Підручник-довідник із системи комп'ютерної алгебри **Maxima**

Є.А. Чичкарьов
Ю.О. Чорноіван (переклад українською)

18 лютого 2016 р.

Зміст

Вступ **7**

1 Виникнення і розвиток систем комп'ютерної математики **9**
1.1 Означення систем комп'ютерної алгебри . . . 9
1.1.1 Недоліки обчислювальних розрахунків . . . 9
1.1.2 Відмінності символьних обчислень від числових . . . 10
1.2 Класифікація, структура і можливості систем комп'ютерної математики . . . 10
1.2.1 Класифікація систем комп'ютерної математики . . . 10
1.2.2 Задачі систем комп'ютерної алгебри . . . 11
1.2.3 Місце комп'ютерної алгебри в інформатиці . . . 11
1.2.4 Взаємозв'язок систем комп'ютерної алгебри і традиційних математичних дисциплін . . . 11
1.2.5 Можливості підвищення ефективності розв'язання математичних і обчислювальних задач . . . 12
1.3 Комерційні і вільно розповсюджувані системи комп'ютерної математики . . . 12

2 Основи Maxima **15**
2.1 Структура **Maxima** . . . 15
2.1.1 Області математики, підтримувані у **Maxima** . . . 15
2.2 Переваги програми . . . 15
2.3 Встановлення і запуск програми . . . 16
2.4 Інтерфейс **wxMaxima** . . . 16
2.5 Введення найпростіших команд **Maxima** . . . 16
2.5.1 Позначення команд і результатів обчислень . . . 16
2.6 Числа, оператори і константи . . . 17
2.6.1 Введення числової інформації . . . 17
2.6.2 Арифметичні операції . . . 17
2.6.3 Сталі . . . 18
2.7 Типи даних, змінні і функції . . . 18
2.7.1 Списки . . . 18
2.7.2 Масиви . . . 21
2.7.3 Матриці і найпростіші операції з ними . . . 23
2.7.4 Математичні функції . . . 26
2.7.5 Обчислення і перетворення аналітичних виразів . . . 27
2.7.6 Перетворення раціональних виразів . . . 29
2.7.7 Перетворення тригонометричних виразів . . . 31
2.7.8 Перетворення степеневих і логарифмічних виразів . . . 32
2.7.9 Задані користувачем функції . . . 32
2.8 Розв'язування задач елементарної математики . . . 33
2.8.1 Знаходження коренів рівнянь і систем алгебраїчних рівнянь . . . 33
2.9 Побудова графіків і поверхонь . . . 34
2.9.1 Побудова графіка явної функції $y = f(x)$ . . . 34
2.9.2 Побудова графіків функцій, заданих параметрично . . . 34
2.9.3 Побудова кривих у полярній системі координат . . . 35
2.9.4 Побудова тривимірних графіків . . . 37

3 Задачі вищої математики з Maxima **39**
3.1 Операції з комплексними числами . . . 39
3.1.1 Подання комплексних чисел . . . 39
3.1.2 Функції для роботи з комплексними числами . . . 39
3.2 Задачі лінійної алгебри . . . 41
3.2.1 Найпростіші операції з матрицями . . . 41
3.2.2 Обернення матриць і обчислення визначників . . . 42
3.2.3 Характеристичний поліном, власні числа і власні вектори матриці . . . 42

3.2.4 Ортогоналізація . . . 42
3.2.5 Перетворення матриці до трикутної форми . . . 43
3.2.6 Обчислення рангу і мінорів матриці . . . 43
3.2.7 Розв'язування матричних рівнянь . . . 44
3.2.8 Спеціальні функції для розв'язання систем лінійних і поліноміальних рівнянь . . . 45
3.3 Класифікація і основні властивості функцій . . . 47
3.3.1 Основні властивості функцій . . . 47
3.3.2 Границя функції і її властивості . . . 48
3.3.3 Неперервні функції . . . 52
3.3.4 Диференціювання за допомогою пакета **Maxima** . . . 54
3.4 Екстремуми функцій . . . 55
3.4.1 Відшукання максимумів і мінімумів . . . 55
3.4.2 Опуклість функції . . . 59
3.4.3 Диференціювання функцій декількох змінних . . . 64
3.5 Аналітичне і обчислювальне інтегрування . . . 65
3.5.1 Основні команди . . . 65
3.5.2 Інтеграли, що залежать від параметра. Обмеження для параметрів . . . 66
3.5.3 Основні прийоми інтегрування . . . 67
3.5.4 Перетворення Лапласа . . . 68
3.6 Методи теорії наближення в обчислювальному аналізі . . . 68
3.6.1 Наближене обчислення математичних функцій . . . 69
3.6.2 Наближене обчислення визначених інтегралів . . . 73
3.7 Перетворення степеневих рядів . . . 74
3.8 Розв'язування диференціальних рівнянь у **Maxima** . . . 75
3.8.1 Основні визначення . . . 75
3.8.2 Функції для розв'язання диференціальних рівнянь у **Maxima** . . . 76
3.8.3 Розв'язування основних типів диференціальних рівнянь . . . 77
3.8.4 Операторний метод розв'язування . . . 81
3.8.5 Додаткові можливості розв'язання ЗДР . . . 83
3.8.6 Обчислювальні методи розв'язання ЗДР . . . 85
3.9 Ряди Фур'є за ортогональними системами . . . 86
3.9.1 Поняття ряду Фур'є . . . 86
3.9.2 Обчислення коефіцієнтів тригонометричних рядів Фур'є . . . 87
3.9.3 Ряди Фур'є для парних і непарних функцій . . . 88
3.9.4 Розкладання функцій у ряд Фур'є на відрізку $[0, \pi]$ . . . 90
3.9.5 Ряд Фур'є для функцій з періодом 2ℓ . . . 91
3.9.6 Комплексна форма ряду Фур'є . . . 93
3.9.7 Додаткові можливості: пакет `fourie` . . . 94
3.9.8 Додаткові можливості: узагальнені ряди Фур'є . . . 95

4 Обчислювальні методи і програмування з Maxima **97**
4.1 Програмування на вбудованій макромові . . . 97
4.1.1 Умовні оператори . . . 97
4.1.2 Оператори циклу . . . 97
4.1.3 Блоки . . . 98
4.1.4 Функції . . . 99
4.1.5 Транслятор і компілятор у **Maxima** . . . 101
4.2 Введення-виведення у пакеті **Maxima** . . . 102
4.2.1 Введення-виведення даних у консолі . . . 102
4.2.2 Файлові операції введення-виведення . . . 103
4.3 Вбудовані обчислювальні методи . . . 104
4.3.1 Числові методи розв'язування рівнянь . . . 104
4.3.2 Розв'язування рівнянь методом Ньютона: пакет `newton1` . . . 104
4.3.3 Інтерполяція . . . 105
4.3.4 Оптимізація з використанням пакета `lbfgs` . . . 106
4.3.5 Обчислювальне інтегрування: пакет `romberg` . . . 108

5 Обрамлення Maxima **109**
5.1 Класичні графічні інтерфейси **Maxima** . . . 109
5.1.1 Графічний інтерфейс **wxMaxima** . . . 109
5.1.2 Графічний інтерфейс xMaxima . . . 112
5.1.3 Використання редактора **TeXmacs** як інтерфейсу **Maxima** . . . 114
5.1.4 Робота з Maxima з Emacs . . . 114
5.2 Робота з Maxima у KDE: інтерфейс Cantor . . . 115

5.2.1 Документ Cantor . . . 116
5.2.2 Налаштовування . . . 116
5.2.3 Інші можливості Cantor . . . 116
5.3 Інтегроване середовище Sage . . . 117
5.4 Побудова графічних ілюстрацій: пакет `draw` . . . 118

6 Моделювання з Maxima **123**
6.1 Загальні питання моделювання . . . 123
6.1.1 Аналітичні моделі . . . 123
6.1.2 Ідентифіковані моделі . . . 124
6.2 Статистичні методи аналізу даних . . . 125
6.2.1 Введення-виведення матричних даних . . . 125
6.2.2 Функції Maxima для розрахунку описової статистики . . . 125
6.2.3 Перевірка статистичних гіпотез . . . 130
6.2.4 Обчислення коефіцієнтів лінійної регресії . . . 134
6.2.5 Використання методу найменших квадратів . . . 136
6.3 Моделювання динамічних систем . . . 137
6.3.1 Моделювання системи хімічних реакцій . . . 137
6.3.2 Фазові портрети динамічних систем . . . 138
6.3.3 Модель динаміки популяцій . . . 139
6.3.4 Рух твердого тіла . . . 141
6.3.5 Атрактор Лоренца . . . 143
6.3.6 Модель автоколивальної системи: рівняння Ван дер Поля . . . 145

7 Розв'язування фізичних і математичних задач із Maxima **147**
7.1 Операції з поліномами і раціональними функціями . . . 147
7.1.1 Спрощення алгебраїчних виразів . . . 147
7.1.2 Розкладання поліномів і раціональних виразів на множники . . . 148
7.1.3 Розв'язування алгебраїчних рівнянь . . . 148
7.2 Деякі фізичні задачі . . . 149
7.2.1 Обчислення середньої квадратичної швидкості молекул . . . 150
7.2.2 Розподіл Максвела . . . 150
7.2.3 Броунівський рух . . . 151
7.3 Приклад побудови статистичної моделі . . . 151

8 Реалізація деяких обчислювальних методів **155**
8.1 Програмування методів розв'язання нелінійних рівнянь у **Maxima** . . . 155
8.1.1 Метод ділення навпіл . . . 155
8.1.2 Метод простих ітерацій . . . 156
8.1.3 Метод Ньютона (метод дотичних) . . . 157
8.1.4 Модифікований метод Ньютона (метод січних) . . . 157
8.1.5 Метод хорд . . . 158
8.2 Числове інтегрування . . . 158
8.2.1 Огляд методів інтегрування . . . 158
8.2.2 Метод прямокутників . . . 159
8.2.3 Метод середніх прямокутників . . . 159
8.2.4 Метод трапецій . . . 159
8.2.5 Метод Сімпсона . . . 160
8.3 Методи розв'язування систем лінійних рівнянь . . . 161
8.3.1 Загальна характеристика і класифікація методів розв'язування . . . 161
8.3.2 Метод Гауса . . . 161
8.3.3 Метод квадратного кореня . . . 163
8.3.4 Коректність постановки задачі і поняття обумовленості . . . 164
8.3.5 Щодо обчислювальних затрат . . . 166
8.4 Ітераційні методи . . . 166
8.4.1 Матричне формулювання ітераційних методів розв'язування систем лінійних рівнянь . . . 166
8.4.2 Метод простої ітерації . . . 167
8.4.3 Метод Зейделя . . . 168
8.5 Розв'язування звичайних диференціальних рівнянь . . . 170
8.5.1 Методи розв'язування задачі Коші . . . 170
8.5.2 Метод рядів, що не потребує обчислення похідних правої частини рівняння . . . 171

Додатки **175**

Вступ

Цю книгу присвячено відкритим програмним засобам, які надають змогу провести весь цикл розробки якоїсь математичної моделі: від пошуку і перегляду необхідної літератури до безпосереднього розв'язання задачі (аналітичного і/або числового) і підготовки звіту або статті до друку. У ній зроблено спробу пояснити, що система аналітичних обчислень **Maxima** – непоганий вибір для будь-якої навчальної задачі або серйозного дослідження, де потрібна математика – від курсової роботи до наукової або інженерної розробки високого класу. За допомогою цієї системи простіше готувати і виконувати завдання, влаштовувати демонстрації і набагато швидше розв'язувати дослідницькі та інженерні задачі.

На сьогодні комп'ютерні програми цього класу (пропрієтарні – **Maple**, **Mathematica**, **MATLAB**, **MathCad** тощо, або з відкритим кодом) дуже широко застосовуються у наукових дослідженнях, стають одним з обов'язкових компонентів комп'ютерних технологій, використовуваних у творчості.

Ці системи мають зручний інтерфейс, реалізують безліч стандартних і спеціальних математичних операцій, обладнані потужними графічними засобами і мають власні мови програмування. Все це надає широкі можливості для ефективної роботи фахівців різних профілів, про що говорить активне застосування математичних пакетів у наукових дослідженнях і викладанні.

Для студентів системи комп'ютерної математики (**СКМ**) – зручний засіб розв'язання всіляких задач, пов'язаних із символьними перетвореннями (математичний аналіз, вища математика, лінійна алгебра і аналітична геометрія тощо), а також засіб розв'язання задач моделювання статичних (описуваних алгебраїчними рівняннями) і динамічних (описуваних диференціальними рівняннями) систем. Крім того, добротна СКМ – чудовий засіб створення графічних ілюстрацій і документів, що містять математичні формули і розрахунки. На сьогодні для виконання обчислень з усіляких технічних дисциплін студентами-нематематиками широко використовується пакет **MatCad**, в основі якого лежить ядро **Maple**. За певних навичок і наявності документації зв'язка **Maxima**+**TeXmacs** або ядро **Maxima**+інтерфейс **wxMaxima** цілком реальна заміна **MathCad** в Unix-середовищі. А наявність універсального інтерфейсу у формі **TeXmacs** або **Emacs** надає змогу поєднувати в одному документі розрахунки, виконані в **Maxima**, **Octave**, **Axiom** тощо.

Для науковців і інженерів **СКМ** незамінний засіб аналізу постановки всіляких задач моделювання. Під системами комп'ютерної математики розуміють програмне забезпечення, яке надає змогу не тільки виконувати обчислювальні розрахунки на комп'ютері, але і робити аналітичні (символьні) перетворення різних математичних і графічних об'єктів. Все широко відомі математичні пакети: **Maple**, **Matlab**, **Mathematica**, надають змогу виконувати як символьні обчислення, так і використати обчислювальні методи. На сьогодні такі системи є одним з основних обчислювальних інструментів комп'ютерного моделювання у реальному часі і застосовуються у різних галузях науки. Вони відкривають також нові можливості для викладання багатьох навчальних дисциплін, таких як алгебра і геометрія, фізика і інформатика, економіка і статистика, екологія. Застосування **СКМ** істотно підвищує продуктивність праці науковця, викладача вишу, учителя.

Кінцевим продуктом дослідження виступають публікації, приготування, поширення і використання яких у цей час вимагає кваліфікованого застосування комп'ютера. Це стосується редагування тексту, виготовлення графічних матеріалів, ведення бібліографії, розміщення електронних версій в інтернеті, пошуку статей і їхнього перегляду. Де-факто зараз стандартними системами приготування науково-технічних публікацій є різні реалізації пакета **TeX** і текстовий редактор **Word**. Крім того, потрібні мінімальні знання про стандартні формати файлів, конвертори, програми та утиліти, використовувані під час приготування публікацій.

Розділ 1

Виникнення і розвиток систем комп'ютерної математики

1.1 Означення систем комп'ютерної алгебри

Історія математики нараховує близько трьох тисячоріч і умовно може бути розділена на кілька періодів. Перший – становлення і розвиток поняття числа, розв'язання найпростіших геометричних задач. Другий період пов'язаний з появою «Основ» Евкліда і утвердженням добре знайомого нам способу доказу математичних тверджень за допомогою ланцюжків логічних умовиводів.

Наступний етап бере свій початок з розвитку диференціального і інтегрального числення. Нарешті, останній період супроводжується появою і поширенням понять і методів теорії множин і математичної логіки, на міцному фундаменті яких піднімається весь будинок сучасної математики.

Ми живемо на початку нового періоду розвитку математики, що пов'язаний з винаходом і застосуванням комп'ютерів. Насамперед, комп'ютер надав можливість виконувати найскладніші обчислювальні розрахунки для розв'язання тих задач, які неможливо (принаймні, на даний момент) розв'язати аналітично. З'явилося так зване «комп'ютерне моделювання» – ціла галузь прикладної математики, у якій за допомогою найсучасніших обчислювальних засобів вивчається поведінка багатьох складних економічних, соціальних, екологічних та інших динамічних систем.

Вивчення математики дає в розпорядження майбутнього інженера, економіста, науковця не тільки певну суму знань, але і розвиває в ньому здатність ставити, досліджувати і розв'язувати найрізноманітніші задачі. Іншими словами, математика розвиває мислення майбутнього фахівця і закладає міцний понятійний фундамент для освоєння багатьох спеціальних дисциплін. Крім того, саме з її допомогою найкраще розвиваються здатності логічного мислення, концентрації уваги, акуратності і ретельності.

Комп'ютерна алгебра – область математики, що лежить на границі алгебри і обчислювальних методів. Для неї, як і для будь-якої області, що лежить на границі різних наук, важко визначити чіткі границі. Часто говорять, що до комп'ютерної алгебри належать питання занадто алгебраїчні, щоб міститися у підручниках з обчислювальної математики, і занадто обчислювальні, щоб міститися у підручниках з алгебри. При цьому відповідь на питання про те, чи належить конкретна задача до комп'ютерної алгебри, часто залежить від схильностей фахівця.

