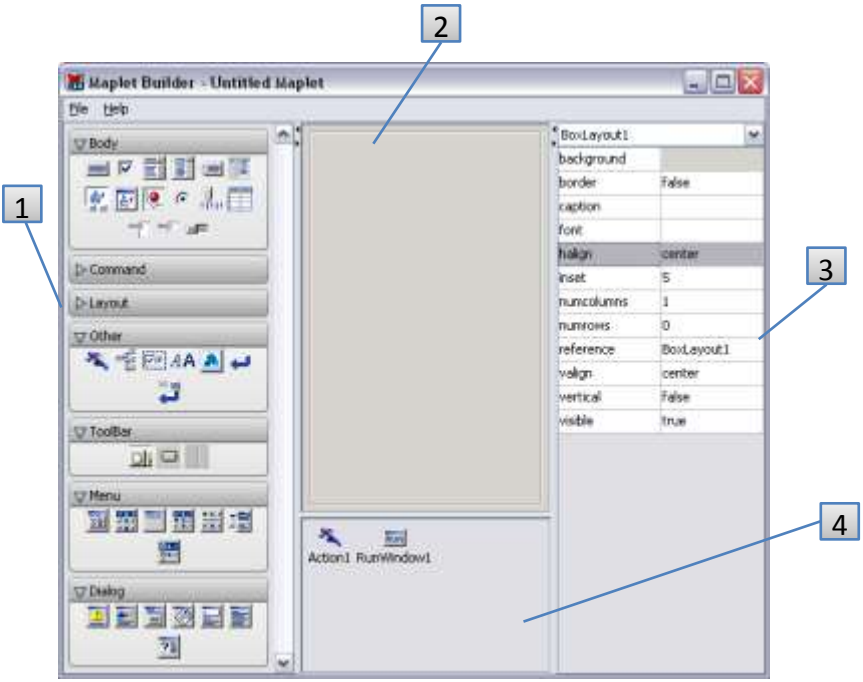



Рисунок 3.1 – Вигляд вікна Maplet Builder: 1 – панель палітр елементів (Palette Pane); 2 – панель компонування (Layout Pane); 3 – панель властивостей (Properties Pane); 4 – панель команд (Command Pane)

Розглянемо покроковий процес створення меплета, який розв'язує рівняння вигляду $f(x) = 0$ та будує графік функції $y = f(x)$ для порівняння числового та графічного розв'язків.

I. Спочатку компонуємо схему майбутнього вікна, тобто ділимо область вікна на декілька комірок, в яких потім розмістимо елементи:

а) на панелі властивостей у верхньому меню, що розкривається, обираємо BoxLayout1, змінюємо значення його властивості numcolumns на 2;

б) в меню, що розкривається, обираємо BoxColumn1, ставимо numrows = 4;

в) в меню, що розкривається, обираємо BoxColumn2, ставимо numrows = 2.

Зараз панель компонування розбита на комірки і має вигляд, як показано на рис. 3.2.

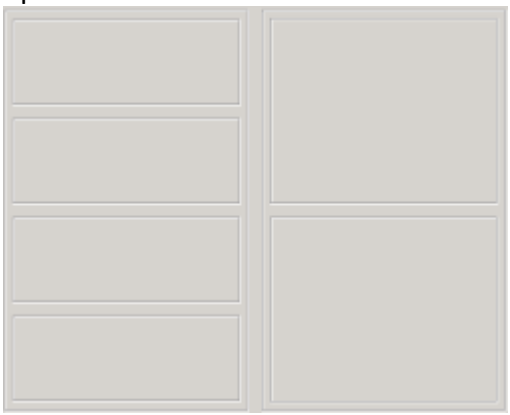


Рисунок 3.2 – Вигляд вікна-заготовки

II. Перетягуємо елементи з палітри шаблонів Body до лівого стовпчика:

а) Label (текстова мітка) до верхньої комірки;

б) TextField (текстове поле) до другої комірки;

в) Label до другої комірки, поруч із текстовим полем;

г) Button (кнопка) до третьої комірки;

ґ) Label до нижньої комірки;

д) TextField до нижньої комірки, поруч із текстовою міткою.

III. Перетягуємо елементи з палітри шаблонів Body до правого стовпчика:

а) Plotter (конструктор графіків) до верхньої комірки;

б) Button до нижньої комірки;

в) Button до нижньої комірки, поруч із попередньою кнопкою.

На цьому етапі схема вікна виглядає, як показано на рис. 3.3.

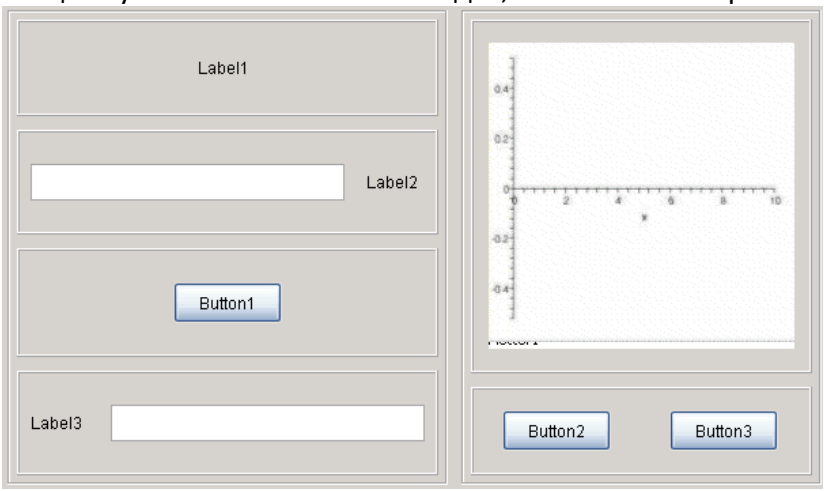



Рисунок 3.3 – Розміщення елементів меплета у вікні

IV. Змінюємо властивості доданих елементів на панелі властивостей. Для цього вибір відповідного елемента можна виконати або лівою кнопкою мишки безпосередньо на схемі вікна, або з меню, що розкривається, на панелі властивостей:

а) обираємо Label1. Змінюємо caption на "Задайте рівняння у вигляді $f(x) = 0$:";

б) обираємо Label2. Змінюємо caption на "= 0";

в) обираємо Button1. Змінюємо caption на "Розв'язати рівняння". Обираємо в onclick Evaluate. У новому віконці

"Evaluate Expression" змінюємо Target на TextField2, у полі Expression вводимо команду solve(TextField1,x), натискаємо Ok;

г) обираємо Label3. Змінюємо caption на "Корені:".

ґ) обираємо Button2. Змінюємо caption на "Побудувати графік $y = f(x)$". Обираємо в onclick Evaluate. У новому віконці "Evaluate Expression" переконуємося, що в меню Target вибрано Plotter1, у полі Expression вводимо команду plot(TextField1,x), натискаємо Ok.

д) обираємо Button3. Змінюємо caption на "Вихід". Обираємо в onclick "Close Window".

V. Запускаємо меплет через меню File → Run. При запиті «зберегти» зберігаємо в окремий файл *.maplet.

На рис. 3.4 зображено віконце запущеного меплета з розв'язаним рівнянням $\sin(x) + \cos(x) = 0$ та побудованим графіком $y = \sin(x) + \cos(x)$.

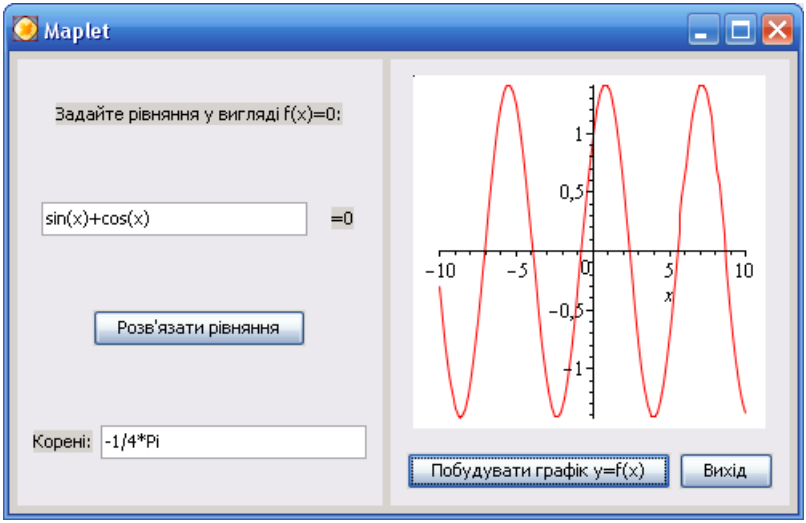



Рисунок 3.4 – Вигляд вікна створеного меплета

Розглянутий меплет можна реалізувати і командним способом, усім діям конструктора відповідають команди Maple. Але це є більш складний шлях і для початківців краще працювати саме з інтерактивним помічником-конструктором.

# 4. Графіка

## 4.1. Побудова графіків функцій у системі Maple

### 4.1.1. Чотири основні способи побудови графіків функцій

Для побудови різноманітних графіків функцій можна діяти чотирма шляхами:

1) використовувати команди, які можуть бути вбудованими в ядро системи або належати до спеціалізованих пакетів команд;

2) використовувати відповідні позиції контекстного меню, яке викликається правим кліком у робочому документі;

3) перетягувати функції та графіки від одних об'єктів до інших методом "drag-and-drop";

4) застосовувати інтерактивний помічник-конструктор **Plot Builder**.

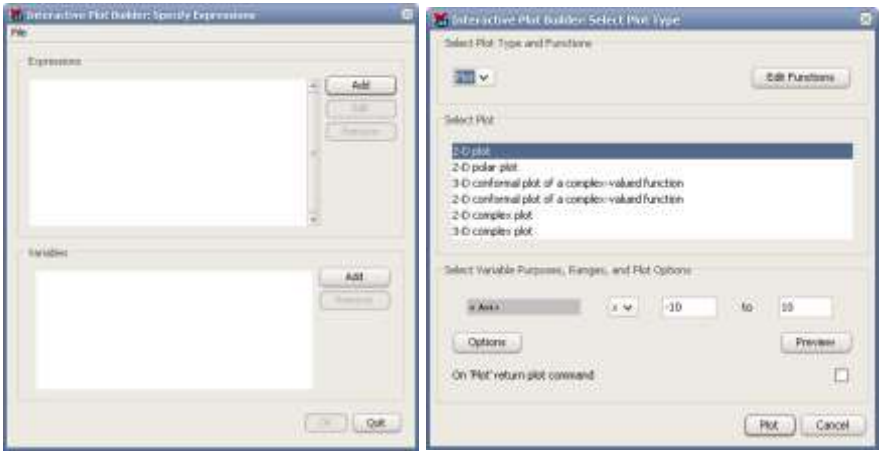
Кожен метод має свої переваги, і його вибір залежить від типу графіка та персональних уподобань.

**Помічник Plot Builder** знаходиться в головному меню **Tools → Assistants → Plot Builder**. Він являє собою зручний інтерактивний інструмент для побудови графіків, тип яких залежить від виразу функції, наприклад 2D- чи 3D-графік, векторне поле, графік густини, анімований графік та ін.

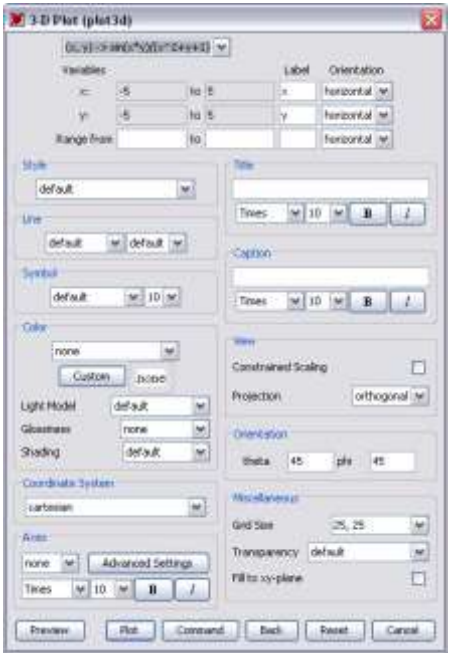
Під час використання помічника користувач задає побудову графіка, працюючи послідовно з трьома віконцями, які зображені на рис. 4.1.

У першому віконці (рис. 4.1 а) додають, редагують або видаляють вирази, графіки яких будуються, а також незалежні змінні.

У другому віконці (рис. 4.1 б) обирають тип графіка зі списку та визначають інтервали відображення по осях.



а б



в

Рисунок 4.1 – Три послідовні віконця помічника Plot Builder (а – крок перший; б – крок другий; в – крок третій)

У третьому віконці (рис. 4.1 в) задають усі основні опції, які розглядаються у вигляді команд окремо в табл. 4.1 та 4.2. Можна кнопкою Preview попередньо переглянути майбутній результат. Кнопка Plot створює кінцевий графік.

У цьому помічнику також передбачена корисна можливість наочного та швидкого оглядання графіків при зміні деяких параметрів в інтерактивному режимі.

**Контекстне меню** системи Maple відображає список команд, які можна використовувати для роботи з поточним виразом. Природно, що вигляд цього меню залежить від типу виразу. Так, можна швидко будувати графік виразу через виклик цього меню (див. рис. 4.2) правим кліком.

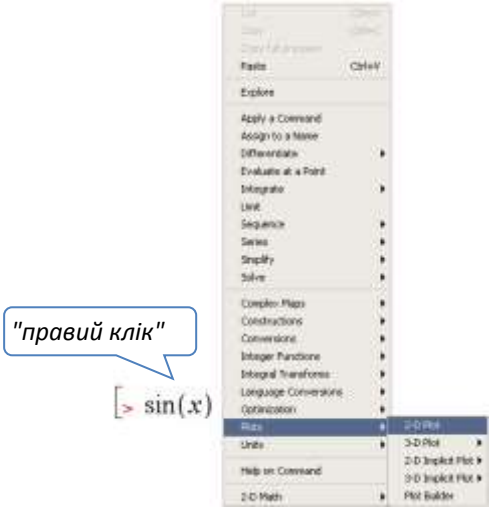



Рисунок 4.2 – Типовий вигляд контекстного меню та вибір команд побудови графіка

Використання методу **"drag-and-drop"** (перетягування мишкою) передбачає дуже простий алгоритм створення графіка:

1) створити пустий графік: Insert → Plot → 2D або 3D;

2) записати вираз, графік якого будується, в області введення [>;

3) перетягти вираз мишкою в область графіка (якщо необхідно скопіювати, то перетягти при одночасно натиснутій кнопці Ctrl).

Цей метод дозволяє перетягування виразів до пустих графіків та вже створених, перетягування декількох виразів до одного графіка, перетягування назад з області графіка до робочої області та розділення графіків.

Найбільш універсальним є саме **командний спосіб**, і в цьому розділі йому буде приділено найбільшу увагу.

Графічні команди Maple мають деякі особливості в порівнянні з іншими:

1) графічні засоби Maple *повертають* деякі графічні об'єкти і розміщують їх у вікні документа: у рядку виведення або окремому графічному об'єкті;

2) ці об'єкти можна використовувати як значення змінних, тобто змінним можна присвоювати значення графічних об'єктів і виконувати над ними відповідні операції;

3) графічні команди задані таким чином, що забезпечують побудову типових графіків без особливої підготовки. Для цього необхідно лише зазначити математичну функцію, графік якої будується, та бажано – границі зміни незалежних змінних. Але за допомогою необов'язкових параметрів (**опцій графіки**) можна істотно змінити вигляд графіків, наприклад визначити стиль та колір ліній, вивести заголовок, написи координатних осей і т. ін.

Необхідно також зазначити про корисну можливість системи – створення власних рисунків-креслень (**drawing**) за допомогою інструментів **Drawing Tools**. Для того щоб ініціювати креслення, необхідно обрати в головному меню **Insert→Canvas**. У робочий документ буде вставлено заготовку-канву для креслення та активовано головне меню Drawing і контекстну панель інструментів креслення. Вони також активуються при виборі мишкою готових графіків, що дозволяє на них малювати та робити позначки. Оскільки інтерфейс креслення є досить

зрозумілий та інтуїтивний, далі це питання розглядатися не буде. Можна знайти додаткову інформацію в системі допомоги, виконавши команду `?DrawingTools`.

### 4.1.2. Двовимірна графіка. Функція plot та опції

Універсальною командою для побудови двовимірних графіків функцій є команда **`plot(f(x), x=`$x_{min}$`..`$x_{max}$`, v, o)`**, де f(x) – одна функція або їх список, які візуалізуються; x=$x_{min}$..$x_{max}$ – незалежна змінна із зазначенням області її зміни; v – залежна змінна із зазначенням області її зміни (необов'язково); o – опції, що задають стиль побудови графіка.

Набір опцій графіки є досить великий, оскільки саме вони визначають вигляд готового графіка. Детально опис опцій наведено в таблиці 4.1. Частина цих опцій є універсальними і використовуються багатьма функціями графіки, частина – може бути застосована лише для двовимірної графіки.

Таблиця 4.1 – Опції графіки у форматі **keyword**=value («ключове слово = значення»)

| Ключове слово | Значення | Зміст опції |
|---|---|---|
| *1* | *2* | *3* |
| **adaptive** | true або false | Включення спеціального адаптивного алгоритму для побудови графіків найкращого вигляду. Система автоматично збільшує кривину графіка, кількість відрізків прямих у тих частинах, де хід різко відрізняється від інтерполюючої прямої |
| **axes** | normal,<br>boxed,<br>frame<br>або none | Тип координатних осей:<br>  <br>або жодних осей |
| **axesfont** | [див. **font**] | Шрифт для написів поділок на осях |
| **axis** або **axis[dir]** | t | t – список із підопцій, що визначають властивості всіх координатних осей або однієї осі dir (дивись пояснення в тексті нижче) |

Продовження таблиці 4.1

| *1* | *2* | *3* |
|---|---|---|
| **axiscoordinates** | cartesian або polar | Задання системи координат для відображення осей: декартової (за замовчуванням) чи полярної. Ця опція використовується разом із **coords**=polar |
| **caption** | c | c – рядок-підпис рисунку |
| **color** | color_name | Колір кривих, метод задання кольору описаний нижче |
| **coords** | bipolar, cartesian, elliptic, logarithmic, polar та ін. | Задання типу координатної системи. Дізнатися про всі типи доступних систем координат можна, виконавши команду ?coords) |
| **discont** | true або false | Задає, чи буде побудований графік, перервний у разі наявності розривів функції |
| **filled** | true або false | У випадку true задає забарвлення кольором, заданим параметром color, для області, обмеженої побудованою кривою та віссю абсцис |
| **font** | [сім'я, стиль, розмір] | Задає тип шрифту: сімей TIMES, COURIER, HELVETICA, SYMBOL; стилі ROMAN, BOLD, ITALIC, BOLDITALIC (для сім'ї TIMES) та BOLD, OBLIQUE, BOLDOBLIQUE (для сімей COURIER, HELVETICA) |
| **gridlines** | true або false | Задає, чи будуть виведені лінії сітки, їх властивості уточнюють в опції **axis** |
| **labels** | [X,Y] | Задає написи координатних осей відповідно для осей x та y |
| **labeldirections** | [X,Y], де X та Y можуть мати значення horizontal або vertical | Задає горизонтальний чи вертикальний напрям написів для кожної координатної осі |
| **labelfont** | t | Задає списком властивості шрифту для написів координатних осей за загальним правилом опції **font** |
| **legend** | s | Задає виведення легенди, тобто позначення кривих, s – список позначень |