1.1.1 Недоліки обчислювальних розрахунків

Більшість перших систем комп'ютерної математики (Eureka, Mercury, Excel, Lotus-123, MathCad для MS-DOS, PC MatLab тощо) призначалися для обчислювальних розрахунків. Вони неначе перетворювали комп'ютер у великий програмований калькулятор, здатний швидко і автоматично (за введеною програмою) виконувати арифметичні і логічні операції над числами або масивами чисел. Їхній результат завжди конкретний – це або число, або набір чисел, що представляють таблиці, матриці або точки графіків. Зрозуміло, комп'ютер надає змогу виконувати такі обчислення з немислимою раніше швидкістю, педантичністю і навіть точністю, виводячи результати у вигляді добре оформлених таблиць або графіків.

Однак результати обчислень рідко бувають абсолютно точними в математичному сенсі: як правило, при операціях з дійсними числами відбувається їхнє округлення, обумовлене принциповим обмеженням розрядної сітки комп'ютера при зберіганні чисел у пам'яті. Реалізація більшості обчислювальних методів (наприклад, розв'язання нелінійних або диференціальних рівнянь) також базується на свідомо наближених алгоритмах. Часто через нагромадження похибок ці методи втрачають обчислювальну стійкість і розбігаються, даючи помилкові розв'язки або навіть ведучи до повного краху роботи обчислювальної системи – аж до прикрого «зависання».

Умови появи помилок і збоїв не завжди відомі – їхня оцінка досить складна у теоретичному сенсі і трудомістка на практиці. Тому рядовий користувач, зіштовхуючись із такою ситуацією, найчастіше плутається або, що набагато гірше, невірно витлумачує явно помилкові результати обчислень, «люб'язно» надані йому комп'ютером. Важко підрахувати, скільки «відкриттів» на комп'ютері було відкинуто через те, що спостережувані коливання, викиди на графіках або асимптоти помилково обчислених функцій невірно витлумачувалися як нові фізичні закономірності модельованих пристроїв і систем, тоді як на ділі були лише грубими погрішностями обчислювальних методів розв'язання задач.

Багато вчених справедливо критикували обчислювальні математичні системи і програми реалізації обчислювальних методів за частковий характер одержуваних з їхньою допомогою результатів. Вони не давали можливості одержати загальні формули, що описують розв'язання задач. Як правило, з результатів обчислень неможливо зробити якісь загальні теоретичні, а часом і практичні висновки. Тому, перш ніж використати такі системи у реалізації серйозних наукових проектів, доводилося вдаватися до дорогої і недостатньо оперативної допомоги математиків-аналітиків. Саме вони розв'язували потрібні задачі в аналітичному вигляді і пропонували більш-менш прийнятні методи їхнього обчислювального розв'язання на комп'ютерах.

1.1.2 Відмінності символьних обчислень від числових

Термін «комп'ютерна алгебра» виник як синонім термінів «символьні обчислення», «аналітичні обчислення», «аналітичні перетворення» тощо. Навіть на сьогодні цей термін французькою мовою дослівно означає «формальні обчислення».

У чому основні відмінності символьних обчислень від обчислювальних і чому виник термін «комп'ютерна алгебра»?

Коли ми говоримо про обчислювальні методи, то вважаємо, що всі обчислення виконуються у полі дійсних або комплексних чисел. У дійсності ж усяка програма для ЕОМ має справу тільки зі скінченним набором раціональних чисел, оскільки тільки такі числа представляються у комп'ютері. Для запису цілого числа виділяється звичайно 16 або 32 двійкових символів (бітів), для дійсного – 32 або 64 біта. Це множина не замкнута щодо арифметичних операцій, що може виражатися у різних переповненнях (наприклад, при множенні достатньо великих чисел або при діленні на маленьке число). Ще більш істотною особливістю обчислювальної математики є те, що арифметичні операції над цими числами, виконувані комп'ютером, відрізняються від арифметичних операцій у полі раціональних чисел.

Особливістю комп'ютерних обчислень є неминуча наявність похибки або скінченна точність обчислень. Кожну задачу потрібно розв'язати з використанням наявних ресурсів ЕОМ за прийнятний час із заданою точністю, тому оцінка похибки – важлива задача обчислювальної математики.

Розв'язання проблеми точності обчислень і скінченності одержуваних обчислювальних результатів деякою мірою виконується розвитком систем комп'ютерної алгебри. Системи комп'ютерної алгебри, що здійснюють аналітичні обчислення, широко використовують множину раціональних чисел. Комп'ютерні операції над раціональними числами збігаються з відповідними операціями у полі раціональних чисел. Крім того, обмеження на припустимі розміри числа (кількість знаків у його записі) надає змогу користуватися практично будь-якими раціональними числами, операції над якими виконуються за прийнятний час.

У комп'ютерній алгебрі речовинні і комплексні числа практично не застосовуються, зате широко використовуються алгебраїчні числа. Алгебраїчне число задається своїм мінімальним многочленом, а іноді для його задання потрібно вказати інтервал на прямій або область у комплексній площині, де міститься єдиний корінь даного многочлена. Многочлени відіграють у символьних обчисленнях винятково важливу роль. На використанні поліноміальної арифметики засновані теоретичні методи аналітичної механіки, вони застосовуються у багатьох областях математики, фізики і інших наук. Крім того, у комп'ютерній алгебрі розглядаються такі об'єкти, як диференціальні поля (функціональні поля), що допускають показникові, логарифмічні, тригонометричні функції, матричні кільця (елементи матриці належать кільцям досить загального виду) тощо. Навіть при арифметичних операціях над такими об'єктами відбувається розбухання інформації, і для запису проміжних результатів обчислень потрібен значний об'єм пам'яті ЕОМ.

У наукових дослідженнях і технічних розрахунках фахівцям доводиться набагато більше займатися перетвореннями формул, ніж власне розрахунком. Проте, з появою ЕОМ основна увага приділялася автоматизації обчислень, хоча ЕОМ почали застосовуватися для розв'язування таких задач символьних перетворень, як, наприклад, символьне диференціювання, ще в 50-х роках минулого століття. Активна розробка систем комп'ютерної алгебри почалася наприкінці 60-х років. З того часу створена значна кількість різних систем, що набули різного ступеня поширеності; деякі системи продовжують розвиватися, інші відмирають, і постійно з'являються нові.

1.2 Класифікація, структура і можливості систем комп'ютерної математики

1.2.1 Класифікація систем комп'ютерної математики

На сьогодні системи комп'ютерної математики (СКМ) можна розділити на сім основних класів: системи для обчислень, табличні процесори, матричні системи, системи для статистичних розрахунків, системи для спеціальних розрахунків, системи для аналітичних розрахунків (комп'ютерної алгебри), універсальні системи.

Кожна система комп'ютерної математики має нюанси у своїй архітектурі або структурі. Проте можна дійти висновку, що у сучасних універсальних СКМ наступна типова структура:

Центральне місце займає ядро системи – коди множини заздалегідь компільованих функцій і процедур, що забезпечують досить широкий набір вбудованих функцій і операторів системи.

Інтерфейс дає користувачеві можливість звертатися до ядра зі своїми запитами і одержувати результат розв'язання на екрані. Інтерфейс сучасних СКМ заснований на засобах популярних операційних систем Windows і забезпечує властиві цим системам зручності роботи.

Функції і процедури, включені в ядро, виконуються гранично швидко. Тому об'єм ядра обмежують, але до нього додають бібліотеки не таких часто вживаних процедур і функцій.

Кардинальне розширення можливостей систем і їхня адаптація до розв'язуваних конкретними користувачами задач досягаються за рахунок пакетів розширення систем. Ці пакети (нерідко і бібліотеки) пишуться власною мовою програмування тієї або іншої СКМ, що уможливлює їхнє створення звичайними користувачами.

Ядро, бібліотеки, пакети розширення і довідкова система сучасних СКМ акумулюють знання в області математики, накопичені за тисячоріччя її розвитку.

Зростання інтересу до алгебраїчних алгоритмів є результатом усвідомлення центральної ролі алгоритмів в інформатиці. Їх легко описати на формальній і строгій мові і з їхньою допомогою забезпечити розв'язання задач, давно відомих і таких, що вивчаються уже століттями. У той час як традиційна алгебра має справу з конструктивними методами, комп'ютерна алгебра цікавиться ще і ефективністю, реалізацією, а також апаратними і програмними аспектами таких алгоритмів. Виявилося, що при ухваленні рішення щодо ефективності і визначення продуктивності алгебраїчних методів потрібні багато інших засобів, наприклад, теорія рекурсивних функцій, математична логіка, аналіз і комбінаторика.

У початковий період застосування обчислювальних машин у символьній алгебрі швидко стало очевидним, що безпосередні методи з підручників часто виявлялися досить неефективними. Замість звертання до методів обчислювальної апроксимації комп'ютерна алгебра систематично вивчає джерела неефективності і веде пошук інших алгебраїчних методів для поліпшення або навіть заміни таких алгоритмів.

1.2.2 Задачі систем комп'ютерної алгебри

Перші ЕОМ від початку створювалися для того, щоб проводити складні розрахунки, на які людина витрачала дуже багато часу. Наступним кроком розвитку ЕОМ стали ПК. Ці машини можуть виконувати обчислення різної складності (від найпростіших до найскладніших). Така їхня особливість використовувалася у різних областях знань. Розвиток комп'ютерних математичних систем призвів до появи окремого класу програм, які називають Системами Комп'ютерної Алгебри (CAS).

Головна задача CAS – це обробка математичних виразів у символьній формі. Символьні операції звичайно містять у собі: обчислення символьних або числових значень для виразів, перетворення, зміна форми виразів, знаходження похідної функції однієї або декількох змінних, розв'язання лінійних і нелінійних рівнянь, розв'язання диференціальних рівнянь, обчислення границь, обчислення визначених і невизначених інтегралів, робота із множинами, обчислення і робота з матрицями. На додаток до перерахованого, більшість CAS підтримують різноманітні обчислювальні операції: розрахунок значень виразів при певних значеннях змінних, побудова графіків на площині і у просторі.

Більшість CAS працюють на основі власної високорівневої мови програмування, що надає змогу реалізувати свої власні алгоритми. Наука яка вивчає алгоритми, застосовувані в CAS, називається комп'ютерною алгеброю.

1.2.3 Місце комп'ютерної алгебри в інформатиці

Комп'ютерна алгебра є та частина інформатики, що займається розробкою, аналізом, реалізацією і застосуванням алгебраїчних алгоритмів. Від інших алгоритмів алгебраїчні алгоритми відрізняються наявністю простих формальних описів, існуванням доведень правильності і асимптотичних границь часу виконання, які можна одержати на основі добре розвинутої математичної теорії. Крім того, алгебраїчні об'єкти можна точно представити у пам'яті обчислювальної машини, завдяки чому алгебраїчні перетворення можуть бути виконані без втрати точності і значущості. Звичайно алгебраїчні алгоритми реалізуються у програмних системах, що допускають введення і виведення інформації у символьних алгебраїчних позначеннях.

Завдяки всьому цьому фахівці, що працюють в інформатиці, математиці і у прикладних областях, усе більше цікавляться комп'ютерною алгеброю. Спираючись на протиставлення, можна сказати, що комп'ютерна алгебра розглядає такі об'єкти, які мають занадто обчислювальний характер, щоб зустрічатися у книгах з алгебри, і занадто алгебраїчний характер, щоб бути представленими у підручниках з інформатики. Багато алгоритмів комп'ютерної алгебри можна розглядати як напівобчислювальні (у сенсі Кнута).

1.2.4 Взаємозв'язок систем комп'ютерної алгебри і традиційних математичних дисциплін

Відокремити комп'ютерну алгебру від таких математичних дисциплін, як алгебра, аналіз або обчислювальний аналіз, нелегко.

Системи комп'ютерної алгебри звичайно включають алгоритми для інтегрування, обчислення елементарних трансцендентних функцій, розв'язання диференціальних рівнянь тощо. Особливість згаданих алгоритмів полягає в наступному:

- вони оперують із термінами і формулами і видають вихідну інформацію у символьній формі;
- розв'язання досягається за допомогою деякого типу алгебраїзації задачі (наприклад, похідну від полінома можна визначити суто комбінаторним чином);
- існують методи точного подання величин, обумовлених за допомогою границь, таких, що мають нескінченне обчислювальне подання.

Часто формули, які отримують у результаті виконання алгоритмів комп'ютерної алгебри, використовуються потім як вхідна інформація в обчислювальних процедурах. Наприклад, при інтегруванні раціональних функцій від декількох змінних перше й, можливо, друге інтегрування виконуються у символьному вигляді, а інші – у числовій формі.

Числові процедури використовують арифметику скінченної точності і ґрунтуються на теорії апроксимації. Наприклад, обчислювальна процедура знаходження коренів не завжди може відокремити усі корені, оскільки працює із числами скінченної точності; вона відокремлює лише кластери коренів, діаметр яких залежить від заданої точності подання чисел і багатьох інших параметрів.

У принципі бажано і можливо описувати обчислювальні алгоритми з тією же строгістю, як і алгебраїчні, однак необхідна при цьому деталізація є набагато вищою, а подібність із математичною постановкою задачі менш прозорою. З іншого боку, при використанні деякого алгебраїчного алгоритму точність оплачується більшими – у загальному випадку істотно – часом виконання і потрібним об'ємом пам'яті, чим для його обчислювального аналога.

Проте можна навести багато прикладів таких задач, у яких апроксимація не має значного сенсу. Тому методи символьних обчислень і чисто обчислювальні алгоритми звичайно доповнюють один одного. Сучасні системи комп'ютерної алгебри обов'язково включають той або інший набір стандартних обчислювальних алгоритмів. Сучасні системи, розраховані на використання у першу чергу обчислювальних розрахунків (**MatLab**, його клони тощо) завжди включають більш-менш повний набір функцій, що здійснюють символьні перетворення.

1.2.5 Можливості підвищення ефективності розв'язання математичних і обчислювальних задач

Реалізація на ЕОМ символьної математики відкрила принципово нові можливості використання обчислювальних машин у дослідженнях природничих та прикладних наук. Зараз уже важко вказати область природничих наук, де методи аналітичних обчислень на ЕОМ не знайшли б плідних застосувань. Характерною рисою проблематики символьних перетворень є сполучення досить тонких математичних і алгоритмічних методів з найсучаснішими методами програмування, що ефективно реалізують необчислювальну математикові в рамках програмних систем аналітичних обчислень. До числа останніх належать, наприклад, такі популярні системи, як Macsyma, Reduce, АНАЛІТИК тощо.

Добре відомо, що аналітичні перетворення є невід'ємною частиною наукових досліджень, і найчастіше на їхнє виконання витрачається більше зусиль, ніж на іншу частину досліджень, а для реалізації спеціалізованих методів, наприклад, методів сучасного групового аналізу диференціальних рівнянь, особливе значення має точність аналітичних виразів. Однак обчислення вручну за кожним з подібних методів вимагають непомірно великих витрат часу. Саме тут і допомагають методи комп'ютерної алгебри (КА) і відповідні програмні системи, що є практично єдиним засобом розв'язання таких задач, що вимагають великих витрат часу на обчислення вручну і дуже чутливих до втрати точності при розрахунку на ПК.

Завдяки методам і алгоритмам аналітичних обчислень сучасний комп'ютер стає вже не стільки обчислювальною, скільки загальноматематичною машиною. ПК під силу реалізувати інтегрування і диференціювання символьних виразів, переставляння і перегрупування членів, підставляння у вирази з наступним їхнім перетворенням, розв'язувати диференціальні рівняння тощо.

Аналітичні обчислення (**АО**) є складовою частиною теоретичної інформатики, що займається розробкою, аналізом, реалізацією і застосуванням алгебраїчних алгоритмів. Мети **АО** лежать в області штучного інтелекту, незважаючи на те, що методи усе більше і більше віддаляються від неї. Крім того, використовувані алгоритми вводять у дію усе менш елементарні математичні засоби.

Таким чином, **АО** як самостійна дисципліна, насправді, лежить на границі декількох областей: інформатики, штучного інтелекту, сучасної математики (що використовує нетрадиційні методи), що одночасно збагачує її і робить більше важкої в дослідницькому плані. Найменування цієї наукової дисципліни тривалий час коливався й, нарешті, стабілізувалося як «Calcul formel» у французькій мові, «Computer algebra» – в англійській мові і «аналітичні обчислення» або «комп'ютерна алгебра» – в українській.

Найбільш інтуїтивна мета **АО** полягає у маніпуляції з формулами. Математична формула, описана на одній зі звичайних мов програмування (Fortran, Pascal, Basic,...), призначена тільки для обчислювальних розрахунків, коли змінним і параметрам привласнені обчислювальні значення.

У мові, що допускає **АО**, для цієї формули також можна одержати обчислювальне значення, але, крім того, вона може стати об'єктом формальних перетворень: диференціювання, розкладання в ряд, різних інших розкладань і навіть інтегрування.

Інтелектуальність розроблених на сьогоднішній день **АО** визначається їхнім використанням для організації баз знань за математичними методами у навчанні і освіті. Можна виокремити три види навчання: підготовка фахівців в області **АО** (студенти і аспіранти); навчання роботі із **АО** широкого кола користувачів (знайомство із сучасним інструментом дослідження) і застосування **АО** в освіті математичного і фізичного профілю (інтенсифікація освіти за курсом бакалаврата).

1.3 Комерційні і вільно розповсюджувані системи комп'ютерної математики

CAS були створені в 70-і роки і розвивалися у межах проектів, пов'язаних зі штучним інтелектом. Тому сфера застосування їх досить більша і різноманітна. Першими популярними системами були **Reduce**, **Derive**, **Macsyma**.

Деякі з них дотепер перебувають у продажі. Вільно розповсюджувана версія **Macsyma** – **Maxima**. На даний момент лідерами продажів є **Maple** і **Mathematica**. Обидва ці пакети активно використовуються у математичних, інженерних і інших наукових дослідженнях. Існує багато комерційних систем комп'ютерної алгебри: **Maple**, **Mathematica**, **MathCad** і інших. Вільно розповсюджувані програми: **Axiom**, **Eigenmath**, **Maxima**, **Yacas** тощо.

Успіх у сучасному використанні **CAS** полягає в інтеграції всіх машинних можливостей (символьний і обчислювальний інтерфейс, вбудована графіка, мультиплікація, бази і банки даних тощо). Всі сучасні комерційні системи комп'ютерної математики (**Mathematica**, **Maple**, **MatLab** і **Reduce**) мають стандартний набір можливостей:

- є вхідна макромова для спілкування користувача із системою, що включає спеціалізований набір функцій для розв'язання математичних задач;
- є основні символьні (математичні) об'єкти: поліноми, ряди, раціональні функції, вирази загального вигляду, вектори, матриці;
- системи використовують цілі, раціональні, дійсні, комплексні числа;
- є декілька режимів роботи, які взаємно доповнюються: редагування, діагностика, діалог, протокол роботи;
- є зв'язок із засобами розробки програм: можливі підставляння, обчислення значень, генерація програм, використання стандартного математичного забезпечення (бібліотек);
- використовуються інтерфейси для зв'язку з офісними засобами, базами даних, графічними програмними засобами тощо;

Хоча між системами є розходження, синтаксис асоційованих мов не є проблемою, що утрудняє використання систем комп'ютерної математики. Синтаксис мов систем у значній мірі аналогічний синтаксису Паскаля. Обов'язково є оператори присвоювання, поняття зухвалої функції (команди), більш-менш багатий вибір керівних структур (if, do, while, repeat тощо.), можливості для визначення процедур,... – загалом, весь арсенал класичних мов програмування, потрібний для запису алгоритмів.

Системи комп'ютерної алгебри можна умовно розділити на системи загального призначення і спеціалізовані. До систем загального призначення належать **Macsyma**, **Reduce**, **Mathematica**, **Maple**, **Axiom** і інші системи.

В 80-і роки минулого століття широке поширення у колишньому СРСР одержала система **Reduce**. Вона спочатку призначалася для розв'язання фізичних задач, розроблялася на найбільше широко розповсюджених комп'ютерах, розробка до певного часу не носила комерційного характеру (система до кінця 80-х років поширювалася безкоштовно). Відкритий характер системи дозволив залучити до її розробки величезну армію користувачів, що збагатили систему численними пакетами для розв'язання окремих задач.

Macsyma, так само, як і **Reduce**, є «старою» системою. На відміну від системи **Reduce**, **Macsyma** розроблялася із самого початку як комерційний продукт. У ній більш ретельно пророблені алгоритмічні питання, її ефективність істотно вище, але менше її поширення можна пояснити двома обставинами: тривалий час вона була реалізована тільки на малому числі «екзотичних» комп'ютерів і поширювалася тільки на комерційній основі.

Система **Maple**, створена в 80-х роках минулого століття у Канаді, із самого початку була задумана як система для персональних комп'ютерів, що враховує їхні особливості. Вона розвивається «ушир і вглиб», навіть її ядро переписувалося з однієї алгоритмічної мови на іншу. На сьогодні **Maple** широко застосовується у багатьох країнах (зокрема, у США і Канаді) у навчальному процесі, а також у різних областях наукових і технічних досліджень.

Наприкінці минулого століття одержала широке поширення і зараз швидко розвивається система **Mathematica**. Її успіх у значній мірі пояснюється її широкими графічними можливостями, а також електронною документацією, яку можна розглядати як електронну бібліотеку, присвячену різним розділам математики і інформатики.

Особливе місце серед систем комп'ютерної алгебри займає система **Axiom**. На відміну від інших систем, що представляють собою пакети програм, спілкування з якими здійснюється деякою алголо-подібною мовою, система **Axiom**, що розвинулася із системи Scratchpad-II, має справу зі звичнішими для математиків об'єктами. Зокрема, у ній ключовим поняттям є поняття категорії: тут можна розглядати, наприклад, категорії множин, напівгруп, диференціальних кілець, лівих модулів тощо. Система має високий ступінь універсальності, вимагає для своєї реалізації потужних комп'ютерів, поширюється за досить високу плату, тому використовується тільки в обмеженому числі потужних університетських і наукових центрів.

Спеціалізовані системи відрізняються більше високою ефективністю, але область їхнього застосування обмежена. До спеціалізованих систем належать такі системи, як **Caley** і **GAP** – спеціалізовані системи для обчислень у теорії груп, **Macauley**, **CoCo**, **Singular** – системи різного степеня універсальності для обчислень у кільці многочленів, **Schoonship** – спеціалізована система для обчислень у фізиці високих енергій, **muMath** і її правонаступниця **Derive** – системи, широко використовувані у навчальному процесі (зокрема, в Австрії ліцензія на встановлення системи **Derive** придбана для всіх середніх шкіл), і багато які інші.

Maple – це система для аналітичного і обчислювального розв'язання математичних задач, що виникають як у математику, так і у прикладних науках. Розвинена система команд, зручний інтерфейс і широкі можливості дозволяють ефективно застосовувати **Maple** для розв'язання проблем математичного моделювання.

Maple складається з ядра, процедур, написаних мовою C і найвищою мірою оптимізованих, бібліотеки, написаної на **Maple**-мові, і інтерфейсу. Ядро виконує більшість базових операцій. Бібліотека містить множину команд і процедур, виконуваних у режимі інтерпретації. Програмуючи власні процедури, користувач може поповнювати ними

стандартний набір і, таким чином, розширювати можливості **Maple**. Робота у **Maple** відбувається у режимі сеансу (session). Користувач уводить пропозиції (команди, вираз, процедури тощо), які сприймаються Maple.

Типово, результати сеансу зберігаються у файлі з розширенням 'ms'. Якщо задано режим збереження стану сеансу (session), то у файлі із суфіксом «m» будуть записані поточні призначення.

Mathematica – це широко використовувана CAS споконвічно розроблена Стівеном Вольфрандом, що продається компанією Wolfram Research. Він почав роботу над **Mathematica** в 1986 році, а випустив в 1988 році. **Mathematica** не лише CAS, але і потужна мова програмування. Ця мова програмування реалізована на основі об'єктно-орієнтованого варіанта мови C, розширюваного за допомогою так званих бібліотек коду. Ці бібліотеки являють собою текстові файли, написані мовою **Mathematica**.

Архітектура **Mathematica** представлена ядром і користувацьким інтерфейсом. Ядро програми відповідає за інтерпретацію програм, написаних мовою Mathematica, і безпосередньо займається обчисленнями. Користувацькі інтерфейси призначені для виведення результатів у формі, зрозумілій користувачеві. На думку компанії-розробника, більша частина користувачів **Mathematica** – це технічні професіонали. Також **Mathematica** широко використовується в освіті. Зараз кілька тисяч курсів на основі цього продукту читаються в багатьох навчальних закладах, починаючи від середньої школи і закінчуючи аспірантурою. **Mathematica** використовується у найбільших університетах по усьому світі і у групі компаній Fortune 500, а також у всіх 15 основних міністерствах уряду США.

MathCad – це CAS дуже схожа на **Mathematica**. Поширюється компанією Mathsoft. **MathCad** орієнтований на підтримку концепцій робочого аркуша. Рівняння і вираз показуються на робочому аркуші так, як вони виглядали б на якійсь презентації, а не так, як виглядають мовою програмування. Деякі задачі, які виконує програма: розв'язання диференціальних рівнянь, графіки на площині і у просторі, символьне числення, операції з векторами і матрицями, символьне розв'язання систем рівнянь, підбирання графіків, набір статистичних функцій і імовірнісних розподілів. На думку розробників **MathCad**, головний конкурент цього пакета – електронні таблиці.

Багато користувачів використовують електронні таблиці або мови програмування для виконання обчислень. Але ні ті, ні інші не можуть впоратися із задачею, коли справа доходить до обробки отриманих даних. Електронні таблиці розроблені для бухгалтерських, а не для інженерних розрахунків! Для останніх вони не надто зручні: рівняння заховані у комірках, складно вставити коментарі. Це робить роботу досить марудною, а усувати помилки і розбиратися в чиїхось обчисленнях взагалі складно. Електронні таблиці важкі для розуміння і повторного використання іншими користувачами.

Yacas – це Open Source CAS загального призначення. Базується на власній мові програмування, головною метою при розробці цієї мови була простота реалізації нових алгоритмів. Ця мова дуже схожа на **Lisp**, підтримує введення і виведення у звичайному текстовому режимі як інтерактивно, так і в режимі пакетного виразу.

Maxima є нащадком **DOE Macsyma**, що почала своє існування наприкінці 1960 року у MIT. **Macsyma** перша створила систему комп'ютерної алгебри, вона проклала шлях для таких програм як **Maple** і **Mathematica**. Головний варіант **Maxima** розроблявся Вільямом Шелтером з 1982 до 2001 року. В 1998 році він одержав дозвіл на реалізацію відкритого коду на GPL. Завдяки його вмінню **Maxima** зуміла вижити і зберегти свій оригінальний код у робочому стані. Незабаром Вільям передав **Maxima** групі користувачів і розробників, які зберегли її у робочому стані. На сьогоднішній день пакет досить активно розвивається, і в багатьох аспектах не поступається таким розвиненим системам комп'ютерної математики, як **Maple** або **Mathematica**.

Розділ 2

Основи Maxima

2.1 Структура Maxima

Пакет **Maxima** складається з інтерпретатора макромови, написаного на **Lisp**, і декількох поколінь пакетів розширень, написаних на макромові пакета або безпосередньо на **Lisp**. **Maxima** надає змогу розв'язувати досить широке коло задач, що належать до різних розділів математики.

2.1.1 Області математики, підтримувані у Maxima

- Операції з поліномами (маніпуляція раціональними і степеневими виразами, обчислення коренів тощо)
- Обчислення з елементарними функціями, зокрема з логарифмами, експоненційними функціями, тригонометричними функціями
- Обчислення зі спеціальними функціями, зокрема еліптичними функціями і інтегралами
- Обчислення границь і похідних
- Аналітичне обчислення визначених і невизначених інтегралів
- Розв'язування інтегральних рівнянь
- Розв'язування алгебраїчних рівнянь і їхніх систем
- Операції зі степеневими рядами і рядами Фур'є
- Операції з матрицями і списками, більша бібліотека функцій для розв'язання задач лінійної алгебри
- Операції з тензорами
- Теорія чисел, теорія груп, абстрактна алгебра

Перелік додаткових пакетів для **Maxima**, які потрібно завантажувати перед використанням, істотно розширювальні її можливості і коло розв'язуваних задач, наведений у додатку 1.

2.2 Переваги програми

Основними перевагами програми **Maxima** є:

- можливість вільного використання (Maxima ставиться до класу вільних програм і поширюється на основі ліцензії GNU);
- можливість функціонування під керуванням різних ОС (зокрема Linux і Windows™);
- невеликий розмір програми (дистрибутив займає порядку 23 мегабайт, у встановленому виді з усіма розширеннями буде потрібно близько 80 мегабайт);
- широкий клас розв'язуваних задач;
- можливість роботи як у консольній версії програми, так і з використанням одного із графічних інтерфейсів (xMaxima, **wxMaxima** або як додаток (plug-in) до редактора TexMacs);
- розширення **wxMaxima** (що входить у комплект програми) надає користувачеві зручний і зрозумілий інтерфейс, рятує від потреби у вивченні особливостей введення команд для розв'язання типових задач;
- інтерфейс програми українською мовою;

2.3 Встановлення і запуск програми

Отримати останню версію програми можна з її сайта у мережі Інтернет: http://maxima.sourceforge.net/.

Система комп'ютерної алгебри **Maxima** є частиною більшості дистрибутивів **Linux**, однак найчастіше перебуває у списку додаткових програм, які можна отримати у інтернеті у версії для даного дистрибутива. Приклади і розрахунки у цій книзі виконані з використанням дистрибутива Mageia 5.

2.4 Інтерфейс wxMaxima

Для зручності роботи відразу звернемося до графічного інтерфейсу **wxMaxima**, тому що він є найбільш дружнім для користувачів-початківців. Перевагами **wxMaxima** є:

- можливість графічного виведення формул (див. ілюстрації нижче)
- спрощене введення найпоширеніших функцій (через діалогові вікна), а не набирання команд, як у класичній **Maxima**.
- поділ вікна введення даних і області виведення результатів (у класичної **Maxima** ці області об'єднані, і введення команд відбувається в єдиній робочій області з отриманими результатами).

Розгляньмо робоче вікно програми. Згори вниз розташовуються: текстове меню програми – доступ до основних функцій і налаштувань програми. У текстовому меню **wxMaxima** можна знайти функції для розв'язання великої кількості типових математичних задач, розділені за групами: рівняння, алгебра, аналіз, спрощення, графіки, обчислення. Введення команд через діалогові вікна спрощує роботу із програмою для новачків.

При використанні інтерфейсу **wxMaxima**, ви можете позначити у вікні виведення результатів необхідну формулу і викликавши контекстне меню правою кнопкою миші скопіювати будь-яку формулу у текстовому вигляді, у форматі TEXабо у вигляді графічного зображення, для наступного вставлення в якийсь документ.

Також у контекстному меню, при виборі результату обчислення, вам буде запропонований ряд операцій з позначеним виразом (наприклад спрощення, розкриття дужок, інтегрування, диференціювання тощо).

2.5 Введення найпростіших команд Maxima

Всі команди вводяться у рядок введення, роздільником команд є символ ; (точка з комою). Після введення команди слід натиснути клавішу `Enter`[1] для її обробки і виведення результату. У ранніх версіях **Maxima** і деяких її оболонках (наприклад, xMaxima) наявність крапки з комою після кожної команди є строго обов'язковою. Завершення введення символом $ (замість крапки з комою) надає змогу обчислити результат введеної команди, але не виводити його на екран. У випадку, коли вираз треба показати, а не обчислити, перед ним слід поставити знак ' (одинарні лапки). Але цей метод не працює, коли вираз має явне значення, наприклад, вираз $\sin(\pi)$ замінюється на значення рівне нулю.

Двоє одинарних лапок послідовно, застосовані до виразу у вхідному рядку, приводять до заміщення вхідного рядка результатом обчислення виразу, що вводиться.

Приклад:

```
(%i1) aa: 1024;
(%o1) 1024
(%i2) bb:19;
(%o2) 19
(%i3) sqrt(aa)+bb;
(%o3) 51
(%i4) '(sqrt(aa)+bb);
(%o4) sqrt(aa)+bb
(%i5) ''%;
(%o5) 51
```

2.5.1 Позначення команд і результатів обчислень

Після введення, кожній команді надається порядковий номер. У розглянутому прикладі, уведені команди мають номери 1-5 і позначаються відповідно (%i1), (%i2) тощо.

Результат обчислення також має порядковий номер, наприклад (%o1), (%o2) тощо, де «i» – скорочення від англ. *input* (введення), а «o» – англ. *output* (виведення). Цей механізм надає змогу уникнути в наступних обчисленнях повторення повного запису вже виконаних команд, наприклад (%i1)+(%i2) буде означати додавання до виразу першої команди виразу другої і наступне обчислення результату. Також можна використати і номери результатів обчислень, наприклад (%o1)*(%o2). Для останньої виконаної команди у **Maxima** є спеціальне позначення – «%».

[1] У **wxMaxima** потрібно натиснути `Ctrl`+`Enter`.