Продовження таблиці 4.1

| *1* | *2* | *3* |
|---|---|---|
| **legendstyle** | [font=f, location=loc] | Задає властивості легенди, де font – тип шрифту за загальними правилами опції **font**; location – розміщення легенди, яке може мати значення top, bottom, right або left |
| **linestyle** | 1 – solid, 2 – dot, 3 – dash, 4 – dashdot, 5 – longdash, 6 – spacedash, 7 – spacedot | Задає стиль ліній графіка числом або ключовим словом (1 – суцільна лінія, 2 – крапки, 3 – тире, 4 – крапка і тире, 5 – довге тире, 6 – пробіл і тире, 7 – пробіл і крапка) |
| **numpoints** | n | Задає мінімальну кількість точок на графіку (за замовчуванням n = 50) |
| **resolution** | n | Задає горизонтальне розділення (за замовчуванням n = 200, використовується при відключеній опції **adaptive**) |
| **scaling** | constrained або unconstrained | Задає стиснутий або нестиснутий масштаб графіка (останнє за замовчуванням) |
| **style** | point, line, polygon або patch | Задає стиль побудови графіка |
| **symbol** | asterisk *****, box □, solidbox ■, circle ○, solidcircle ●, cross **+**, diagonalcross x, diamond ◊, soliddiamond ♦, point ▪ | Задає символи для точок графіка |
| **symbolsize** | n | Задає розмір символів точок графіка (за замовчуванням n = 10) |
| **thickness** | n | Визначає товщину ліній графіка (за замовчуванням n = 0) |
| **tickmarks** | [m, n] | Значення m та n визначають або кількість позначок осей, або їх розміщення (в такому разі зазначають список координат) |
| **title** | t | t задає рядок-заголовок графіка |

Продовження таблиці 4.1

| *1* | *2* | *3* |
|---|---|---|
| **titlefont** | f | Задає властивості шрифту заголовка за загальними правилами опції **font** |
| **transparency** | n | Задає прозорість графіка (n набуває значень від 0 до 1) |
| **view** | [xmin...xmax, ymin..ymax] | Визначає діапазон координат по осях, у межах яких графік буде відображатися на екрані |

Однією з найбільш складних опцій є опція визначення властивостей координатних осей **axis=t** або **axis[dir]=t**. Перший варіант запису цієї опції буде стосуватися всіх осей, другий варіант – лише однієї осі, ім'я якої зазначається в квадратних дужках [dir]. Параметр t – список із підопцій, до яких можна вміщувати такі:

- **color=color_name** (колір осей);

- **gridlines** або **gridlines=g** (лінії сітки; g – список із підопцій для визначення властивостей ліній сітки, серед яких такі: color, linestyle, majorlines, subticks та thickness);

- **location=low, origin** або **high** (розміщення стосовно області виведення на екран);

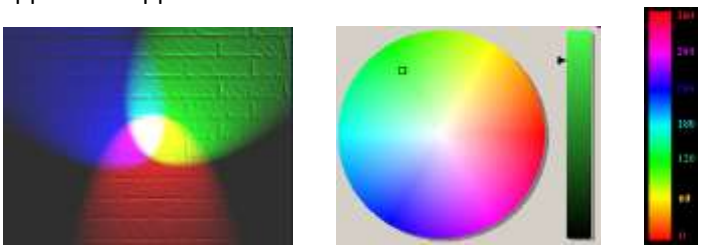
- **mode=linear, log** (лінійний чи логарифмічний масштаб осей);

- **tickmarks** (відмітки осей, властивості задаються так само, як і **gridlines**).

Опція задання кольору **color=color_name** може застосовуватися до багатьох графічних об'єктів, наприклад для задання кольору ліній графіка, координатних осей, позначок на осях, ліній сітки і т. д. У системі Maple передбачені такі головні способи визначення кольору: за ім'ям кольору або за числовими значеннями складових кольору за шкалами RGB, HSV або HUE.

Основні імена кольорів такі: **black, red, green, blue, yellow, orange, magenta, coral, pink, brown, indigo, olive, violet**. Таким чином, колір можна задати так: **color=red** і т. п.

За шкалою RGB колір визначається сумарним вкладом від частки червоного (Red), зеленого (Green) та синього (Blue) кольорів (рис. 4.3 а), тобто відповідними трьома числами (з плаваючою точкою) від 0 до 1. За шкалою HSV колір визначається також трьома параметрами: тоном або кутом на схемі рис. 4.3 б (Hue), насиченістю або відстанню від центра кольорового кола (Saturation), яскравістю (Value). Значення цих параметрів також береться числами з плаваючою точкою від 0 до 1. Схема HUE використовує лише одне значення, яке відповідає обраному тону кольору (відтінку) (див. рис. 4.3 в), тобто задається одним числом із плаваючою точкою.



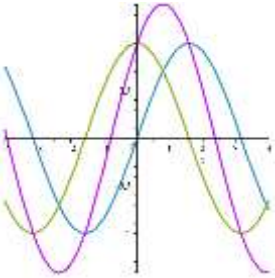
а б в

Рисунок 4.3 – Кольорові схеми визначення кольору за шкалами RGB (а), HSV(б) та HUE(в)

Визначення кольору через шкали RGB, HSV та HUE проводиться через структуру **COLOR()**, в круглих дужках зазначається вид шкали та числові параметри кольору, які відповідають даній шкалі. Наступний приклад ілюструє використання даної структури.

**Приклад 4.1.** Визначити три власні відтінки кольору за шкалами RGB, HSV, HUE та використати їх під час побудови графіків функцій sin(*x*), cos(*x*) та sin(*x*) + cos(*x*).

```
> macro(mycolor1 = COLOR(RGB, 0.200, 0.600, 0.800)) :
  macro(mycolor2 = COLOR(HSV, 0.200, 0.800, 0.700)) :
  macro(mycolor3 = COLOR(HUE, 0.8)) :
> plot([sin(x), cos(x), sin(x) + cos(x)], x =-4 ..4, color = [mycolor1, mycolor2, mycolor3]);
```



У даному прикладі три створені кольори мають імена mycolor1, mycolor2 та mycolor3. Вони застосовуються до графіків конкретних функцій відповідно до порядку їх розміщення в списках у команді **plot**, тобто колір mycolor1 має графік sin(x), колір mycolor2 має графік cos(x), колір mycolor3 має графік sin(x) + cos(x).

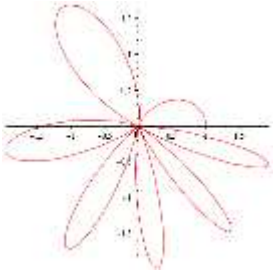
Детальну інформацію стосовно визначення кольору можна знайти в системі допомоги, виконавши команди **?plot,coords**, **?plot,colornames** та **?plot,structure**.

**Приклад 4.2.** Виведення ліній сітки та задання їх властивостей.

```
> plot(x*cos(x), x=0..10, axis=[gridlines=[10, color=blue]])
```

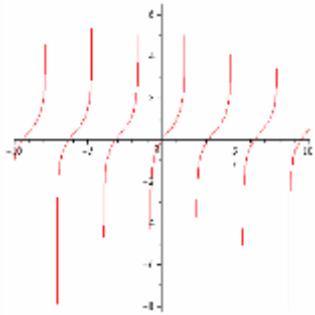
**Приклад 4.3.** Побудова графіка функції в полярній системі координат.

```
> plot([1-sin(t^2), t, t=0..2*π], coords=polar)
```

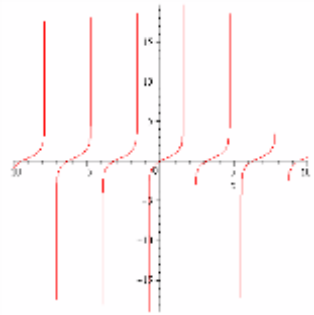


**Приклад 4.4.** Побудова графіка функції, яка має точки розриву, і використання опції **discont=true**.

```
> plot(ln(1 + tan(x)), x)
```

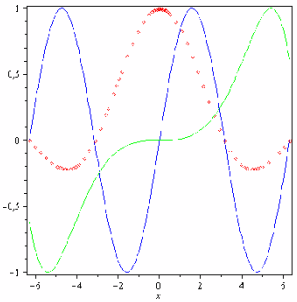


```
> plot(ln(1 + tan(x)), x, discont = true)
```



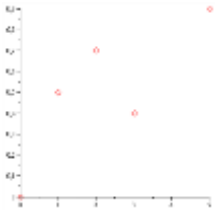
**Приклад 4.5.** Побудова графіків декількох функції в одних координатних осях за допомогою функції plot.

```
> plot([[sin(x), sin(x)/x, sin(x^3/100)]], x = -2*Pi..2*Pi, color = [blue, red,
  green], style = [line, point, line], axes = boxed);
```



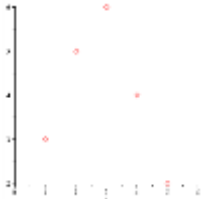
**Приклад 4.6.** Побудова графіків функцій, заданих точками (а) та векторами (б):

```
> plot([[0, 0], [1, 0.5], [2, 0.7], [3, 0.4],
    [5, 0.9]], style = point,
    symbolsize = 20);
```



а

```
> X := [1, 2, 3, 4, 5]:
  Y := [3, 5, 6, 4, 2]:
  XY := [seq([X[i], Y[i]], i = 1..5)];
  plot(XY, x = 0..6, style = point, symbolsize = 20);
                XY := [[1, 3], [2, 5], [3, 6], [4, 4], [5, 2]]
```



б

### 4.1.3. Тривимірна графіка. Функція plot3d

До найперших завдань тривимірної графіки належить графічна побудова функцій двох змінних виду $z = f(x,y)$, де $z$ є висотою, або аплікатою. Особливістю такої побудови є те, що зображення на екрані є пласким, тобто воно є спеціальною проекцією об'ємних об'єктів на площину. Для побудови графіків (поверхонь) функцій типу $z = f(x,y)$ служить команда **plot3d**, синтаксис якої такий:

для декартових або інших координат:

```
> plot3d(z(x,y), x=x1..x2, y=y1..y2, o)
```

або

```
> plot3d(f, x1..x2, y1..y2, o),
```

де $z(x,y)$ – функція, графік якої будується; $x_1..x_2$ та $y_1..y_2$ – інтервали значень незалежних змінних; o – опції графіки;

для параметрично заданої функції:

```
> plot3d([exprf, exprg, exprh], s=a..b, t=c..d, o)
```

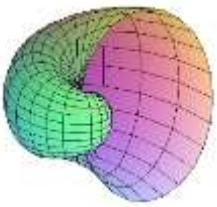
або

```
> plot3d([f, g, h], a..b, c..d, o),
```

де exprf, exprg, exprh – вирази що задають поверхню параметрично; f, g, h – процедури або оператори; a..b та c..d – інтервали значень параметрів $s$ та $t$, через які пов'язані $x$, $y$ та $z$ між собою; o – опції графіки.

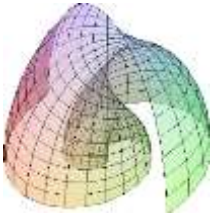
**Приклад 4.7.** Побудова поверхні, заданої явно (а) та параметрично (б).

```
> plot3d(1.3^x sin(y),
  x=-1..2 π, y=0..π,
  coords=spherical)
```



а

```
> plot3d([x sin(x) cos(y),
  x cos(x) cos(y), x sin(y)],
  x=0..2 π, y=0..π,
  transparency=0.5)
```



б

У цьому прикладі застосовано корисну опцію прозорості поверхні **transparency**, щоб бачити побудову детально.

Команда **plot3d** може використовувати багато опцій графіки з тих, які розглядалися для двовимірного випадку (табл.4.1). Існують також специфічні опції, що застосовуються лише для тривимірної графіки, розглянуті в таблиці 4.2.

Таблиця 4.2 – Опції, специфічні для тривимірної графіки

| Ключове слово | Значення | Зміст опції |
|---|---|---|
| **ambientlight** | [R, G, B] | Задає інтенсивність кольорів підсвічування за кольоровою схемою RGB |
| **grid** | [m, n] | Задає кількість ліній каркаса поверхні |
| **gridstyles** | rectangular або triangular | Задає стиль ліній каркаса |
| **light** | [phi, theta, r, g, b] | Задає кути phi та theta, під якими розміщене джерело освітлення поверхні, та інтенсивності складових світла r, g, b за кольоровою схемою RGB |
| **lightmodel** | none, light1, light2, light3 *або* light4 | Задає схему освітлення |
| **orientation** | [theta, phi] | Задає кути орієнтації поверхні відносно площини екрана (за замовчуванням обидва кути дорівнюють 45°) |
| **projection** | 0..1 або fisheye, normal, orthogonal | Задає перспективу при огляданні поверхні (1 відповідає ортогональній перспективі, 0 відповідає ширококутовій перспективі) |
| **shading** | xyz, xy, z, zgreyscale, zhue або none | Спосіб функціонального забарвлення (тіні) |
| **coords** | cylindrical, spherical та ін. | Задає вид тривимірної системи координат (див. систему допомоги **?coords**) |

Опції графіки можна змінювати в процесі роботи з уже створеним графічним об’єктом через використання контекстного меню, а також контекстної панелі інструментів, зображеної на рис. 4.4.

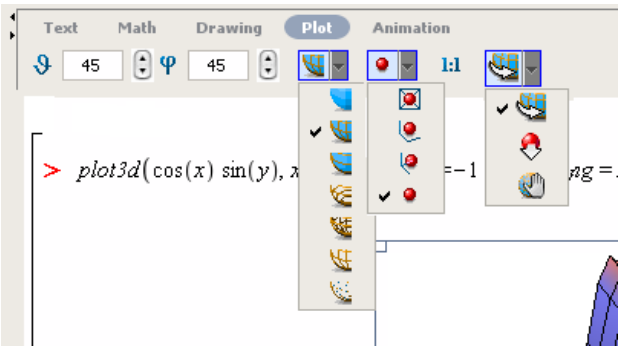
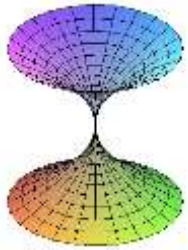



Рисунок 4.4 – Загальний вигляд контекстної панелі інструментів **Plot** під час роботи з тривимірним графічним об’єктом: 1 – кути орієнтації $\vartheta$ та $\varphi$; 2 – вигляд каркасу поверхні; 3 – меню вигляду координатних осей; 4 – масштабування; 5 – інструмент для взаємодії з об’єктом (обертання, масштабування, пересування)

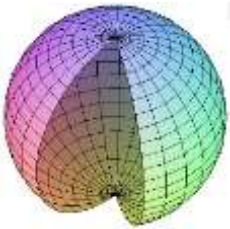
Розглянемо декілька прикладів, що ілюструють можливості побудови складних тривимірних фігур за допомогою команди **plot3d**.

**Приклад 4.8.** Побудова фігур у різних системах координат.

```
plot3d(x^2, a = -π .. π, x = -4 ..4,
coords = cylindrical, style = patch)
```

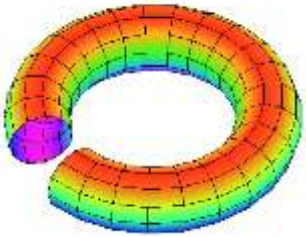


```
plot3d(1, t = π/3 .. 2·π, p = 0 ..Pi,
coords = spherical)
```



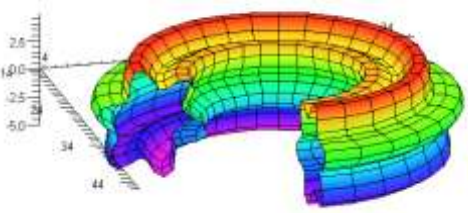
**Приклад 4.9.** Побудова фігури, заданої параметрично.

```
> plot3d( [30 + (7 + 2·cos(s))·cos(t),
          10 + (7 + 2·cos(s))·sin(t),
          2·sin(s)],
          s = 0..2·Pi, t = 0..11·Pi/6,
          grid = [12, 24], scaling = constrained,
          shading = zhue)
```



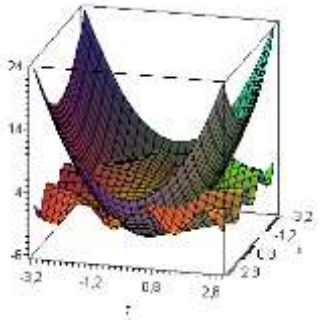
У цьому прикладі будується простий тороїд – циліндр, згорнутий у кільце. При побудові використано прийом видалення частини фігури для більш наочного подання фігури. Насправді параметричне задання рівнянь поверхні відкриває майже невичерпні можливості побудови цікавих та складних фігур різного вигляду:

```
> plot3d([30 + (11 + (4 + sin(6·s))·cos(s))·cos(t),
          20 + (11 + (4 + sin(6·s))·cos(s))·sin(t),
          (4 + sin(6·s))·sin(s)],
          s = 0..2·π, t = 0..3·π/2, grid = [36, 24], orientation = [-60, 64],
          scaling = constrained, shading = zhue)
```



**Приклад 4.10.** Побудова декількох фігур в одних координатних осях.

```
> plot3d([2·sin(x·y) + 1, x^2 + 2·y^2 − 6], x = -Pi..Pi, y = -Pi..Pi,
  axes = boxed, lightmodel = light2)
```



Як бачимо з прикладу, можна будувати одночасно декілька фігур, що перетинаються в просторі. Необхідно задати список функцій поверхонь, і команда **plot3d** автоматично обчислює точки перетину фігур та виводить зображення перетину в природному для сприйняття вигляді.

## 4.2. Створення графічних структур

**Графічні структури** – це термін, який застосовується в системі Maple для визначення складних графічних об'єктів, утворених з елементарних об'єктів типу точка, многокутник та ін. і створені за допомогою функцій PLOT та PLOT3D (зверніть увагу – написання великими літерами).

Графічні структури 2D-графіки будуються командою **PLOT(s1, s2, …, options)**, де $s_i$ – окремі елементарні структури (примітиви), options – додаткові параметри побудови. Кожен з примітивів визначається з високою точністю в заданій системі координат.