Приклад:
Обчислити значення похідної функції $y(x) = x^2 \cdot e^{-x}$:

```
(%i1) diff(x^2*exp(-x),x);
(%o1) 2xe^-x - x^2e^-x
(%i2) f(x):=''%;
(%o2) f(x) := 2xe^-x - x^2e^-x
```

Подвійні лапки перед символом попередньої операції дозволяють замістити цей символ значенням, тобто текстовим рядком, отриманим у результаті диференціювання.
Інший приклад (з очевидним змістом):

```
(%i3) x:4;
(%o3) 4
(%i4) sqrt(x);
(%o4) 2
(%i5) %^2;
(%o5) 4
```

2.6 Числа, оператори і константи

2.6.1 Введення числової інформації

Правила введення чисел у **Maxima** точно такі, як і для багатьох інших подібних програм. Ціла і дробова частина десяткових дробів розділяються символом крапки. Перед від'ємними числами ставиться знак мінус. Чисельник і знаменник звичайних дробів розділяється за допомогою символу / (прямий слеш). Зверніть увагу, що якщо в результаті виконання операції виходить деякий символьний вираз, а потрібно одержати конкретне числове значення у вигляді десяткового дробу, то розв'язати цю задачу дозволить застосування прапорця `numer`. Зокрема він надає змогу перейти від звичайних дробів до десяткових. Перетворення до форми із рухомою крапкою здійснює також функція `float`.

```
(%i1) 3/7+5/3;
(%o1)
```

$$\frac{44}{21}$$

```
(%i2) 3/7+5/3, float;
(%o2)
```

$$2.095238095238095$$

```
(%i3) 3/7+5/3, numer;
(%o3)
```

$$2.095238095238095$$

```
(%i4) float(5/7);
(%o4)
```

$$0.71428571428571$$

2.6.2 Арифметичні операції

Позначення арифметичних операцій у **Maxima** нічим не відрізняється від класичного подання: + , — , * , /. Піднесення до степеня можна позначати декількома способами: ^, ^^, **. Видобування кореня степеня n записуємо, як степінь $\frac{1}{n}$. Операція знаходження факторіала позначається знаком оклику, наприклад 5!. Для збільшення пріоритету операції, як і у математиці, використовуються круглі дужки: (). Список основних арифметичних і логічних операторів наведений у таблицях 2.1 і 2.2 нижче.

+	оператор додавання
-	оператор віднімання або зміни знака
*	оператор множення
/	оператор ділення
^ або **	оператор піднесення до степеня

Табл. 2.1. Арифметичні оператори

<	оператор порівняння менше
>	оператор порівняння більше
<=	оператор порівняння менше або дорівнює
>=	оператор порівняння більше або дорівнює
#	оператор порівняння не дорівнює
=	оператор порівняння дорівнює
and	логічний оператор і
or	логічний оператор або
not	логічний оператор не

Табл. 2.2. Логічні оператори

Назва	**Позначення**
ліворуч (у границях)	minus
праворуч (у границях)	plus
плюс нескінченність	inf
мінус нескінченність	minf
число π	%pi
e (експонента)	%e
уявна одиниця $\sqrt{-1}$	%i
Істина	true
Хиба	false
Золотий переріз $(1+\sqrt{5})/2$	%phi

Табл. 2.3. Основні сталі **Maxima**

2.6.3 Сталі

У **Maxima** для зручності обчислень є ряд вбудованих сталих. Найпоширеніші з них показані у таблиці 2.3:

2.7 Типи даних, змінні і функції

Для зберігання результатів проміжних розрахунків застосовуються змінні. Зазначимо, що при введенні назв змінних, функцій і сталих важливий регістр літер, так змінні x і X – дві різні змінні. Присвоювання значення змінної здійснюється з використанням символу : (двокрапка), наприклад `x:5`. Якщо потрібно вилучити значення змінної (очистити її), то застосовується метод `kill`:

`kill(x)` – вилучити значення змінної x;

`kill(all)` – вилучити значення усіх використовуваних раніше змінних.

Зарезервовані слова, використання яких як назв змінних призводить до синтаксичної помилки: `integrate`, `next`, `from`, `diff`, `in`, `at`, `limit`, `sum`, `for`, `and`, `elseif`, `then`, `else`, `do`, `or`, `if`, `unless`, `product`, `while`, `thru`, `step`.

2.7.1 Списки

Списки – базові будівельні блоки для **Maxima** і **Lisp**. Всі інші типи даних (масиви, хеш-таблиці, числа) представляються як списки. Щоб задати список, досить записати його елементи через кому і обмежити запис квадратними дужками. Список може бути порожнім або складатися з одного елемента.

```
(%i1) list1:[1,2,3,x,x+y];
(%o1) [1,2,3,x,y + x]
(%i2) list2:[] ;
(%o2) []
(%i3) list3:[3];
(%o3) [3]
```

Елементом списку може і інший список

```
(%i4) list4: [1,2, [3,4], [5,6,7]];
(%o4) [1, 2, [3,4], [5, 6, 7]]
```

Посилання на елемент списку виконується за номером елемента списку:

```
(%i4) list4:[1,2,[3,4],[5,6,7]];
(%o4) [1,2,[3,4],[5,6,7]]
(%i5) list4[1];
(%o5) 1
```

```
(%i6) list4[3];
(%o6) [3,4]
(%i7) list4[3][2];
(%o7) 4
```

2.7.1.1 Функції для елементарних операцій зі списками

Функція `length` повертає кількість елементів списку (при цьому елементи списку самі можуть бути досить складними конструкціями):

```
(%i8) length(list4);
(%o8) 4
(%i9) length(list3);
(%o9) 1
```

Функція `copylist(expr)` повертає копію списку expr:

```
(%i1) list1:[1,2,3,x,x+y];
(%o1) [1, 2, 3, x, y + x]
(%i2) list2:copylist(list1);
(%o2) [1, 2, 3, x, y + x]
```

Функція `makelist` створює список, кожний елемент якого генерується з деякого виразу. Можливі два варіанти виклику цієї функції:

- `makelist (expr, i, i0, i1)` – повертає список, j-й елемент якого дорівнює `ev(expr, i = j)`, при цьому індекс j міняється від $i0$ до $i1$
- `makelist (expr, x, list)` – повертає список, j-й елемент якого дорівнює `ev(expr, x = list [j])`, при цьому індекс j міняється від 1 до `length(list)`.

Приклади:

```
(%i1) makelist(concat(x,i),i,1,6);
(%o1) [x1, x2, x3, x4, x5, x6]
(%i2) list: [1,2,3,4,5,6,7];
(%o2) [1, 2, 3, 4, 5, 6, 7]
(%i3) makelist(exp(i),i,list);
(%o3) [e, e2 ,e3, e4 ,e5, e6 ,e7]
```

Багато в чому аналогічні дії виконує функція `create_list(form,` x_1`,` $list_1$`, ...,` x_n`,` $list_n$`)`.

Ця функція будує список шляхом обчислення виразу `form`, що залежить від x_i, до кожного елемента списку $list_i$ (аналогічно `form`, що залежить і від x_2, застосовується до $list_2$ тощо).

Приклад:

```
(%i1) create_list(x^i,i,[1,3,7]);
(%o1) [x,x3,x7]
(%i2) create_list([i,j],i, [a,b],j, [e,f,h]);
(%o2) [[a, e], [a, f], [a, h], [b, e], [b, f], [b, h]]
```

Функція `append` надає змогу склеювати списки. При виклику `append(`$list_1$`, ..., `$list_n$`)` повертається один список, у якому за елементами $list_1$ слідують елементи $list_2$ тощо, аж до $list_n$.

Приклад:

```
(%i1) append([1], [2,3], [4,5,6,7]);
(%o1) [1, 2, 3, 4, 5, 6, 7]
```

Створити новий список, компонуючи елементи двох списків за порядком, надає змогу функція `join(l,m)`. Новий список містить l_1, потім m_1, потім L_2, m_2 тощо.

Приклад:

```
(%i1) join([1,2,3],[10,20,30]);
(%o1) [1,10,2,20,3,30]
(%i2) join([1,2,3],[10,20,30,40]);
(%o2) [1,10,2,20,3,30]
```

Довжина отриманого списку обмежується мінімальною довжиною списків l і m.

Функція `cons(expr, list)` створює новий список, першим елементом якого буде `expr`, а інші – елементи списку `list`. Функція `endcons(expr, list)` також створює новий список, перші елементи якого – елементи списку `list`, а останній – новий елемент `expr`.

Приклад:

```
(%i1) cons(x,[1,2,3]);
(%o1) [x, 1, 2, 3]
(%i2) endcons(x,[1,2,3]);
(%o2) [1, 2, 3, x]
```

Функція `reverse` змінює порядок елементів у списку на зворотний

```
(%i5) list1:[1,2,3,x];
(%o5) [1, 2, 3, x]
(%i6) list2:reverse(list1);
(%o6) [x, 3, 2,1]
```

Функція `member(`$expr_1$`, `$expr_2$`)` повертає `true`, якщо $expr_1$ є елементом списку $expr_2$, і `false` у протилежному випадку.

Приклад:

```
(%i1) member(8, [8, 8.0, 8b0]);
(%o1) true
(%i2) member(8, [8.0, 8b0]);
(%o2) false
(%i3) member(b, [[a, b], [b, c]]);
(%o3) false
(%i4) member([b, c], [[a, b], [b, c]]);
(%o4) true
```

Функція `rest(expr)` виокремлює залишок після вилучення першого елемента списку `expr`. Можна вилучити перші n елементів, використовуючи виклик `rest(expr, n)`. Функція `last(expr)` виокремлює останній елемент списку `expr` (аналогічно `first` – перший елемент списку).

Приклади:

```
(%i1) list1:[1, 2, 3, 4, a, b];
(%o1) [1, 2, 3, 4, a, b]
(%i2) rest(list1);
(%o2) [2, 3, 4, a, b]
(%i3) rest(%);
(%o3) [3, 4, a, b]
(%i4) last(list1);
(%o4) b
(%i5) rest(list1, 3);
(%o5) [4, a, b]
```

Підсумовування і перемножування списків (як і інших виразів) здійснюється функціями `sum` і `product`. Функція `sum(expr, i, in, ik)` підсумує значення виразу `expr` при зміні індексу `i` від `in` до `ik`. Функція `product(expr, i, in, ik)` перемножує значення виразу expr при зміні індексу `i` від `in` до `ik`.

Приклад:

```
(%i1) product(x + i*(i+1)/2, i, 1, 4);
(%o1) (x + 1) (x + 3) (x + 6) (x + 10)
(%i2) sum(x + i*(i+1)/2, i, 1, 4);
(%o2) 4 x + 20
(%i3) product (i^2, i, 1, 4);
(%o3) 576
(%i4) sum(i^2, i, 1, 4);
(%o4) 30
```

2.7.1.2 Функції, що оперують із елементами списків

Функція `map(`f`, `$expr_1$`,..., `$expr_n$`)` надає змогу застосувати функцію (оператор, символ операції) f до частин виразів $expr_1$, $expr_2$, ..., $expr_n$. При використанні зі списками застосовує f до кожного елемента списку. Варто звернути увагу, що f – саме назва функції (без зазначення змінних, від яких вона залежить).

Приклади:

```
(%i1) map(ratsimp, x/(x^2+x)+(y^2+y)/y);
(%o1)
```

$$y + \frac{1}{x+1} + 1$$

```
(%i2) map("=",[a,b],[-0.5,3]);
(%o2) [a = -0.5, b = 3]
(%i3) map(exp,[0,1,2,3,4,5]);
(%o3)
```

$$[1, e, e^2, e^3, e^4, e^5]$$

Функція f може бути і заданої користувачем, наприклад:

```
(%i5) f(x):=x^2;
(%o5)
```

$$f(x) := x^2$$

```
(%i6) map(f,[1,2,3,4,5]);
(%o6) [1, 4, 9, 16, 25]
```

Функція `apply` застосовує задану функцію до всього списку (список стає списком аргументів функції; при виклику (`F, [`x_1`, ..., `x_n`]` обчислюється вираз `F(`arg_1`, ..., `arg_n`)`). Варто враховувати, що `apply` не розпізнає ординарні функції і функції від масиву.

Приклад:

```
(%i1) L : [1, 5, -10.2, 4, 3];
(%o1) [1, 5, -10.2, 4, 3]
(%i2) apply(max,L);
(%o2) 5
(%i3) apply(min,L);
(%o3) -10.2
```

Щоб знайти максимальний або мінімальний елемент набору чисел, треба викликати функції `max` або `min`. Однак, обидві функції як аргумент очікують кілька чисел, а не список, складений із чисел. Застосовувати подібні функції до списків і надає змогу функція `apply`.

2.7.2 Масиви

Масиви в **Maxima** – сукупності однотипних об'єктів з індексами. Число індексів не повинне перевищувати п'яти. В **Maxima** існують і функції з індексами (функції масиву).

Можливе створення і використання змінних з індексами до оголошення відповідного масиву. Такі змінні розглядаються як елементи масивів з невизначеними розмірностями (так звані хеш-масиви). Розміри невизначених масивів ростуть динамічно в міру присвоювання значень елементам. Цікаво, що індекси масивів з невизначеними границями не обов'язково повинні бути числами. Для підвищення ефективності обчислень рекомендується перетворювати масиви з невизначеними границями у звичайні масиви (для цього використовується функція `array`).

Створення масиву виконується функцією `array`. Синтаксис виклику функції: `array(name, `dim_1`, ..., `dim_n`)` – створення масиву з назвою `name` і розмірностями dim_1, ..., dim_n;

`array(name, type, `dim_1`, ..., `dim_n`)` – створення масиву з назвою `name` і елементами типу `type`; `array([`$name_1$`, ..., `$name_n$`], `dim_1`, ..., `dim_n`)` – створення декількох масивів однакової розмірності.

Індекси звичайного масиву – цілі числа, що змінюються від 0 до dim_i. **Приклад:**

```
(%i1) array(a, 1,1);
(%o1) a
(%i2) a[0,0]:0; a[0,1]:1; a[1,0]:2; a[1,1]:3;
(%o5) 0123
(%i6) listarray(a);
(%o6) [0, 1, 2, 3]
```

Функція `listarray`, використана у прикладі, перетворює масив у список. Синтаксис виклику: `listarray(A)`.

Аргумент `A` може бути визначеним або невизначеним масивом, функцією масиву або функцією з індексами. Порядок включення елементів масиву до списку – за рядками.

Функція `arrayinfo` виводить інформацію про масив `A`. Синтаксис виклику: `arrayinfo(A)`. Аргумент `A`, як і у випадку `listarray`, може бути визначеним або невизначеним масивом, функцією масиву або функцією з індексами.

Приклад використання:

```
(%i1) array(aa, 2, 3);
(%o1) aa
(%i2) aa[2,3]:%pi;
(%o2) π
(%i3) aa[1,2]:%e;
(%o3) e
```

```
(%i4) arrayinfo (aa);
(%o4) [declared, 2, [2, 3]]
(%i5) bb[FOO]:(a + b)^2;
(%o5) (b + a)^2
(%i6) bb[BAR]:(c - d)^3;
(%o6) (c - d)^3
(%i7) arrayinfo(bb);
(%o7) [hashed, 1, [BAR], [FOO]]
(%i8) listarray(bb);
(%o8) [(c - d)^3, (b + a)^2]
```

Функції `listarray` і `arrayinfo` застосовні і до функцій масиву:

```
(%i9) cc[x,y]:=y/x;
(%o9) cc_{x,y} := y
(%i10) cc[1,2];
(%o10) 2
(%i11) cc[2,1];
(%o11) 1/2
(%i12) arrayinfo(cc);
(%o12) [hashed, 2, [1, 2], [2, 1]]
(%i13) listarray(cc);
(%o13) [2, 1/2]
```

Ще один приклад – створення і виведення інформації щодо функції з індексами:

```
(%i1) dd[x](y):=y^x;
(%o1) dd_x(y) := y^x
(%i2) dd[1](4) ;
(%o2) 4
(%i3) dd[a+b];
(%o3) lambda([y], y^(b+a))
(%i4) arrayinfo(dd);
(%o4) [hashed, 1, [1], [b + a]]
(%i5) listarray(dd);
(%o5) [lambda([y], y), lambda([y], y^(b+a))]
```

Функція `make_array (type,` dim_1`, ...,` dim_n`)` створює і повертає масив **Lisp**. Тип масиву може бути `any`, `flonum`, `fixnum`, `hashed`, `functional`. Індекс i може змінюватися в межах від 0 до $dim_i - 1$.

Перевага `make_array` у порівнянні з `array` – можливість динамічно керувати розподілом пам'яті для масивів. Присвоювання `y:make_array(...)` створює посилання на масив. Коли масив більше не потрібний, посилання знищується присвоюванням `y:false`, пам'ять звільняється потім збирачем сміття.

Приклади:

```
(%i1) A1:make_array (fixnum, 8);
(%o1) Lisp Array: #(0 0 0 0 0 0 0 0)
(%i2) A1[1]:8;
(%o2) 8
(%i3) A3:make_array (any, 8);
(%o3) Lisp Array: #(NIL NIL NIL NIL NIL NIL NIL NIL) (%i4) arrayinfo(A3);
(%o4) [declared, 1, [7]]
```

Змінна `arrays` містить список назв масивів першого і другого типів, визначених на даний момент.

Приклад:

```
(%i1) array(a, 1,1);
(%o1) a
(%i2) array(b,2,3);
(%o2) b
(%i3) arrays;
(%o3) [a, b]
```

Функція `fillarray` надає змогу заповнювати масиви значеннями з іншого масиву або списку. Заповнення виконується за рядками.

Приклади:

```
(%i1) array(a, 1,1);
(%o1) a
(%i2) fillarray(a,[1,2,3,4]);
(%o2) a
(%i3) a[1,1];
(%o3) 4
(%i4) a2: make_array(fixnum, 8);
(%o4) Lisp Array #(00000000)
(%i5) fillarray(a2, [1, 2, 3, 4, 5]);
(%o5) Lisp Array #(1 2 3 4 5 5 5 5)
```

Як видно з розглянутих прикладів, довжина списку може і не збігатися з розмірністю масиву. Якщо зазначено тип масиву, він повинен заповнюватися елементами того ж типу. Вилучення масивів з пам'яті здійснюється функцією `remarray`.

Крім того, для зміни розмірності масиву передбачено функцію `rarray(A,` dim_1`, ..., `dim_n`)`. Новий масив заповнюється елементами старого за рядками. Якщо розмір старого масиву менше, ніж нового, залишок нового заповнюється нулями або `false` (залежно від типу масиву).

2.7.3 Матриці і найпростіші операції з ними

У **Maxima** визначені прямокутні матриці.

Основний спосіб створення матриць – використання функції `matrix`. Синтаксис виклику функції: `matrix(`row_1`, ..., `row_n`)`. Кожний рядок список виразів, всі рядки однакової довжини. На множині матриць визначені операції додавання, віднімання, множення і ділення. Ці операції виконуються поелементно, якщо операнди – дві матриці, скаляр і матриця або матриця і скаляр. Піднесення до степеня можливе, якщо один з операндів – скаляр. Перемножування матриць (у загальному випадку некомутативна операція) позначається символом ".". Операція множення матриці самої на себе може розглядатися як піднесення до степеня. Піднесення до степеня -1 – як обернення (якщо це можливо).

Приклад створення двох матриць:

```
(%i1) x: matrix ([17, 3], [-8, 11]);
(%o1)
```

$$\begin{bmatrix} 17 & 3 \\ -8 & 11 \end{bmatrix}$$

```
(%i2) y: matrix ([%pi, %e], [a, b]);
(%o2)
```

$$\begin{bmatrix} \pi & e \\ a & b \end{bmatrix}$$

Виконання арифметичних операцій з матрицями:

```
(%i3) x+y;
(%o3)
```

$$\begin{bmatrix} \pi+17 & 3 \\ a-8 & b+11 \end{bmatrix}$$

```
(%i4) x-y;
(%o4)
```

$$\begin{bmatrix} 17-\pi & 3-e \\ -a-8 & 11-b \end{bmatrix}$$

```
(%i5) x*y;
(%o5)
```

$$\begin{bmatrix} 17\pi & 3e \\ -a8 & 11b \end{bmatrix}$$

```
(%i6) x/y;
(%o6)
```

$$\begin{bmatrix} \dfrac{17}{\pi} & 3e^{-1} \\ -\dfrac{8}{a} & \dfrac{11}{b} \end{bmatrix}$$

Зверніть увагу – операції виконуються поелементно. При спробі виконувати арифметичні операції, як представлено вище, над матрицями різних розмірів, видається помилка.

Приклад операцій з матрицями і скалярами:

```
(%i9) x^3;
(%o9)
```

$$\begin{bmatrix} 4913 & 27 \\ -512 & 1331 \end{bmatrix}$$

```
(%i10) 3^x;
(%o10)
```

$$\begin{bmatrix} 129140163 & 27 \\ \frac{1}{6561} & 177147 \end{bmatrix}$$

Множення матриці на матрицю:

```
(%i11) x.y;
(%o11)
```

$$\begin{bmatrix} 3a+17\pi & 3b+17e \\ 11a-8\pi & 11b-8e \end{bmatrix}$$

```
(%i12) x.y;
(%o12)
```

$$\begin{bmatrix} 17\pi-8e & 3\pi+11e \\ 17a-8b & 11b+3a \end{bmatrix}$$

Очевидно, що для успішного перемножування матриці повинні бути узгоджені за розмірами. Піднесення до степеня -1 дає обернену матрицю:

```
(%i13) x^^-1;
(%o13)
```

$$\begin{bmatrix} \frac{11}{211} & -\frac{3}{211} \\ \frac{8}{211} & \frac{17}{211} \end{bmatrix}$$

```
(%i14) x.(x^^-1);
(%o14)
```

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Варто зауважити, що операції `x^^-1` і `x^-1` дають різний результат!

Приклад:

```
(%i2) x^-1;
(%o2)
```

$$\begin{bmatrix} \frac{1}{17} & \frac{1}{3} \\ -\frac{1}{8} & \frac{1}{11} \end{bmatrix}$$

```
(%i3) x^^-1;
(%o3)
```

$$\begin{bmatrix} \frac{11}{211} & -\frac{3}{211} \\ \frac{8}{211} & \frac{17}{211} \end{bmatrix}$$

Функція `genmatrix` повертає матрицю заданої розмірності, складену з елементів двохіндексного масиву. Синтаксис виклику:

`genmatrix(a,` i_2`,` j_2`,` i_1`,` j_1`)`
`genmatrix(a,` i_2`,` j_2`,` i_1`)`
`genmatrix(a,` i_2`,` j_2`)`

Індекси i_1, j_i і i_2, j_2 вказують лівий і правий нижній елементи матриці у вихідному масиві.

Приклад:

```
(%i1) h[i,j]:=1/(i+j-1);
(%o1)
```

$$h_{i,j} := -\frac{1}{i+j-1}$$

```
(%i2) genmatrix(h,3,3);
(%o2)
```

$$\begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \end{bmatrix}$$

```
(%i3) array(a,fixnum,2,2);
(%o3) a
(%i4) a[1,1]:%e;
(%o4) e
(%i5) a[2,2]:%pi;
(%o5) π
(%i6) genmatrix (a, 2, 2);
(%o6)
```

$$\begin{bmatrix} e & 0 \\ 0 & \pi \end{bmatrix}$$

Функція `zeromatrix` повертає матрицю з нулів заданої розмірності (синтаксис виклику – `zeromatrix(m,n)`).

```
(%i7) zeromatrix(2,2);
(%o7)
```

$$\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

Функція `ident` повертає одиничну матрицю заданої розмірності (синтаксис `ident(n)`)

```
(%i9) ident(2);
(%o9)
```

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Функція `copymatrix(M)` створює копію матриці `M`. Зверніть увагу, що присвоювання не створює копії матриці (як і присвоювання не створює копії списку).

Приклад:

```
(%i1) a:matrix([1,2],[3,4]);
(%o1)
```

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

```
(%i2) b:a;
(%o2)
```

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

```
(%i3) b[2,2]:10;
(%o3) 10
(%i4) a;
(%o4)
```

$$\begin{bmatrix} 1 & 2 \\ 3 & 10 \end{bmatrix}$$

Присвоювання нового значення елементу матриці `b` змінює і значення відповідного елемента матриці `a`. Використання `copymatrix` надає змогу уникнути цього ефекту.

Функції `row` і `col` надають змогу видобути відповідно рядок і стовпець заданої матриці. Результатом є список. Синтаксис виклику:

`row(M, i)` – повертає `i`-й рядок;

`col(M,i)` – повертає `i`-й стовпець.

Функції `addrow` і `addcol` додають до матриці рядок або стовпець відповідно. Синтаксис виклику:

`addcol(M,` $list_1$`, ...,` $list_n$`)`

`addrow(M,` $list_1$`, ...,` $list_n$`)`

Тут $list_1$, ..., $list_n$ – рядки або стовпчики, які додаються.

Приклад:

```
(%i1) a:matrix([1,2],[3,4]);
(%o1)
```

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

```
(%i2) b:addrow(a,[10,20]);
(%o2)
```

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 10 & 20 \end{bmatrix}$$

```
(%i3) addcol(b,[x,y,z]);
(%o3)
```

$$\begin{bmatrix} 1 & 2 & x \\ 3 & 4 & y \\ 10 & 20 & z \end{bmatrix}$$

Функція `submatrix` повертає нову матрицю, що складається з підматриці заданої. Синтаксис виклику:
`submatrix(`i_1`, ..., `i_m`, `M`, `j_1`, ..., `j_n`)`
`submatrix(`i_1`, ..., `i_m`, `M`)`
`submatrix(`M`, `j_1`, ..., `j_n`)`
Підматриця будується у такий спосіб: з матриці M віддаляються рядки i_1, ..., i_m і j_1, ..., j_n.
Приклад (використаємо останній результат з попереднього прикладу, видаляємо третій рядок і третій стовпець):

```
(%i6) submatrix(3,%,3);
(%o6)
```

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

Для заповнення матриці значеннями деякої функції використовується функція `matrixmap` (аналог `map`, `apply`, `fullmap`). Синтаксис виклику: `matrixmap(f, M)`. Функція matrixmap повертає матрицю з елементами i, j, рівними `f(M[`i`,`j`])`.
Приклад:

```
(%i1) a:matrix([1,2],[3,4]);
(%o1)
```

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

```
(%i2) f(x):=x^2;
(%o2) f(x) := x^2
(%i3) matrixmap(f,a);
(%o3)
```

$$\begin{bmatrix} 1 & 4 \\ 9 & 16 \end{bmatrix}$$

Для роботи з матрицями існує ще багато функцій, але вони належать до розв'язання різних задач лінійної алгебри, тому обговорюються нижче, у главі 4.2.1.

2.7.4 Математичні функції

У **Maxima** є досить великий набір вбудованих математичних функцій. Перелік основних класів вбудованих функцій наведений нижче:

- тригонометричні функції: `sin` (синус), `cos` (косинус), `tan` (тангенс), `cot` (котангенс);
- зворотні тригонометричні функції: `asin` (арксинус), `acos` (арккосинус), `atan` (арктангенс), `acot` (арккотангенс);
- `sec` (секанс, $\sec(x) = \frac{1}{\cos(x)}$), `csc` (косеканс, $\csc(x) = \frac{1}{\sin(x)}$);
- `sinh` (гіперболічний синус), `cosh` (гіперболічний косинус), `tanh` (гіперболічний тангенс), `coth` (гіперболічний котангенс), `sech` (гіперболічний секанс), `cosh` (гіперболічний косеканс)
- `log` (натуральний логарифм);
- `sqrt` (квадратний корінь);

- `mod` (остача від ділення);
- `abs` (модуль);
- `min`(x_1, ..., x_n) і `max`(x_1, ..., x_n) – знаходження мінімального і максимального значення в списку аргументів;
- `sign` (визначає знак аргументу: `pos` – додатний, `neg` – від'ємний, `pnz` – не визначений, `zero` – значення дорівнює нулю);
- Спеціальні функції – функції Бесселя, гамма-функція, гіпергеометрична функція тощо;
- Еліптичні функції різних типів.

2.7.5 Обчислення і перетворення аналітичних виразів

Функція `ev` є основною функцією, що обробляє вираз. Синтаксис виклику: `ev(expr,` arg_1`, ...,` arg_n`)`

Функція `ev` обчислює вираз `expr` у середовищі, обумовленому аргументами arg_1, ..., arg_n. Аргументи можуть бути ключами (булевими прапорцями, присвоюваннями, рівняннями і функціями. Функція `ev` повертає результат (інший вираз).

У багатьох випадках можна опускати назву функції `ev` (тобто застосовувати значення змінних до деякого виразу)

$$expr, flag1, flag2, \ldots expr, x = val1, y = val2, \ldots expr, flag1, x = val1, y = val2, flag2, \ldots$$

На вираз `expr` типово діє функція спрощення. Необхідність виконання спрощення регулюється прапорцем `simp` (якщо встановити `simp = false`, спрощення буде вимкнено). Крім того, використовують прапорці `float` і `numer`, що визначають формат подання раціональних чисел (у вигляді дробів або із рухомою крапкою) і результатів обчислення математичних функцій. Прапорець `pred` визначає необхідність обчислення для логічних виразів.

Аргументами `ev` можуть бути і вбудовані функції, що виконують спрощення або перетворення виразів (`expand`, `factor`, `trigexpand`, `trigreduce`) або функція `diff`.

Якщо зазначені підставляння (у вигляді `x=val1` або `x:val2`), то вони виконуються.

При цьому повторний виклик функції `ev` цілком здатний ще раз змінити вираз, тобто обробка виразу не виконується до кінця при однократному виклику функції `ev`.

Приклад:

```
(%i1) ev((a+b)^2,expand);
(%o1) b^2 + 2ab + a^2
(%i2) ev((a+b)^2,a=x);
(%o2) (x+b)^2
(%i3) ev((a+b)^2,a=x,expand,b=7);
(%o3) x^2 + 14 x + 49
```

Інший приклад показує застосування `diff` до відкладеного обчислення похідної:

```
(%i1) sin(x) + cos(y) + (w+1)^2 + 'diff (sin(w), w);
(%o1) cos(y) + sin(x) + d/dw sin(w) + (w + 1)^2
(%i2) ev (%, sin, expand, diff, x=2, y=1);
(%o2) cos(w) + w^2 + 2w + cos (1) + 1.909297426825682
```

Прапорець `simp` надає змогу або забороняє спрощення виразів. Типово він дорівнює `true`, якщо встановити його рівним `false`, то спрощення виконуватися не будуть:

```
(%i1) f:a+2*a+3*a+4*a;
(%o1) 10 a
(%i2) simp:false;
(%o2) false
(%i3) f:a+2*a+3*a+4*a;
(%o3) a + 2 a + 3 a + 4 a
```

Функцію `ev` не обов'язково вказувати явно, наприклад:

```
(%i3) x+y, x: a+y, y: 2;
(%o3) y + a + 2
```

Оператор, примусового обчислення, позначений двома апострофами, є синонімом до функції `ev(вираз)`. Сама функція `ev` надає набагато ширші можливості, ніж просте примусове обчислення заданого виразу: вона може приймати довільну кількість аргументів, перший з яких – вираз, що обчислюється, а інші – спеціальні параметри, які саме і впливають на те, як саме виконуються обчислення.

У термінології **Maxima** необчислена форма виразу називається «noun form», обчислена – «verb form». Зберігаючи лінгвістичні паралелі, українською це можна перекласти як «недоконана форма» і «доконана форма». Значення виразу, що вводиться, у **Maxima** закономірно зберігається до його обчислення (тобто у недоконаній формі), а значення виведеного виразу – після (тобто у доконаній); інакше кажучи, маємо природний порядок «введення – обчислення – виведення».

Функція `factor` факторизує (тобто представляє у вигляді добутку деяких співмножників) заданий вираз (функція `gfactor` – аналогічно, але на множині комплексних чисел і виразів).