Основними примітивами є такі:

- **точки**, задані координатами: **POINTS([x1,y1], [x2,y2], …, [xn,yn])**;
- **криві**, задані координатами точок: **CURVES([[x11,y11], [x12,y12], …, [x1n,y1n]], [[], [],…,[]], …)**;
- **многокутники** – замкнені області, задані координатами вузлів: **POLYGONS([[x11,y11], [x12,y12], …, [x1n,y1n]], [[], [],…,[]], …)**;

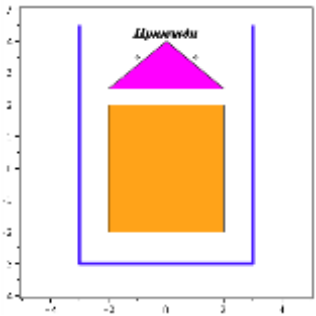
- **текстові написи**: **`TEXT([x,y], 'string', horizontal або vertical)`**, що задані початковими координатами [x,y], змістом напису 'string' та (необов'язково) горизонтальною та/або вертикальною орієнтацією, тобто параметр horizontal може мати значення ALIGNLEFT або ALIGNRIGHT (вліво чи вправо зміщено напис відносно початкових координат), а параметр vertical – ALIGNABOVE або ALIGNBELOW (вгору чи вниз зміщено напис відносно початкових координат).

Основні **опції**, що визначають стиль графічних структур, є такі:

- стиль побудови: STYLE (POINT, LINE, PATCH або PATCHNOGRID);
- товщина ліній: THICKNESS (n);
- вигляд символів для точок: SYMBOL (BOX, CIRCLE, POINT або DIAMOND );
- стиль ліній: LINESTYLE ( ) (див. табл. 4.1);
- колір: COLOR ( ) (див. табл. 4.1);
- шрифт: FONT ( ) (див. табл.4.1);
- заголовок: TITLE ( *рядок* );
- ім'я об'єкта: NAME ( *рядок*);
- стиль координатних осей: AXESSTYLE (BOX, NORMAL, NONE або DEFAULT).

**Приклад 4.11.** Побудова елементарних структур у межах одних координатних осей.

```
> S1 := POINTS([-1, 3.5], [1, 3.5], SYMBOL(DIAMOND)) :
  S2 := CURVES([[-3, 4.5], [-3, -3], [3, -3], [3, 4.5]], THICKNESS(4), COLOR(HUE, 0.7)) :
  S3 := POLYGONS([[-2, 2.5], [2, 2.5], [0, 4]], COLOR(RGB, 1, 0, 1)) :
  S4 := POLYGONS([[-2, 2], [-2, -2], [2, -2], [2, 2]], COLOR(HUE, 0.1)) :
  S5 := TEXT([0, 4],'Приклади', ALIGNABOVE, FONT(TIMES, BOLD, 14)) :
  PLOT(S1, S2, S3, S4, S5, AXESSTYLE(BOX), VIEW(-5..5, -4..5));
```



Для побудови графічних структур тривимірної графіки використовується команда **PLOT3D(s1, s2, …, options)**. Опції використовуються як для звичайної 3D-графіки.

Як примітиви 3D-графіки можна використовувати вищеописані POINTS, CURVES, POLYGONS та TEXT, для яких додають ще значення третьої координати. Серед специфічних 3D-об'єктів є два такі:

- **GRID(a..b, c..d, listlist)**, де **listlist:=[[$z_{11}$, $z_{12}$, …, $z_{1n}$], …, [$z_{m1}$, $z_{m2}$, …, $z_{mn}$]]**;
- **MESH(listlistlist)**, де **listlistlist=[[[$x_{11}$, $y_{11}$, $z_{11}$], ..], ...]**.

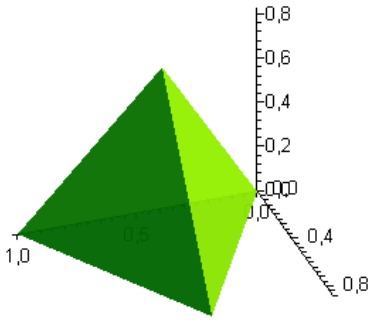
GRID є поверхнею над областю координатної площини (*x*,*y*), обмеженої відрізками a..b, c..d і рівномірно дискретизованої. Дані для побудови задані аплікатами точок у вигляді двовимірного списку listlist із розмірністю *m*x*n*.

MESH є тривимірною поверхнею, що задана тривимірним списком listlistlist, який, у свою чергу, містить повні координати точок поверхні.

Різниця між цими об'єктами полягає в тому, що в першому випадку сітка дискретизації в площині (*x*,*y*) рівномірна, а в другому випадку – задається користувачем.

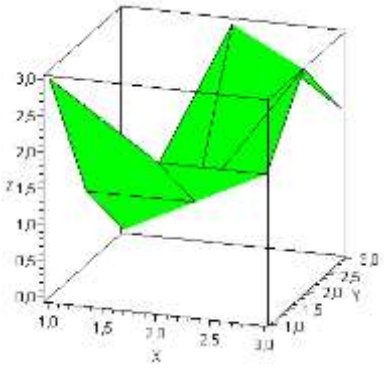
**Приклад 4.12.** Створення тетраедра.

```
> S1 := POLYGONS([[0, 0, 0], [1, 0, 0], [0.5, 0.866, 0]],
                 [[0, 0, 0], [1, 0, 0], [0.5, 0.289, 0.817]],
                 [[1, 0, 0], [0.5, 0.866, 0], [0.5, 0.289, 0.817]],
                 [[0, 0, 0], [0.5, 0.866, 0], [0.5, 0.289, 0.817]]) :
  PLOT3D(S1, TRANSPARENCY(0.2), LIGHT(90, 90, 0.7, 0.8, 0.8),
  AMBIENTLIGHT(0, 0.4, 0.4), AXESSTYLE(NORMAL), STYLE(PATCH), COLOR(RGB, 1, 1, 0));
```

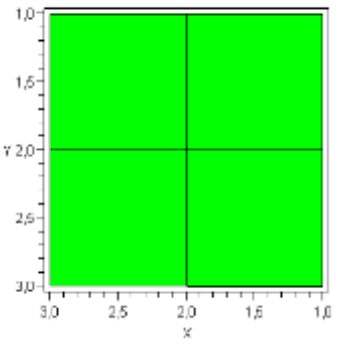


**Приклад 4.13.** Задання графічної структури типу GRID. Отримання зображень під різними кутами.

```
> PLOT3D(GRID( 1..3, 1..3, [[3, 1, 0], [2, 1, 3], [2, 3, 2]]),
  AXESLABELS(X, Y, Z), ORIENTATION( -71, 72), AXES(BOXED),
  COLOR(RGB, 0, 1, 0))
```



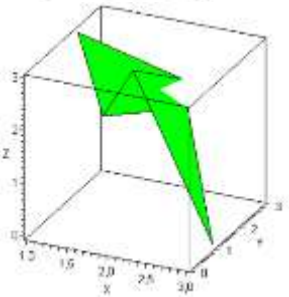

*ORIENTATION*(-71,-72)



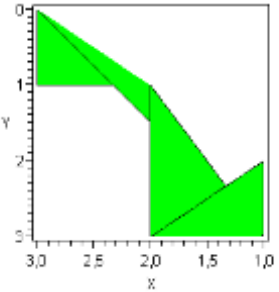

*ORIENTATION*(90,0)

**Приклад 4.14.** Задання графічної структури типу MESH. Отримання зображень під різними кутами.

```
> PLOT3D(MESH([ [[3, 1, 0], [2, 1, 3], [2, 3, 2]],
                [[3, 0, 3], [1, 3, 1], [1, 2, 3]]]),
  AXESLABELS(X, Y, Z), ORIENTATION( -65, 65), AXES(BOXED),
  COLOR(RGB, 0, 1, 0))
```




*ORIENTATION*(-65,-65)




*ORIENTATION*(90,0)

## 4.3. Використання спеціалізованих графічних пакетів команд

### 4.3.1. Пакет plots

До пакета plots входить досить велика кількість графічних функцій для побудови різноманітних 2D- та 3D-графіків. Деякі з основних функцій з цього пакета є такі:

- **animate** – анімація 2D- та 3D-графіків;
- **animatecurve** – анімація кривих;
- **contourplot** та **contourplot3d** – створення відповідно 2D- та 3D- контурних графіків;
- **densityplot** – 2D-графік густини;
- **fieldplot**, **fieldplot3d** – побудова графіків у вигляді 2D- та 3D- векторних полів;
- **gradplot** та **gradplot3d** – побудова графіків у вигляді 2D- та 3D- векторних полів градієнтів;
- **cylinderplot**, **sphereplot**, **polarplot** – побудова графіків у різних координатах: циліндричних, сферичних, полярних;
- **semilogplot**, **logplot**, **loglogplot** – побудова графіків із використанням логарифмічного масштабу відповідно по осі абсцис, ординат та обом осям;
- **pointplot** та **pointplot3d** – побудова точкових 2D- та 3D-графіків;
- **listplot** та **listplot3d**, **listcontplot** та **listcontplot3d**, **listdensityplot** – побудова 2D- та 3D-графіків, якщо вихідна функція задана списками;
- **odeplot** – побудова 2D- та 3D-графіків розв'язків диференціальних рівнянь (див. п. 2.6.4.1);
- **textplot** та **textplot3d** – виведення тексту на графік;
- **display** та **display3d** – виведення декількох графічних об'єктів в одних координатах одночасно.

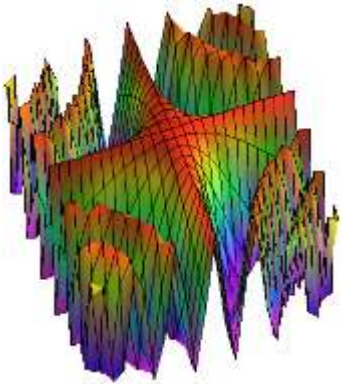
Повний перелік та опис функцій пакета **plots** можна знайти в системі допомоги, виконавши команду **?plots**.

Наступні приклади ілюструють застосування команд цього пакета.