Приклад:

```
(%i1) x^3-1,factor;
(%o1) (x - 1) (x^2 + x + 1)
(%i2) factor(x^3-1);
(%o2) (x - 1)(x^2 + x + 1)
```

Ще приклади факторизації різних виразів:

```
(%i3) factor(-8*y - 4*x + z^2*(2*y + x));
(%o3) (2y + x)(z - 2)(z + 2)
(%i4) factor(2^63 - 1);
(%o4) 7^2 73 127 337 92737 649657
(%i5) factor(1 + %e^(3*x));
(%o5) (e^x + 1)(e^(2x) - e^x + 1)
```

Приклад використання функції `gfactor`:

```
(%i6) gfactor(x^2+a^2);
(%o6) (x - ia)(x + ia)
(%i7) gfactor(x^2+2*%i*x*a-a^2);
(%o7) (x + ia)^2
```

Функція factorsum факторизує окремі доданки у виразі.

```
(%i8) expand((x + 1)*((u + v)^2 + a*(w + z)^2));
(%o8) axz^2 + az^2 + 2awxz + 2awz + aw^2x + v^2x + 2uvx + u^2x + aw^2 + v^2 + 2uv + u^2
(%i9) factorsum(%);
(%o9) (x + 1)(a(z + w)^2 + (v + u)^2)
```

Функція gfactorsum відрізняється від `factorsum` тим же, чим `gfactor` відрізняється від `factor`:

```
(%i10) gfactorsum(a^3+3*a^2*b+3*a*b^2+b^3+x^2+2*%i*x*y-y^2 );
(%o10) (b + a)^3 - (y - ix)^2
```

Функція `expand` розкриває дужки, виконує множення, піднесення до степеня, наприклад:

```
(%i1) expand((x-a)^3);
(%o1) x^3 - 3ax^2 + 3a^2x - a^3
(%i2) expand((x-a)*(y-b)*(z-c));
(%o2) xyz - ayz - bxz + abz - cxy + acy + bcx - abc
(%i3) expand((x-a)*(y-b)^2);
(%o3) xy^2 - ay^2 - 2bxy + 2aby + b^2x - ab^2
```

Функція `combine` поєднує доданки з ідентичним знаменником

```
(%i4) combine(x/(1+x^2)+y/(1+x^2));
(%o4) (y + x)/(x^2 + 1)
```

Функція `xthru` приводить вираз до загального знаменника, не розкриваючи дужок і не намагаючись факторизувати доданки:

```
(%i5) xthru(1/(x+y)^10+1/(x+y)^12);
(%o5) ((y + x)^2 + 1)/(y + x)^12
(%i6) ((x+2)^20-2*y)/(x+y)^20+(x+y)^(-19)-x/(x+y)^20;
(%o6) 1/(y + x)^19 + ((x + 2)^20 - 2y)/(y + x)^20 - x/(y + x)^20
(%i7) xthru(%);
(%o7) ((x + 2)^20 - y)/(y + x)^12
```

Функція `multthru` множить кожний доданок у сумі на множник, причому при множенні дужки у виразі не розкриваються. Вона допускає два варіанти синтаксису:

`multthru(mult, sum);`

`multthru(expr);`

В останньому випадку вираз `expr` включає і множник і суму (див. (%i4) у прикладі нижче).

Приклад:

```
(%i1) x/(x-y)^2-1/(x-y)-f(x)/(x-y)^3;
```

$$(\%o1)\quad -\frac{1}{x-y}+\frac{x}{(x-y)^2}-\frac{f(x)}{(x-y)^3}$$

```
(%i2) multthru ((x-y)^3, %);
```

$$(\%o2)\quad -(x-y)^2+x(x-y)-f(x)$$

```
(%i3) ((a+b)^10*s^2+2*a*b*s+(a*b)^2)/(a*b*s^2);
```

$$(\%o3)\quad \frac{(b+a)^{10}s^2+2abs+a^2b^2}{abs^2}$$

```
(%i4) multthru (%);
```

$$(\%o4)\quad \frac{2}{s}+\frac{ab}{s^2}+\frac{(b+a)^{10}}{ab}$$

Функції `assume` (введення обмежень) і `forget` (зняття обмежень) надають змогу керувати умовами виконання (контекстом) інших функцій і операторів.

Приклад:

```
(%i20) sqrt(x^2);
(%o20) |x|
(%i21) assume (x<0);
(%o21) [ x<0 ]
(%i22) sqrt(x^2);
(%o22) -x
(%i23) forget(x<0);
(%o23) [ x<0 ]
(%i24) sqrt(x^2) ;
(%o24) |x|
```

Функція `divide` надає змогу обчислити частку і остачу від ділення одного многочлена на інший:

```
(%i1) divide(x^3-2,x-1);
(%o1) [x^2 + x + 1, -1]
```

Перший елемент отриманого списку – частка, другий – остача від ділення.

Функція `gcd` надає змогу знайти найбільший загальний дільник многочленів.

Підстановки виконуються функцією `subst`. Виклик цієї функції: `subst(a, b, c)` (підставляємо `a` замість `b` у виразі `c`).

Приклад:

```
(%i1) subst (a, x+y, x + (x+y)^2 + y);
(%o1) y + x + a^2
```

2.7.6 Перетворення раціональних виразів

Для виокремлення чисельника і знаменника дробових виразів використовуються функції `num` і `denom`:

```
(%i1) expr:(x^2+1)/(x^3-1);
```

$$(\%o1)\quad \frac{x^2+1}{x^3-1}$$

```
(%i2) num(expr);
(%o2) x^2 + 1
(%i3) denom(expr);
(%o3) x^3 - 1
```

Функція `rat` зводить вираз до канонічного подання. Вона спрощує будь-який вираз, розглядаючи його як дробово-раціональну функцію, тобто працює з операціями «+», «-», «*», «/» і з піднесенням до цілого степеня. Синтаксис виклику:

`rat(expr)`

`rat(expr,` x_1`, ..., `x_n`)`

Змінні впорядковуються у відповідності зі списком x_1, ..., x_n. При цьому вигляд відповіді залежить від способу упорядковування змінних. Від початку змінні впорядковані за абеткою. Приклад використання `rat`:

```
(%i1) ((x-2*y)^4/(x^2-4*y^2)^2+1)*(y+a)*(2*y+x)/(4*y^2+x^2);
```

$$(\%o1)\quad \frac{(y+a)(2y+x)\left(\dfrac{(x-2y)^4}{(x^2-4y^2)^2}+1\right)}{4y^2+x^2}$$

```
(%i2) rat(%);
```

$$(\%o2)\quad \frac{2y+2a}{2y+x}$$

Після зазначення порядку використання змінних одержуємо наступний вираз:

```
(%i3) rat(%o1,y,a,x);
```

$$(\%o3)\quad \frac{2a+2y}{x+2y}$$

Функція `ratvars` надає змогу змінити типовий порядок пріоритетності змінних за абеткою. Виклик `ratvars(z, y, x, w, v, u, t, s, r, q, p, o, n, m, l, k, j, i, h, g, f, e, d, c, b, a)` міняє пріоритетність на зворотну, а виклик `ratvars(m, n, a, b)` упорядковує змінні `m, n, a, b` у порядку зростання пріоритетності.

Прапорець `ratfac` вмикає або вимикає часткову факторизацію виразів при зведенні їх до стандартної форми (CRE). Від початку встановлене значення `false`. Якщо встановити значення `true`, буде виконуються часткова факторизація.

Функція `ratsimp` зводить всі частини (у тому числі аргументи функцій) виразу, що не є дробово-раціональною функцією, до канонічного подання, роблячи спрощення, які не виконує функція `rat`. Повторний виклик функції у загальному випадку може змінити результат, тобто не обов'язкове спрощення виконується до кінця. Застосуванням спрощення до експоненційних виразів керує прапорець `ratsimexpons`, типово рівний `false`, якщо його встановити в `true`, спрощення застосовується і до показників степеня або експоненти.

```
(%i1) sin(x/(x^2+x))=exp((log(x)+1)^2-log(x)^2);
```

$$(\%o1)\quad \sin\left(\frac{x}{x^2+x}\right)=e^{(\log(x)+1)^2-\log(x)^2}$$

```
(%i2) ratsimp(%) ;
```

$$(\%o2)\quad \sin\left(\frac{1}{x+1}\right)=ex^2$$

```
(%i3) ((x-1)^(3/2)-(x+1)*sqrt(x-1))/sqrt((x-1)*(x+1));
```

$$(\%o3)\quad (x-1)^{\frac{3}{2}}-\sqrt{x-1}(x+1)$$

```
(%i4) ratsimp(%) ;
```

$$(\%o4)\quad -\frac{2\sqrt{x-1}}{\sqrt{x^2-1}}$$

```
(%i5) x^(a+1/a), ratsimpexpons: true;
```

$$(\%o5)\quad x^{\frac{a^2+1}{a}}$$

Функція `fullratsimp` викликає функцію `ratsimp` доти, доки вираз не перестане мінятися.

Приклад:

```
(%i1) expr: (x^(a/2)+1)^2*(x^(a/2)-1)^2/(x^a-1);
```

$$(\%o1)\quad \frac{\left(x^{\frac{a}{2}}-1\right)^2\left(x^{\frac{a}{2}}+1\right)^2}{x^a-1}$$

```
(%i2) ratsimp(expr);
```

$$(\%o2)\quad \frac{x^{2a}-2x^a+1}{x^a-1}$$

```
(%i3) fullratsimp(expr);
```

$$(\%o3)\quad x^a-1$$

```
(%i4) rat(expr);
```

$$(\%o4)\quad \frac{x^{\frac{a}{2}}-2\left(x^{\frac{a}{2}}\right)^2+1}{x^a-1}$$

Приклад впливу прапорця `ratsimpexponds` на результат обчислень:

```
(%i1) fullratsimp(exp((x^(a/2)-1)^2 *(x^(a/2) + 1)^2/(x^a-1)) );
```

$$(\%o1)\quad e^{\frac{x^{2a}}{x^a-1}-\frac{2x^a}{x^a-1}+\frac{1}{x^a-1}}$$

```
(%i2) ratsimpexpons:true;
```

$$(\%o2)\quad true$$

```
(%i3) fullratsimp( exp((x^(a/2)-1)^2 *(x^(a/2) + 1)^2 / (x^a-1) ) );
```

$$(\%o3)\quad e^{x^a-1}$$

Функція `ratexpand` розкриває дужки у виразі. Відрізняється від функції `expand` тим, що приводить вираз до канонічної форми, тому відповідь може відрізнятися від результату застосування функції `expand`:

```
(%i1) ratexpand((2*x-3*y)^3);
```
(%o1) $-27y^3+54xy^2-36x^2y+8x^3$
```
(%i2) expr:(x-1)/(x+1)^2+1/(x-1);
```
(%o2) $\dfrac{x-1}{(x+1)^2}+\dfrac{1}{x-1}$
```
(%i3) expand(expr);
```
(%3) $\dfrac{x}{x^2+2x+1}-\dfrac{1}{x^2+2x+1}+\dfrac{1}{x-1}$
```
(%i4) ratexpand(expr);
```
(%o4) $\dfrac{2x^2}{x^3+x^2-x-1}+\dfrac{2}{x^3+x^2-x-1}$

Підстановка в раціональних виразах здійснюється функцією `ratsubst`. Синтаксис виклику: `ratsubst(a, b, c)` Вираз `a` підставляється замість виразу `b` у вираз `c` (`b` може бути сумою, добутком, степенем тощо).

Приклад використання `ratsubst`:

```
(%i1) ratsubst(a,x*y^2,x^4*y^3+x^4*y^8);
```
(%o1) ax^3y+a^4
```
(%i2) cos(x)^4 + cos(x)^3 + cos(x)^2 + cos(x) + 1;
```
(%o2) $\cos(x)^4+\cos(x)^3+\cos(x)^2+\cos(x)+1$
```
(%i3) ratsubst(1-sin(x)^2,cos(x)^2,%);
```
(%o3) $\sin(x)^4-3\sin(x)^2+\cos(x)\left(2-\sin(x)^2\right)+3$

2.7.7 Перетворення тригонометричних виразів

Функція `trigexpand` розкладає всі тригонометричні і гіперболічні функції від сум і добутків у комбінації відповідних функцій одиничних кутів і аргументів. Для посилення користувацького контролю один виклик `trigexpand` виконує спрощення на одному рівні. Для керування обчисленням є прапорець `trigexpand`. Від початку прапорець `trigexpand` встановлений у `false`. Якщо прапорець `trigexpand` установити у `true`, то функція `trigexpand` буде працювати доти, поки вираз не перестане мінятися.

```
(%i1) x+sin(3*x)/sin(x),trigexpand=true,expand;
```
(%o1) $-\sin(x)^2+3\cos(x)^2+x$
```
(%i2) trigexpand(sin(10*x+y));
```
(%o2) $\cos(10x)\sin(y)+\sin(10x)\cos(y)$
```
(%i3) trigexpand(sin(3*x)+cos(4*x));
```
(%o3) $\sin(x)^4-\sin(x)^3-6\cos(x)^2\sin(x)^2+3\cos(x)^2\sin(x)+\cos(x)^4$

Функція `trigreduce` згортає всі добутки тригонометричних і гіперболічних функцій у комбінації відповідні функції від сум. Функція працює не до кінця, так що повторний виклик може змінити вираз. При виклику функції у форматі `trigreduce(expr, x)` перетворення здійснюються щодо функцій `x`.

Приклади:

```
(%i8) trigreduce(cos(x)^4+cos(x)^3+cos(x)^2+cos(x)+1);
```
(%o8) $\dfrac{\cos(4x)+4\cos(2x)+3}{8}+\dfrac{\cos(3x)+3\cos(x)}{4}+\dfrac{\cos(2x)+1}{2}+\cos(x)+1$
```
(%i9) trigreduce(-sin(x)^2+3*cos(x)^2+x);
```
(%o9) $\dfrac{\cos(2x)}{2}+3\left(\dfrac{\cos(2x)}{2}+\dfrac{1}{2}\right)+x-\dfrac{1}{2}$

Функція `trigsimp` спрощує тригонометричні і гіперболічні вирази, застосовуючи до них правила $\sin^2 x+\cos^2 x=1$ і $\cosh^2 x-\sinh^2 x=1$.

Приклад:

```
(%i1) trigsimp(sin(x)^2+3*cos(x)^2);
```
(%o1) $2\cos(x)^2+1$
```
(%i2) trigsimp(sinh(x)^2+3*cosh(x)^2);
```
(%o2) $4\cosh(x)^2-1$

Функція `trigrat` (синтаксис виклику `trigrat(expr)`) приводить заданий тригонометричний вираз `expr` до канонічної спрощеної квазілінійної форми. Цей вираз розглядається як раціональний вираз, що містить `sin`, `cos`, `tan`, аргументи яких лінійні форми деяких змінних і $\frac{\pi}{n}$ (n – ціле). Завжди, коли можливо, заданий вираз лінеаризується.

Приклад:

```
(%i1) trigrat((1+sin(2*b)-cos(2*b))/sin(b));
```
(%o1) $\sin(b)+2\cos(b)$

2.7.8 Перетворення степеневих і логарифмічних виразів

Функція `radcan` спрощує вирази, що містять експоненти, логарифми і радикали, шляхом перетворення до форми, що є канонічною для широкого класу виразів. Змінні у виразі впорядковуються. Еквівалентні вирази у цьому класі не обов'язково однакові, але їхня різниця спрощується застосуванням `radcan` до нуля.

Приклади:

```
(%i1) (log(x+x^2)-log(x))^a/log(1+x)^(a/2);
```
(%o1) $\frac{\left(\log(x^2+x)-\log(x)\right)^a}{\log(x+1)^{\frac{a}{2}}}$
```
(%i2) radcan(%);
```
(%o2) $\log(x+1)^{\frac{a}{2}}$
```
(%i3) (%e^x-1)/(1+%e^(x/2));
```
(%o3) $\frac{e^x-1}{e^{\frac{x}{2}}+1}$
```
(%i4) radcan(%);
```
(%o4) $e^{\frac{x}{2}}-1$

Функція `logcontract(expr)` рекурсивно сканує вираз `expr`, перетворюючи вираз типу $a1*log(b1)+a2*log(b2)+c$ до форми $log(ratsimp(b1^{a1}*b2^{a2}))+c$.

Приклад:

```
(%i1) 2*(a*log(x)+3*b*log(y));
```
(%o1) $2(3b\log(y)+a\log(x))$
```
(%i2) logcontract(%);
```
(%o2) $b\log(y^6)+a\log(x^2)$

Якщо оголосити змінну `n` цілою (використовуючи `declare(n,integer)`), функція `logcontract` надає змогу включити цю змінну у показник степеня:

```
(%i1) declare(n,integer);
```
(%o1) *done*
```
(%i2) logcontract(3*a*n*log(x));
```
(%o2) $a\log\left(x^{3n}\right)$

2.7.9 Задані користувачем функції

Для запису функції слід вказати її назву, а потім, у круглих дужках записати через кому значення аргументів. Якщо значенням аргументу є список, то його обмежують квадратними дужками, а елементи списку також розділяються комами.

Приклад:

```
sin(x);
integrate(sin(x),x,-5,5);
plot2d([sin(x)+3,cos(x)],[x,-/pi,/pi],[y,-5,5]);
```

Користувач може задати власні функції. Для цього спочатку вказується назва функції, у дужках перераховуються назви аргументів, після знаків `:=` (двокрапка і дорівнює) слідує опис функції. Після задання функція, створена користувачем, викликається точно так, як і вбудовані функції **Maxima**.

Приклад:

```
(%i44) f(x):=x^2;
```
(%o44) $f(x):=x^2$
```
(%i45) f(3+7);
```
(%o45) 100

Не слід використовувати для функцій назви, зарезервовані для вбудованих функцій **Maxima**. Для створення функцій використовується також вбудована функція `define`, що надає змогу перетворити вираз у функцію. Синтаксис виклику `define` досить різноманітний:

`define(f(`x_1`, ..., `x_n`), expr)`
`define(f[`x_1`, ..., `x_n`], expr)`
`define(funmake(f, [`x_1`, ..., `x_n`]), expr)`
`define(arraymake(f, [`x_1`, ..., `x_n`]), expr)`
`define(ev(`$expr_1$`), `$expr_2$`)`

Варіанти виклику функції `define` розрізняються і залежать від того, який саме об'єкт створюється: ординарна функція (аргументи у круглих дужках) або масив (аргументи у квадратних дужках). Якщо перший аргумент – оператори `funmake`, `arraymake`, то функція створюється і обчислюється (аналогічно і `ev`).

Приклади:

Ординарна функція:

```
(%i1) expr:cos(y)-sin(x);
(%o1) cos(y) − sin(x)
(%i2) define(F1(x,y),expr);
(%o2) F1(x, y) := cos(y) − sin(x)
(%i3) factor(F1(a,b));
(%o3) cos(b) − sin(a)
```

Створення функції-масиву:

```
(%i1) define(G2[x,y],x.y-y.x);
(%o1) G2_{x,y} := x.y − y.x
```

Створення масиву:

```
(%i2) define(arraymake(F,[u]),cos(u)+1);
(%o2) F_u := cos(u) + 1
```

Використання функції `ev` для завдання користувацької функції:

```
(%i3) define(ev(foo(x,y)),sin(x)-cos(y));
(%o3) foo(x, y) := sin(x) − cos(y)
```

2.8 Розв'язування задач елементарної математики

2.8.1 Знаходження коренів рівнянь і систем алгебраїчних рівнянь

Розв'язування алгебраїчних рівнянь і їхніх систем здійснюється за допомогою функції `solve`. У перших квадратних дужках вказується список рівнянь через кому, у других – список змінних, через кому (або трохи спрощені форми запису):

`solve(expr, x)` – розв'язання одного рівняння щодо змінної x;

`solve(expr)` – розв'язання рівняння з однією невідомою і числовими коефіцієнтами;

`solve([`eqn_1`, ..., `eqn_n`], [`x_1`, ..., `x_n`])` – розв'язання системи рівнянь.

Приклади: Розв'язування одного рівняння з одним невідомим

```
(%i7) solve(x^2-5*x+4);
(%o7) [x = 1, x = 4]
```

Розв'язування одного рівняння у символьному вигляді:

```
(%i2) solve([x-a/x+b], [x]);
(%o2) [x = −(√(b^2+4a)+b)/2, x = (√(b^2+4a)−b)/2]
```

Розв'язування системи рівнянь у символьному вигляді:

```
(%i10) solve([x*y/(x+y)=a,x*z/(x+z)=b,y*z/(y+z)=c], [x,y,z]);
(%o10) [[x = 0, y = 0, z = 0], [x = 2abc/((b+a)c−ab), y = 2abc/((b−a) c+ab), z = −2abc/((b−a)c−ab)]]
```

В останньому прикладі розв'язків декілька, і **Maxima** видає результат у вигляді списку.

Функція `solve` застосовна і для розв'язування тригонометричних рівнянь. При цьому у випадку декількох розв'язків у тригонометричних рівняннях видається відповідне повідомлення і лише один із розв'язків.

Приклад:

```
(%i13) solve([sin(x)=0], [x]);
solve: using arc-trig functions to get a solution.
Some solutions will be lost.
(%o13) [x = 0]
```

Також **Maxima** надає змогу знаходити комплексні корені

```
(%i18) solve([x^2+x+1], [x]);
(%o18) [x = −(√3 i+1)/2, x = (√3 i−1)/2]
```

2.9 Побудова графіків і поверхонь

Для виведення графіків на екран або друку за допомогою **Maxima** існує декілька варіантів форматів і, відповідно, програм для виведення графіки, а саме:

- `openmath` (**Tcl/Tk** програма із графічним інтерфейсом користувача; елемент **xMaxima**)
- `gnuplot` (потужна утиліта для побудови графіків, обмін з **Maxima** – через канал)
- `mgnuplot` (**Tk**-інтерфейс до **gnuplot** з рудиментарним графічним інтерфейсом користувача; є частиною дистрибутиву **Maxima**)
- `wxMaxima` (вбудовані можливості оболонки до **Maxima**)

Всі варіанти інтерфейсу (крім **wxMaxima**) для побудови графіків використовують дві базових функції: `plot2d` (побудова двовимірних графіків) і `plot3d` (побудова тривимірних графіків).

При використанні **wxMaxima** крім них використовуються ще дві аналогічні команди: `wxplot2d` і `wxplot3d`. Всі команди дозволяють або вивести графік на екран, або (залежно від параметрів функції) до файла.

2.9.1 Побудова графіка явної функції $y = f(x)$

Графік функції $y = f(x)$ на відрізку $[a, b]$ можна побудувати за допомогою функції `plot2d(f(x), [x,a,b], параметри)` або `plot2d(f(x), [x,a,b], [y,c,d], параметри)`. Параметри не обов'язкові, однак, для зміни властивостей графіка їх слід задати. Параметр $[y, c, d]$ можна не задавати, тоді висота графіка вибирається типово. Побудуємо графік функції $y = \sin(x)$ на відрізку $[-4\pi, 4\pi]$.

```
(%i2) plot2d(sin(x), [x, -4*%pi, 4*%pi]);
(%i3) plot2d(sin(x), [x, -4*%pi, 4*%pi],[y,-2,2]);
```

Результати наведені на рис. 2.1, 2.2.

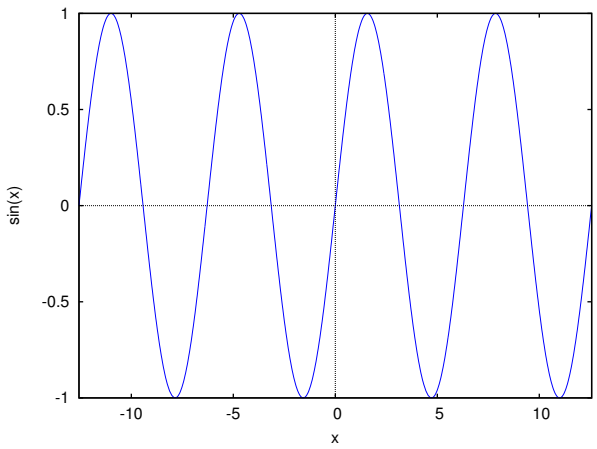


Рис. 2.1. Найпростіша команда побудови графіка

2.9.2 Побудова графіків функцій, заданих параметрично

Для побудови графіків функцій, заданих параметрично, використовується параметр `parametric`. Для побудови графіка вказується область зміни параметра. Приклад графіка найпростішої параметричної функції представлений на рисунку 2.3.

Команда побудови графіка: `plot2d([parametric,cos(t),sin(t), [t,-%pi,%pi], [nticks,80]],[x,-4/3,4/3])`

Параметр `ntics` вказує число точок, за якими проводиться крива.

Розгляньмо деякі параметри.

Параметри слід вказувати у вигляді аргументів функції `plot2d` у квадратних дужках. Можна встановити панель умовних позначень, міток на осях, кольорів і стилю графіка. Застосування декількох параметрів характеризує наступний приклад:

```
(%i17) plot2d([[discrete,xy], 2*%pi*sqrt(l/980)], [l,0,50],
[style, [points,5,2,6], [lines,1,1]],
[legend, experiment , theory ],
[xlabel,"pendulum's length (cm)"], [ylabel,"period (s)"]);
```

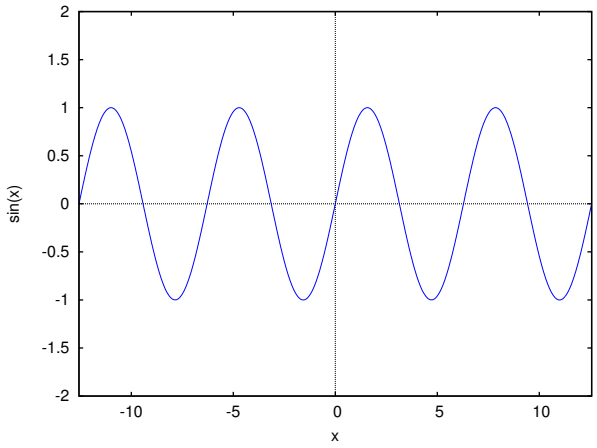


Рис. 2.2. Найпростіша команда побудови графіка із вказівкою інтервалу за віссю Oy

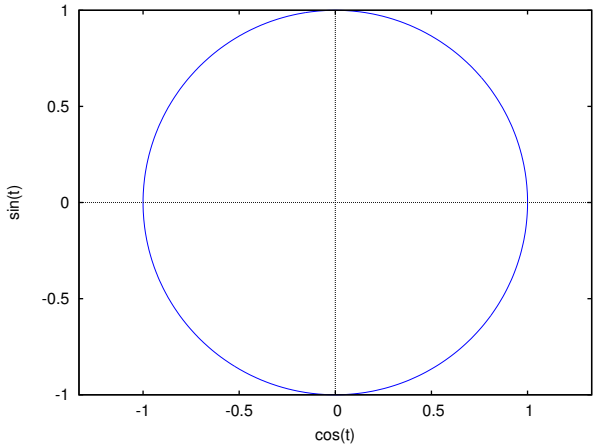


Рис. 2.3. Найпростіша команда побудови графіка функції, заданої параметрично

У даному прикладі в одних осях будуються дві графіки. Перший (`[discrete, xy]`) будується у вигляді точок за масивом `xy` із зазначенням стилю `points`. Другий будується за рівнянням функції `2*%pi*sqrt(1/980)` із зазначенням стилю `lines`. Параметр `legend` вказує підписи кривих, параметри `xlabel` і `ylabel` – підписи осей. Результат наведений на рис. 2.4.

Формування масивів для побудови графіка здійснюється у такий спосіб:

```
(%i12) xx:[10, 20, 30, 40, 50];
(%i13) yy:[.6, .9, 1.1, 1.3, 1.4];
(%i14) xy:[[10,.6], [20, . 9] , [30,1.1], [40,1.3], [50,1.4]];
```

Можна комбінувати в одних осях графіки кривих різного типу: функції $y = f(x)$ або параметричні

$$\begin{cases} x = \varphi(t), \\ y = \psi(t). \end{cases}$$

наприклад (див. Рис. 2.5):

```
plot2d([x^3+2, [parametric, cos(t), sin(t), [t, -5, 5], [nticks,80]]], [x, -2, 2],
[xlabel, "x"],[ylabel, "y"], [style, [linespoints,3,2], [lines,3,1]],
[gnuplot_term, ps], [gnuplot_out_file, "test.eps"]);
```

Параметри `[gnuplot_term, ps], [gnuplot_out_file, "test.eps"]` указують, що графічна ілюстрація виводиться до файла `test.eps` у форматі `postscript` (модуль для виведення графіків – `gnuplot`).

Параметри `[style,[linespoints,3,2],lines,3,1]]` надають змогу вказати стиль ліній на графіку (лінія із крапками або суцільна лінія).

Для виведення результатів у форматі `png` можна скористатися параметрами (вказування розмірів 400, 400 у загальному випадку необов'язкове): `[gnuplot_term, png size 400,400], [gnuplot_out_file, max.png]`

2.9.3 Побудова кривих у полярній системі координат

Для побудови графіка у полярних координатах потрібно задати зміну значень полярного радіуса і полярного кута. Нехай $r = r(f)$ $(a < f < b)$ – залежність полярного радіуса r від полярного кута f.

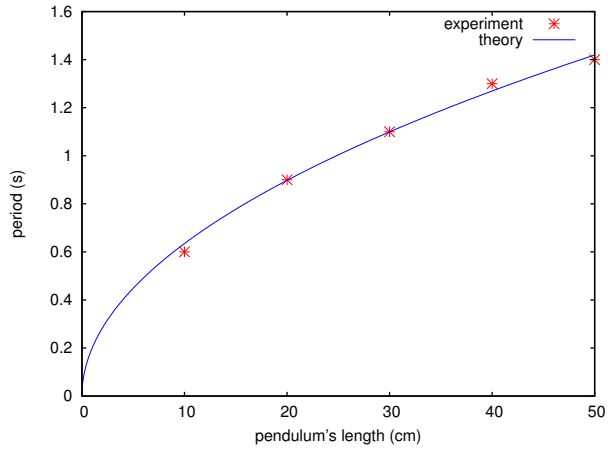


Рис. 2.4. Сполучення на одному графіку дії серії параметрів

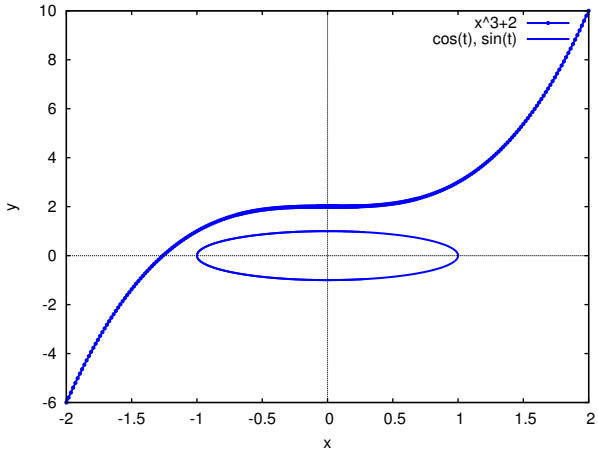


Рис. 2.5. Сполучення на одному графіку параметричної і заданої явно кривих

Приклад: побудувати у полярних координатах графік функції $r = 3(1 - \varphi + \varphi^2)$, $0 \leqslant \varphi < 2\pi$.
Для створення графіка використаємо команду:

```
(%i1) load(draw);draw2d(terminal=eps,nticks=200,xaxis=true,polar(3*(1-ph+ph^2),ph,0,(2*%pi)));
```

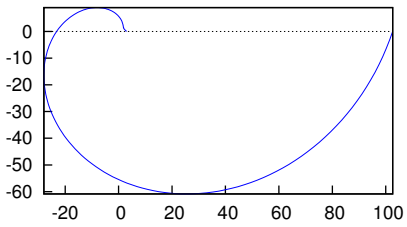


Рис. 2.6. Крива у полярних координатах

Результат наведений на рис. 2.6.
Приклад: побудувати у полярних координатах графіки трьох функцій $r = 6\cos(\varphi)$, $r = \varphi$, $r = 2\sin(\varphi)$, $0 \leqslant \varphi \leqslant 2\pi$.
Для створення графіка скористаємося такою командою:

```
(i%1) load(draw); draw2d(terminal=eps, file_name="fig7",
nticks=200, xaxis=true, polar(6*cos(ph),ph,0,(2*%pi)),
polar(ph,ph,0,(2*%pi)), polar(2*sin(ph),ph,0,(2*%pi)));
```

Результат наведений на рис. 2.7.

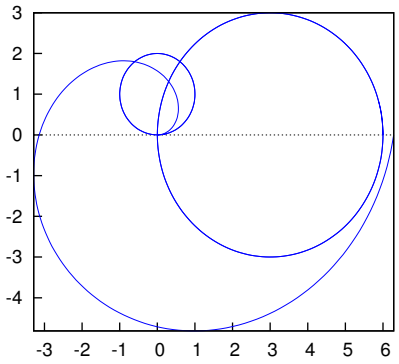


Рис. 2.7. Сполучення на одному графіку декількох полярних кривих

2.9.4 Побудова тривимірних графіків

Основна команда для побудови тривимірних графіків – `plot3d`. Розгляньмо технологію побудови графіків з використанням інтерфейсу `gnuplot`. Поверхня функції у кольоровому зображенні будується з використанням параметра `pm3d` (рис. 2.8).

Приклад:

```
(%i2) plot3d (atan (-x^2 + y^3/4), [x, -4, 4], [y, -4, 4],
[grid, 50, 50], [gnuplot\_pm3d,true],[gnuplot_term,ps], [gnuplot_out_file,"fig8.eps"]);
```

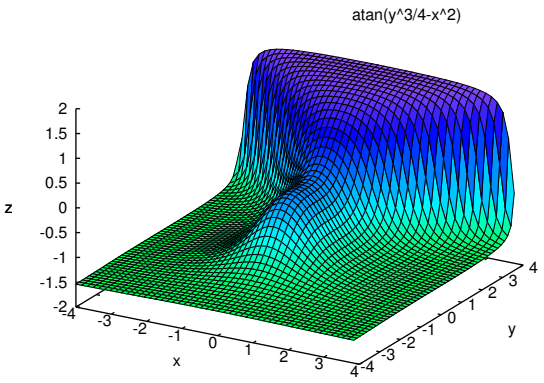


Рис. 2.8. Графік функції двох змінних з фарбуванням поверхні

З використанням цього параметра і особливостей програми **gnuplot** можна побудувати і зображення ліній рівня функції.