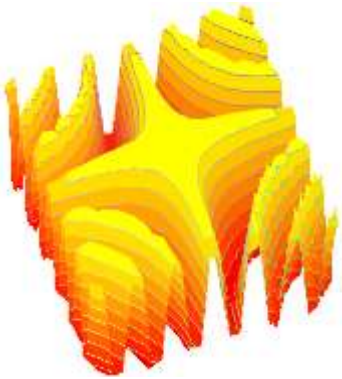
**Приклад 4.15.** Побудова контурних графіків. Контурними є зображення, в яких використовуються лінії однакового рівня, або однакової висоти. Ці лінії отримують, якщо задану поверхню перетнути площиною, паралельною координатній площині (*xy*). Такий вигляд часто використовують у картографії.

Нехай є функція $z = \cos^2(x \cdot y)$. Порівняємо роботу стандартної команди побудови 3D-графіків та команд контурних графіків:

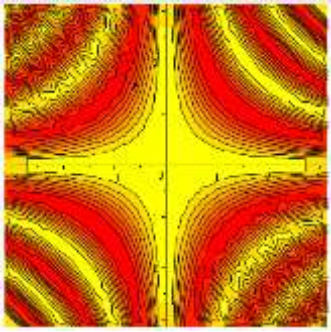
```
> plot3d(cos^2(x*y), x=-π..π, y=-π..π)
```



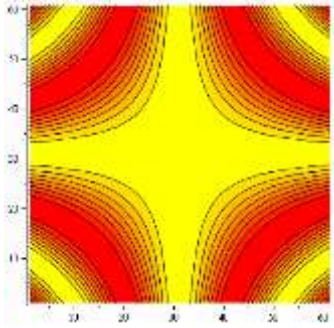
```
> contourplot3d(cos^2(x*y), x=-π..π, y=-π..π, filled=true)
```



```
> contourplot(cos^2(x*y), x=-π..π, y=-π..π, filled=true)
```



```
> listcontplot([seq([seq(cos(x*y), x=-3..3, 0.1)], y=-3..3, 0.1)], filledregions=true)
```



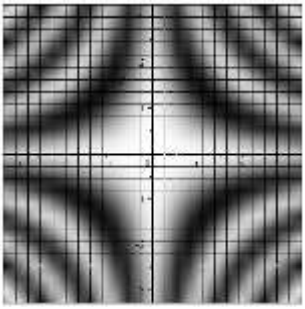
Бачимо, що перевагою контурних зображень є виразність мінімумів та максимумів.

**Приклад 4.16.** Побудова графіків густини і векторних полів.

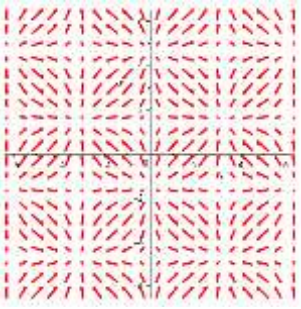
На графіку густини зображення формується так: чим вища висота точки поверхні, тим щільніше (густіше) забарвлення цієї точки.

Векторне поле є зображенням, в якому кожна точка сітки зображується вектором, довжина якого визначається значенням градієнта поля, а напрямок – напрямком зміни градієнта.

```
> densityplot(cos^2(x·y),
  x = -π .. π, y = -π .. π)
```



```
> fieldplot([sin(x), cos(y)],
  x = -2 π .. 2 π, y = -2 π .. 2 π);
```



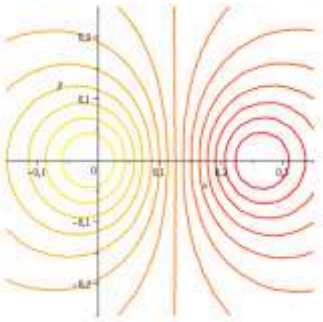
Векторне поле часто застосовують для зображення полів, наприклад електростатичного поля електричних зарядів.

Нехай потенціал електростатичного поля двох зарядів, які знаходяться на відстані 0,25 один від одного, визначається формулою

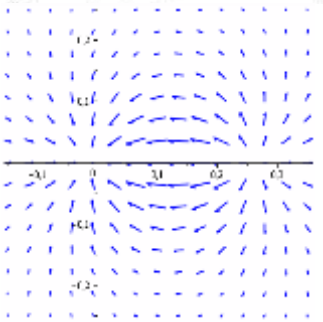
```
> φ := 1/sqrt(x^2 + y^2 + 0.01) - 1/sqrt((x - 0.25)^2 + y^2 + 0.01) :
```

Зобразити поле можна контурним графіком, що буде відображати еквіпотенціальні лінії поля, і векторним полем градієнта, що буде відображати силові лінії:

```
> contourplot(φ, x = -0.15 ..0.35,
  y = -0.25 ..0.25, contours = 15,
  thickness = 2);
```

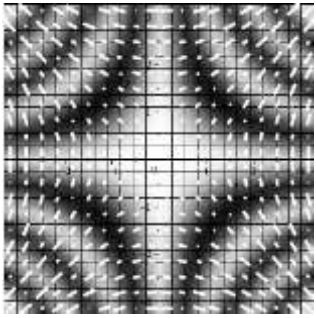


```
> gradplot(φ, x = -0.15 ..0.35,
  y = -0.25 ..0.25, thickness = 2,
  grid = [15,15], color = blue)
```



**Приклад 4.17.** Сумісне відображення різних графіків.

```
> with(plots):
> p1 := densityplot(cos^2(x·y), x=-π..π, y=-π..π);
  p2 := gradplot(cos^2(x·y), x=-π..π, y=-π..π, color=white);
                                        p1 := PLOT(...)
                                        p2 := PLOT(...)
> display(p1, p2)
```



Крім статичних графіків, можна створювати **анімовані графіки**, або динамічні. Вони дозволяють спостерігати процес побудови графіків – кривих та поверхонь, а також зміни графіків при зміні деякого параметра, наприклад часу.

Для цього є дві основні графічні функції: **animatecurve** та **animate**.

Функція **animatecurve** використовується для спостереження побудови кривих і створює послідовність кадрів, що ілюструють цей процес. Синтаксис функції подібний до команди **plot**. Новою опцією для анімованих графіків є frames=n, де n – число, яке задає кількість кадрів мультика.

**Приклад 4.18.** Створення анімації побудови графіка функції $y = \sin(x^2)$.

```
> animatecurve(sin(x^2), x=-π..π, frames=50)
```

У результаті виконання команди **animatecurve** створюється динамічний графік, для якого стає активною панель програвача кліпів (рис. 4.5). Керування анімацією відбувається за допомогою інструментів цієї панелі.

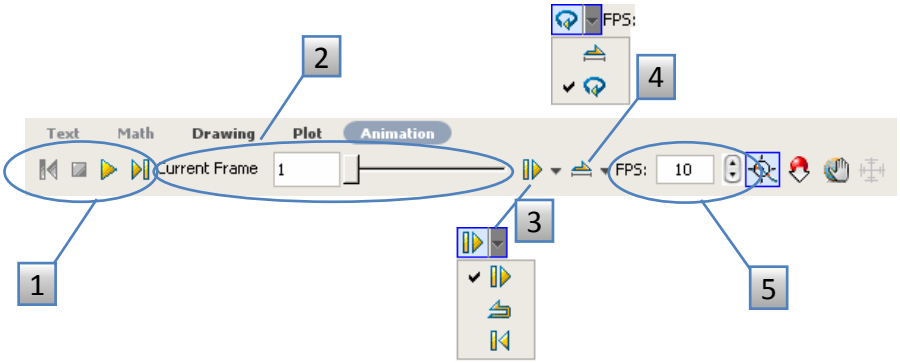



Рисунок 4.5 – Структура панелі програвача кліпів анімації:

1 – запуск анімації, зупинка, перехід до наступного або попереднього кадру (кадр – frame); 2 – лічильник поточного кадру та смуга прокручування; 3 – кнопки встановлення програвання: від початку до кінця, з кінця до початку, по колу "початок – кінець – початок –…"; 4 – установлення одиночного циклу анімації або серії циклів багатократного програвання; 5 – задання швидкості анімації, яка дорівнює кількості кадрів за секунду (FPS – frames per second)

Більші можливості анімації дає функція **animate**, яка створює мультики як для кривих, так і поверхонь. Ці мультики є серією кадрів, і кожен кадр пов'язаний зі значенням спеціального змінюваного параметра *t*. Тому до аргументів даної функції додається інтервал зміни *t* у вигляді *t=a..b*. Явно число кадрів може задаватись опцією frames=n. На відміну від функції **animatecurve** функція **animate** ілюструє не послідовність побудови графіка, а те, яким чином повний вигляд графіка змінюється при зміні додаткового *t*.

**Приклад 4.19.** Використання команди **animate** для анімації 2D- та 3D-графіків.

```
> animate(plot, [t*sin(x), x=-2*Pi..2*Pi], t=0..10);
> animate(plot3d, [t*sin(x*y), x=-2*Pi..2*Pi, y=-2*Pi..2*Pi], t=0..10)
```

### 4.3.2. Графіка пакета plottools

Команди спеціалізованого пакета **plottools** використовуються для створення елементарних графічних структур (примітивів) на площині та в просторі. До них відносять такі примітиви, як відрізки прямих, дуг, кола, конуси, кубики та ін. Ці об'єкти можна потім використовувати у всіх структурах типу plot, наприклад у команді `plots[display]`, тобто на їх основі можна будувати практично будь-які типи складних графічних структур.

Основні примітиви створюються такими командами:

– `arc` – дуга (ділянка кола);

– `arrow` – 2D- або 3D-стрілка;

– `curve` – 2D- або 3D-лінія, що задана списком координат точок;

– `circle` – коло;

– `sphere` – куля;

– `cone` – конус;

– `rectangle` – прямокутник;

– `cuboid` – прямокутний паралелепіпед;

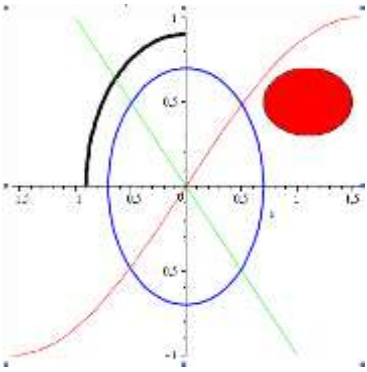
– `octahedron` – октаедр;

– інші.

Над примітивами можна виконувати такі перетворення, як `rotate` (поворот), `scale` (масштабування), `reflect` (віддзеркалення) та інші.

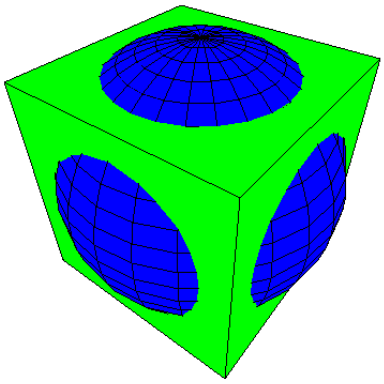
**Приклад 4.20.** Побудова графічних примітивів пакета **plottools** на площині.

```
> with(plottools):
  a := arc([0, 0], 0.9, π/2..π, thickness = 4):
  b := line([1, -1], [-1, 1], color = green):
  c := circle([0, 0], 0.7, thickness = 2, color = blue):
  d := ellipse([1.1, 0.5], 0.4, 0.2, color = red, filled = true):
  e := plot(sin(x), x = -π/2..π/2):
  plots[display](a, b, c, d, e);
```



**Приклад 4.21.** Побудова тривимірних графічних примітивів пакета **plottools**.

```
> p1 := sphere([0, 0, 0], 1, color = blue) :
  p2 := hexahedron([0, 0, 0], 0.8, color = green) :
  plots[display]([p1, p2]) :
```



# 5. Робота з файлами. Взаємодія Maple з іншими програмами

Система Maple не є ізольованою і дозволяє обмінюватися даними з іншими програмними продуктами та працювати з файлами інших форматів, ніж "рідні формати" *.mw чи *.mws. У системі Maple передбачені можливості запису даних та виразів у файл, читання даних та виразів із файла, експорт Maple-документів у деякі інші формати, переведення Maple-коду в коди інших мов програмування, доступ до зовнішніх програм із системи Maple та навпаки.

## 5.1. Робота з файлами

### 5.1.1. Експорт робочих документів

Як відомо, стандартним форматом файлів, створених у стандартному інтерфейсі системі Maple 12, є формат .mw, а при використанні класичного інтерфейсу системи – класичний формат .mws. Як правило, в таких файлах зберігаються всі робочі документи та результати обчислень, для чого обирають пункт **Save** або **Save as…** головного меню **File**. Але система Maple підтримує й інші формати. Так, робочі документи можна експортувати через головне меню **File → Export as...** як файли типу HTML, LaTeX, Maple input, Maplet application, Maple text, Plain text, Maple T.A., Rich Text Format.

Файли типу HTML мають розширення .html та можуть бути відкриті будь-яким HTML-браузером.

Експорт у форматі LaTeX означає створення файла з розширенням .tex, який можна в подальшому обробляти в системі LaTeX.

Maple input є форматом, який може бути прочитаний версією системи командного рядка (Command-line Maple). Такі файли мають розширення .mpl.

Формат Maplet application є форматом програм-меплетів (див. підрозділ 3.3), які зберігаються у файлах .maplet. Вони можуть бути оброблені командним рядком та програмою-оглядачем MapletViewer.

Якщо виконати експорт робочого документа як Maple text, то буде створено текстовий файл .txt, в якому збережена мінімальна та найголовніша різниця між командами, результатами обчислень і текстовими коментарями. Це дозволяє, наприклад, зекономити обсяг файла, зокрема під час пересилання через електронну пошту. Файл може бути відкритий системою Maple, яка відновить структуру робочого документа.

Plain text (.txt) є форматом простого нешифрованого тексту.

Maple T. A. є автоматизованою системою для створення контрольних завдань для студентів та їх перевірки, яка базується на мережі Інтернет. Експортуються такі завдання як zip-файли.

Rich Text Format (.rtf) є форматованим текстом, який може бути відкритим будь-яким текстовим редактором, що підтримує формат rtf.

Ще однією корисною можливістю є вміщення даних, створених у системі Maple, до веб-сторінок як "живого" контенту, наприклад включення динамічних формул та діаграм. Ця можливість реалізується за допомогою системи MapleNet, яка постачається окремо від системи Maple.

### 5.1.2. Запис у файл

Дані, створені в робочому документі Maple, можна записати в простий текстовий файл для подальшої обробки в інших програмах.

Якщо результатом обчислень є довгий список або масив чисел, його можна перетворити на матрицю та записати числа у файл за допомогою команди **`ExportMatrix()`**. Ця команда створює стовпчики даних, що й дозволить їх прочитати далі в інших програмах. Найпростіший синтаксис цієї команди

**ExportMatrix("file.txt",m)**, де file.txt – ім'я файла; m – ім'я експортованої матриці.

Якщо результатом обчислень є одновимірний масив даних, то його можна перетворити на вектор та записати у файл за допомогою команди **ExportVector()**. Її синтаксис аналогічний команді **ExportMatrix**.

У результаті роботи зазначених команд експорту буде створено текстовий файл із даними, а самі команди повертають значення, що відповідає результуючій кількості записаних байтів.

**Приклад 5.1.** Перетворення двовимірного списку *S* на матрицю *M* та експорт у файл.

```
> S := [[1, 2, 3], [a, b, c], [0, exp, π]];
```

$$S := [[1, 2, 3], [a, b, c], [0, \exp, \pi]]$$

```
> M := convert(S, Matrix);
```

$$M := \begin{bmatrix} 1 & 2 & 3 \\ a & b & c \\ 0 & \exp & \pi \end{bmatrix}$$

```
> ExportMatrix("data_01.txt", M);
```

21

У результаті створено файл data_01.txt, який розміщено в папці з установленою системою Maple. Зміст файла буде такий:

```
1       2       3
a       b       c
0       exp     Pi
```

Сама функція **ExportMatrix()** повернула значення 21, що означає кількість експортованих байтів.

Повний синтаксис команд **ExportMatrix()** та **ExportVector()** відповідає такій схемі:

**> ExportMatrix(file, matrix, target, delimiter, transpose)**, де:

– file – ім'я файла, до якого записуються дані, у вигляді рядка;

– matrix – дані у вигляді матриці, що записуються до файла;

– target – додаткова опція у вигляді target=name, що означає формат записуваних даних, де name обирається серед Matlab,

MatrixMarket або delimited (останнє означає наявність розділювачів між даними);

– delimiter – додаткова опція у вигляді delimiter=string, що задає рядок string – знак-розділювач між даними, наприклад, кома delimiter = "," або табуляція delimiter = "\t", або будь-який інший одиночний символ;

– transpose – додаткова опція у вигляді transpose або transpose=true, яка визначає, чи транспонувати дані перед записом у файл.

Якщо необхідно записати у файл складний вираз або процедуру, то можна використати команду **`save()`**, яка запише вираз у файл .m "внутрішнього формату Maple". Це корисно, якщо планується використовувати його в Maple у майбутньому. Подібні записи Maple сприймає більш ефективно саме з внутрішнього формату, а не формату робочого документа.

**Приклад 5.2.** Збереження математичних виразів у файл.

$$> f1 := \frac{1}{1 + \exp\left(\frac{E - E_F}{k_B T}\right)} :$$

$$> f2 := \exp\left(-\frac{E - E_F}{k_B T}\right) :$$

> save f1, f2, "functions.m"

Запис даних у файл може альтернативно проводитися командою **`fprintf(file, format, x1, x2, …, xn)`**, що базується на команді стандартної бібліотеки мови С. Команда записує у файл з ім'ям file набір даних $x_1$, $x_2$, …, $x_n$ відповідно до послідовності специфікацій, заданої параметром format. Цей параметр є рядком (тобто записується у лапках " "), і містить послідовність специфікацій, кожна з яких має загальний вигляд: %[flags][width][.precision][modifiers]code. Починається кожна специфікація символом %, за яким йдуть нуль або інші прапорці flags, мінімальна кількість символів width (додатково), точність precision (додатково означає кількість цифр після десяткової точки для даних типу float або максимальний розмір для даних типу string), модифікатори (додатково) та одна літера коду

форматування. Детально значення цих параметрів можна знайти в системі допомоги, виконавши команду **?fprintf**. Мінімально необхідною для кожного окремого елемента даних є специфікація вигляду %code, наприклад для рядка – %s, для цілого числа – %d, для числа з плаваючою точкою – %f.

Існує аналогічна команда **sprint()**, яка виконує запис не у файл, а у рядок, який і повертає, і тому не має параметра file.

**Приклад 5.3.** Запис даних у файл за допомогою функції **fprintf()**.

```
> x := 5:
  y := sin(x):
> fprintf("file1.dat", "%s\nx = %d, y = %f", "Значення x, y:", x, y):
  fclose("file1.dat"):
```

У цьому прикладі створюється файл з ім'ям file1.dat, і до нього записується три елементи: один рядок і числа *x* та *y*, визначені попередньо. Для переходу у файлі на початок нового рядка використано \n. Після запису файл потрібно закрити командою **fclose()**. Зміст записаного файла матиме такий вигляд:

```
Значення x, y:
x = 5, y = -0.958924
```

### 5.1.3. Читання з файлу

Якщо дані отримані у вигляді стовпчиків чисел і записані в текстовому файлі, то їх можна використати і в системі Maple, виконавши спочатку процедуру їх імпорту. "Читати" з файла часто необхідно з метою завантажити дані, отримані, наприклад, в експерименті.

Окрім числових даних, підтримується також імпорт рисунків, аудіофайлів, а також файлів Excel, Matlab, Matrix Market, Delimited.

Для імпорту даних із файла можна використати помічник, якій знаходиться в головному меню **Tools menu → Assistants → → Import Data**. На першому кроці необхідно обрати тип файла та сам імпортований фай. На другому кроці відкриється віконце, як показано на рис. 5.1, в якому відбувається попереднє читання файла і де необхідно визначити деякі параметри імпорту, а саме: тип даних (Data Type), структуру матриці даних (Source From), знак-розділювач між даними (Separator), чи потрібно

транспонувати дані (Transpose) та які рядки пропустити від початку файла (Skip Lines). Завершується імпорт натисканням кнопки Done.

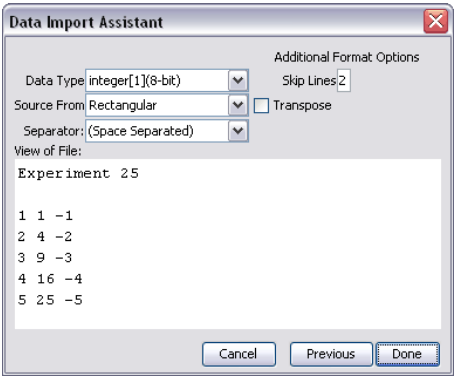



Рисунок 5.1 – Помічник для імпорту даних **Import Data**

Робота цього помічника аналогічна командам **`ImportMatrix()`** або **`ImportVector()`**. Синтаксис цих команд відповідає такій схемі: **`ImportMatrix(file, source, format, datatype, delimiter, transpose, skiplines)`**. Взагалі команди **`ImportMatrix`**, **`ImportVector`**, **`ExportMatrix`** та **`ExportVector`** є спорідненими і мають спільні параметри.

Так, параметри для **`ImportMatrix()`** або **`ImportVector()`** є такі:

– file – рядок-ім'я файла, який читається;

– source – додаткова опція, що визначає формат даних у файлі та задається у вигляді source=Matlab, MatrixMarket, csv або delimited (останнє означає дані, розділені певним символом);

– format – додаткова опція, що визначає формат зберігання даних у файлі і задається у вигляді format=rectangular, entries або vectors (див. пояснення нижче);

– datatype – додаткова опція, що визначає будь-який тип даних результуючої матриці та задається у вигляді datatype=type;

– delimiter – додаткова опція у вигляді delimiter=string, що задає знак, який розділяє дані у файлі, наприклад кома delimiter = "," або табуляція delimiter = "\t";

– transpose – додаткова опція у вигляді transpose або transpose=true, яка визначає, чи транспонувати дані під час імпорту;

– skiplines – додаткова опція, яка має вигляд skiplines=n, де n – натуральне число, і задає, скільки рядків від початку файла потрібно пропустити під час імпорту даних.

Опція format по суті зазначає метод зберігання даних у файлі. Кожен елемент даних визначається розміщенням (номер рядка, номер стовпчика) та значенням. Якщо метод rectangular, то дані розміщуються у вигляді двовимірного прямокутного масиву, де елементи розділені певним розділовим знаком (див. опцію delimiter). Якщо метод entries, то елементи даних у файлі зазначені неупорядковано у вигляді переліку наборів із трьох чисел: (№ рядка, № стовпчика, значення). Якщо метод vectors, то дані зберігаються у вигляді [$v_1$] [$v_2$] [$v_3$], де $v_1$ – вектор індексів рядків; $v_2$ – вектор індексів стовпчиків; $v_3$ – вектор значень.

Для читання виразів, записаних у файли типу .m, використовується команда **`read()`**.

**Приклад 5.4.** Читання виразу з файла, який був створений раніше командою **`save()`** (див., наприклад, приклад 5.2).

```
> read "functions.m"
> fl
```

$$\frac{1}{1+e^{\frac{E-E_F}{k_B T}}}$$

Читання даних із файла може альтернативно проводитися командою **`fscanf(file, format)`**, що базується на команді стандартної бібліотеки мови С. Ця команда аналізує файл відповідно до заданої деталізації форматів, читає символи у файлі, відмічає числа і рядки відповідно до заданого переліку форматів та повертає список просканованих об’єктів.

Параметр format має таку структуру: `%[*][width][modifiers]code`, де * є необов'язковим символом, який за наявності вказує, що об'єкт сканується, але не включається до результату, а width визначає максимальну кількість символів, що буде скануватися. Детально про значення параметрів можна прочитати в системі допомоги, виконавши команду `?fscanf`.

Існує споріднена команда `sscanf(string, format)`, яка читає не файли, а рядки.

**Приклад 5.5.** Читання рядка за допомогою команди `sscanf()`.

```
> sscanf("123.456E7 123.456E7", "%s %f")
```

$$["123.456E7",\ 1.23456\ 10^{9}]$$

Як бачимо, однакові дані з текстового рядка були перетворені на рядок та число типу float відповідно до застосованих специфікацій.

## 5.2. Взаємодія з іншими програмами

У системі Maple є можливість перетворювати коди на мові Maple на коді інших мов, таких, як C, Fortran77, Java, MatLab та Visual Basic. Для цього існує спеціальний пакет команд **CodeGeneration**.

**Приклад 5.6.** Генерація кодів на мовах C, Fortran та Matlab для математичного виразу другої похідної від степеневої функції.

```
> with(CodeGeneration):
> C(diff(x^n, x$2));
cg = pow(x, n) * n * n * pow(x, -0.2e1) - pow(x,n)
     * n * pow(x, -0.2e1);
> Fortran(diff(x^n, x$2));
cg0 = x ** n * n ** 2 / x ** 2 - x ** n * n
      / x ** 2
> Matlab(diff(x^n, x$2))
cg1 = x ^ n * n ^ 2 / x ^ 2 - x ^ n * n / x ^ 2;
```

Крім того, в системі Maple можна здійснити зовнішній виклик, тобто використати скомпільований код C, Fortran або Java в системі Maple. Іншими словами, на функції, що написані на цих мовах, можна посилатися та використовувати так, немов би

вони були процедурами Maple без необхідності переводити їх на мову Maple. Для цього служать інструменти пакета **ExternalCalling**.

Два спеціалізовані пакети **Matlab** та **MmaTranslator** дозволяють звертатися до деяких функцій системи MatLab та Mathematica відповідно.

І навпаки, можна здійснювати доступ до Maple з інших програм. Так, існує спеціальна надбудова MS Excel для взаємодії із системою Maple зсередини. Крім того, є спеціальний набір функцій **OpenMaple**, який дає доступ до алгоритмів Maple та структур даних за допомогою інших програм, наприклад скомпільованих C, Fortran або Java-програм.

# Список рекомендованої літератури

1. Дьяконов В. П. MAPLE 9.5/10 в математике, физике и образовании : учебник / В. П. Дьяконов. – М. : СОЛОН-Пресс, 2006. – 720 с.

2. Говорухин В. Н. Введение в MAPLE. Математический пакет для всех : учебник / В. Н. Говорухин, В. Г. Цибулин. – М. : Мир, 1997. – 208 с.

3. Дьяконов В. П. Maple 7 : учебник / В. П. Дьяконов. – Санкт-Петербург : Питер, 2002. – 672 с.

4. Говорухин В. Н. Компьютер в математическом исследовании / В. Н. Говорухин, В. Г. Цибулин. – СПб. : Питер, 2001. – 624 с.

5. Maple 12 User Manual. – Maplesoft, a division of Waterloo Maple Inc. – 1996–2008.

6. Матросов А. В. MAPLE 6. Решение задач высшей математики и механики : учебник / А. В. Матросов. – СПб. : BHV-Санкт-Петербург, 2001. – 528 с.

7. Enns R. H. Computer Algebra Recipes. An advanced guide to scientific modeling / R. H. Enns, G. C. McGuire. – Springer Science, 2007. – 372 p.

8. Tocci Ch. Applied Maple for engineers and scientists / Ch. Tocci, S. Adams. – Artech. House, 1996. – 406 p.

9. Wang F. Y. Physics with Maple. The Computer Algebra Resource for Mathematical Methods in Physics / F. Y. Wang. – WILEY-VCH, 2005. – 605 p.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

# КОМП'ЮТЕРНІ ТЕХНОЛОГІЇ В ЕЛЕКТРОНІЦІ

Конспект лекцій

для студентів спеціальності
7(8).05080102 «Фізична та біомедична електроніка»
денної форми навчання

Затверджено
на засіданні кафедри
наноелектроніки
як конспект лекцій
з дисципліни «Комп'ютерні
технології в електроніці».
Протокол № 16 від 14.05.2014.

Суми
Сумський державний університет
2014

Навчальне видання

# КОМП'ЮТЕРНІ ТЕХНОЛОГІЇ В ЕЛЕКТРОНІЦІ

## Конспект лекцій

для студентів спеціальності
7(8).05080102 «Фізична та біомедична електроніка»
денної форми навчання

Відповідальний за випуск О. Д. Погребняк
Редактор С. М. Симоненко
Комп'ютерне верстання Ю. О. Космінська

Підписано до друку 11.06.2014, поз.
Формат 60×84/16. Ум. друк. арк. 8,84. Обл.-вид. арк. 8,16. Тираж 35 пр. Зам. №
Собівартість вид. грн к.

Видавець і виготовлювач
Сумський державний університет,
вул. Римського-Корсакова, 2, м. Суми, 40007
Свідоцтво суб'єкта видавничої справи ДК № 3062 від 17.12.2007.