Приклад (рис. 2.9):

```
(%i3) plot3d(cos (-x^2 + y^3/4), [x, -4, 4], [y, -4, 4], [gnuplot_preamble,"set view map"],
[gnuplot_pm3d, true] , [grid, 150, 150] , [gnuplot_term,ps] , [gnuplot_out_file,"fig9.eps"]);
```

Строгіший результат можна одержати, використовуючи стандартний формат функції `plot3d`. Приклад (рис. 2.10):

Для виведення графіка до файла також слід використати параметри gnuplot (установити термінал `gnuplot` і назву файла результату). Відповідна команда:

```
(%i5) plot3d(2^(-u^2+v^2), [u, -3, 3], [v, -2, 2], [gnuplot_term,ps], [gnuplot_out_file,"fig10.eps"]);
```

Функція, для якої будується тривимірний графік, може задаватися як функція **Maxima** або `Lisp`-функція, лямбда-функція або вираз **Maxima** загального вигляду. При використанні формату `plot3d(f,...)` вираз `f` розглядається як функція двох змінних. При використанні формату `plot3d([`f_1`,` f_2`,` f_s`], ...)`, кожна функція (f_1, f_2, ... f_s) розглядається як функція трьох змінних.

Функція `plot3d` надає змогу будувати графіки функцій, заданих у циліндричних або сферичних координатах за рахунок використання перетворення координат (параметр `[transform_xy, polar_to_xy]` або функції `make_transform (vars, fx, fy, fz)`).

Певні переваги забезпечує формат wxplot, наявний у графічному інтерфейсі **wxMaxima** (`wxplot2d` і `wxplot3d`). Команда побудови графіка у форматі **wxMaxima** за синтаксисом мало відрізняється від синтаксису команд `plot2d` і `plot3d`. Якість відтворення графіків на екрані **wxMaxima** відносно невисока, але легко, виділивши графік клацанням кнопкою миші, зберегти його до файла (типово maxout.png). Якість копії у файлі набагато краща, ніж рисунка у вікні **wxMaxima**.

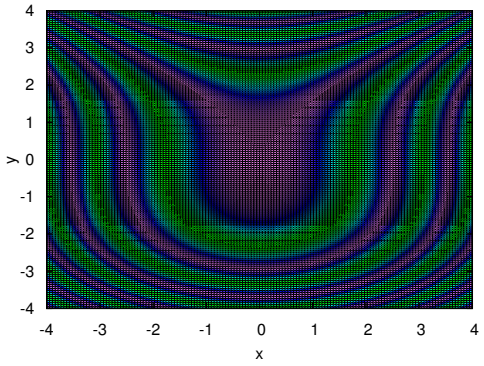


Рис. 2.9. Графік ліній рівня функції двох змінних з фарбуванням поверхні

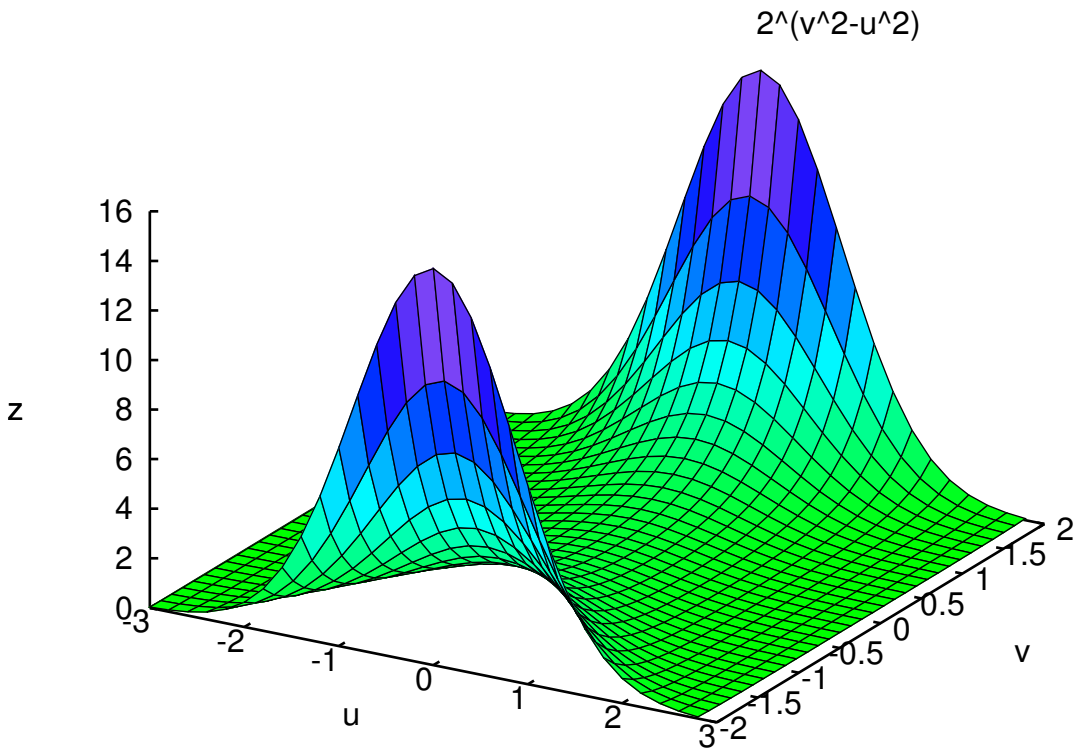


Рис. 2.10. Простий графік функції двох змінних

Розділ 3

Задачі вищої математики з Maxima

3.1 Операції з комплексними числами

3.1.1 Подання комплексних чисел

Значення цілого додатного степеня комплексного аргументу найпростіше обчислювати у тригонометричній формі. Якщо $z = x + iy = r(\cos(\varphi) + i\sin(\varphi))$ (тут $r = \sqrt{x^2 + y^2}$ – модуль комплексного числа, $\varphi = \text{arctg}\,\frac{y}{x}$ – його аргумент), то для будь-якого цілого додатного числа n має місце формула: $w = f(z) = z^n = r^n(\cos(n\varphi) + i\sin(n\varphi))$.

Коренем n-го степеня з комплексного числа z називається число $w = \sqrt[n]{z}$ таке, що $w^n = z$. Для будь-якого комплексного числа z існує n комплексних чисел w таких, що $w^n = z$. Значення кореня, тобто значення функції $f(z) = \sqrt[n]{z}$ також зручно обчислювати у тригонометричній формі. Якщо $z = x+iy = r(cos(\varphi)+isin(\varphi))$, то для будь-якого цілого додатного числа n має місце формула: $f(z) = \sqrt[n]{z} = \sqrt[n]{r}(\cos\varphi + i\sin\varphi) = \sqrt[n]{r}\left(\cos\frac{\varphi+2k\pi}{n} + i\sin\frac{\varphi+2k\pi}{n}\right)$, тобто функція $f(z) = \sqrt[n]{z}$ є багатозначною функцією – кожному значенню аргументу відповідає n різних значень кореня.

Якщо $z = x + iy = r(\cos(\varphi + i\sin(\varphi))$, то значення функції $f(z) = exp(z)$ обчислюються за формулою $f(z) = e^z = e^{x+iy} = e^x(\cos(y) + i\sin(y))$.

Логарифмом комплексного числа z називається таке число w, що $e^w = z$. Значення логарифмічної функції $F(z) = \text{Ln}(z)$ обчислюються за формулою $\text{Ln}(z) = \ln(|z|) + i\text{Arg}z = \ln(|z|) + i\text{arg}z + 2k\pi$, $k = 1, 2, \ldots$. Величину $\ln(|z|) + i\text{arg}z$ називають головним значенням логарифма. Функція $f(z) = \text{Ln}(z)$ є багатозначною функцією – кожному значенню аргументу відповідає нескінченна множина різних значень логарифма.

Комплексний вираз визначається у **Maxima** за допомогою додавання до дійсної частини виразу і добутку `%i` (уявної одиниці) і уявної частини (тобто в алгебраїчній формі). Наприклад, корені з рівняння $x^2 - 4x + 13 = 0$ рівні `2 + 3*%i` і `2 - 3*%i`. Розв'язування у **Maxima**:

```
(%i1) eq:x^2-4*x+13=0;
(%o1) x^2-4*x+13=0
(%i2) solve(eq,x);
(%o2) [x=2-3%i,x=3%i+2]
(%i3) x1:%o2[1]$ x2:%o2[2];
(%o4) x=3%i+2
(%i5) print(x1,x2);
(%o5) x=2-3%i,x=3%i+2
```

Складніший приклад обчислення кореня алгебраїчного рівняння n-го степеня:

```
(%i1) solve(x^3=1,x);
(%o1) [x=(sqrt(3)%i-1)/2,x=-(sqrt(3)%i+1)/2,x=1]
(%i2) solve(x^5=1,x);
(%o2) [x=e^(2%i*%pi/5),x=e^(4%i*%pi/5),x=e^-(4%i*%pi/5),x=e^-(2%i*%pi/5),x=1]
```

Кількість коренів, що повертає **Maxima**, відповідає основній теоремі алгебри (рівняння третього степеня має три корені, п'ятого – п'ять тощо).

Перетворення комплексних виразів може здійснюватися функціями для роботи з алгебраїчними виразами (`radcan`, `expand` тощо), але передбачений і ряд специфічних функцій, розрахованих на операції саме з комплексними числами.

3.1.2 Функції для роботи з комплексними числами

Спрощення часток, коренів і інших функцій комплексних виразів можна звичайно виконати за допомогою функцій `realpart`, `imagpart`, `rectform`, `polarform`, `cabs`, `carg`.

Обчислення модуля комплексного числа виконується функцією `cabs`. Аргумент комплексного виразу обчислюється за допомогою функції `carg`. Комплексний аргумент – θ у межах $[-\pi, \pi]$ таким чином, що $r\exp(\theta i) = z$, де r – модуль

комплексного числа z. Варто враховувати, що `carg` – обчислювальна функція, не призначена для спрощення комплексних виразів. (У деяких випадках зручно використати параметр `numer`, встановлення якого змушує представляти результати у форматі із рухомою крапкою – див. приклад нижче).

Приклад:

```
(%i1) carg(1);
(%o1) 0
(%i2) carg(1+%i);
(%o2) π/4
(%i3) carg(exp(%i)),numer;
(%o3) 1.0
(%i4) carg(exp(%pi*%i));
(%o4) π
(%i5) carg(exp(3/2*%pi*%i));
(%o5) -π/2
```

Для перетворення комплексних виразів використовують також функцію `demoivre`. Керування її роботою здійснюється прапорцем `demoivre`.

Коли змінна `demoivre` встановлена (`demoivre = true`), комплексні показникові функції перетворені до еквівалентного виразу у термінах тригонометричних функцій: $e^{a+i\cdot b}$ спрощує до вигляду $e^a \cdot (\cos(b) + i \cdot \sin(b))$, якщо вираз b не містить `%i`. Типове значення `demoivre` – `false`.

Крім того, перетворення різних форм комплексних чисел здійснюється функцією `exponentialize`, що перетворить тригонометричні і гіперболічні функції в експоненційну форму. Прапорці `demoivre` і `exponentialize` не можуть обидва бути встановлені у `true` одночасно.

Приклад:

```
(%i1) demoivre:true;
(%o1) true
(%i2) demoivre(exp(3+3/2*%pi*%i));
(%o2) -%e^3%i
(%i3) demoivre(exp(%pi+3/2*%pi*%i));
(%o3) -%e^π%i
```

Комплексно-спряжені вирази обчислюються за допомогою функції `conjugate(x)`.

Приклад:

```
(%i1) declare ([aa, bb], real, cc, complex, ii, imaginary);
(%o1) done
(%i2) conjugate (aa + bb*%i);
(%o2) aa - %ibb
(%i3) conjugate(ii);
(%o3) -ii
```

Як видно із прикладу, функція declare надає змогу оголосити тип виразів: дійсні, комплексні і суто уявні (imaginary).

Функція `plog(x)` представляє основну гілку комплексного логарифма, що відповідає $-\pi < carg(x) \leqslant +\pi$, наприклад:

```
(%i1) a:1+%i;
(%o1) %i + 1
(%i2) plog(a);
(%o2) log(2)/2 + %i%pi/4
```

Функція `polarform(expr)` повертає вираз $re^{i\theta}$, еквівалентне `expr` (параметри r і θ – дійсні).

Перетворення комплексного виразу до алгебраїчної форми виконується функцією `rectform(x)`.

Приклад:

```
(%i1) a:1+%i;
(%o1) %i + 1
(%i2) polarform(a);
(%o2) √2%e^(%i%pi/4)
(%i3) rectform(%);
(%o3) %i + 1
```

Функція `residue(expr, z, z0)` обчислює лишок у комплексній площині для виразу `expr`, коли змінна z приймає значення z_0. Лишок – коефіцієнт при $(z - z_0)^{-1}$ ряду Лорана для `expr`.

Приклад:

```
(%i1) residue(s/(s**2+a**2), s, a*%i);
```
(%o1) $\frac{1}{2}$
```
(%i2) residue (sin(a*x)/x**4, x, 0);
```
(%o2) $-\frac{a^3}{6}$

3.2 Задачі лінійної алгебри

Пакет **Maxima** включає велику кількість функцій для розв'язування різноманітних задач лінійної алгебри.

Розгляньмо основні функції, що надають змогу оперувати матрицями і розв'язувати основні задачі лінійної алгебри.

3.2.1 Найпростіші операції з матрицями

У **Maxima** на матрицях визначені звичайні операції множення на число, додавання і матричне множення. Останнє реалізується за допомогою бінарної операції «.» (крапка). Розмірності матриць-співмножників мають узгоджуватися.

Розгляньмо кілька прикладів.

Створення двох прямокутних матриць:

```
(%i1) a:matrix([1,2,3],[4,5,6]);
```
(%o1) $\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$
```
(%i2) b:matrix([2,2],[3,3],[4,4]);
```
(%o2) $\begin{pmatrix} 2 & 2 \\ 3 & 3 \\ 4 & 4 \end{pmatrix}$

Функція `transpose` транспонує матрицю:

```
(%i1) a:matrix([1,2,3]); transpose(a);
```
(%o1) $\begin{pmatrix} 1 & 2 & 3 \end{pmatrix}$

(%o2) $\begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$

Множення матриці на число:

```
(%i2) c:b*2;
```
(%o2) $\begin{pmatrix} 4 & 4 \\ 6 & 6 \\ 8 & 8 \end{pmatrix}$

Додавання матриць (природно, матриці повинні бути однакової розмірності, інакше буде виведено повідомлення щодо помилки):

```
(%i4) b+c;
```
(%o4) $\begin{pmatrix} 6 & 6 \\ 9 & 9 \\ 12 & 12 \end{pmatrix}$
```
(%i5) a+b;
fullmap: arguments must have same formal structure.
- an error. To debug this try: debugmode(true);
```

Множення матриць (у даному випадком вихідні матриці a і b узгоджені за розмірами):

```
(%i6) f:a.b;
```
(%o6) $\begin{pmatrix} 20 & 20 \\ 47 & 47 \end{pmatrix}$

Якщо матриця – лівий співмножник, то правим співмножником може бути не тільки вектор-стовпець, але і вектор-рядок і навіть список.

Maxima надає змогу також підносити матриці до степеня, але фактично ця операція застосовується до кожного елемента.

3.2.2 Обернення матриць і обчислення визначників

Для обернення матриць використовується функція `invert`.

Приклад:

```
(%i1) a:matrix([1,2],[3,4]); b:invert(a); b.a;
(%o1)
```

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

```
(%o2)
```

$$\begin{pmatrix} -2 & 1 \\ \frac{3}{2} & -\frac{1}{2} \end{pmatrix}$$

```
(%o3)
```

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

Визначник обчислюється функцією `determinant`:

```
(%i4) determinant(a);
(%o4) −2
```

3.2.3 Характеристичний поліном, власні числа і власні вектори матриці

Характеристичний поліном матриці обчислюється функцією `charpoly(M, x)` (`M` – матриця, `x` – змінна, щодо якої будується поліном).

Приклад:

```
(%i6) charpoly(a,x);
(%o6) (1 − x)(4 − x) − 6
(%i7) ratsimp(%) ;
(%o7) x^2 − 5x − 2
```

Корені характеристичного полінома є власними числами матриці.

Однак для обчислення власних чисел і власних векторів матриці звичайно використовують спеціальні функції: `eigenvalues` і `eigenvectors`.

Функція `eigenvectors` аналітично обчислює власні значення і власні вектори матриці, якщо це можливо. Вона повертає список, перший елемент якого – список власних чисел (аналогічно `eigenvalues`), а далі йдуть власні вектори, кожен з яких представлений як список своїх проекцій.

Приклад:

```
(%i1) a:matrix([1,1,1],[2,2,2],[3,3,3]);
(%o1)
```

$$\begin{pmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \end{pmatrix}$$

```
(%i2) eigenvalues(a);
(%o2) [[0, 6], [2, 1]]
(%i3) eigenvectors(a);
(%o3) [[[0, 6], [2, 1]], [[[1, 0, −1], [0, 1, −1]], [[1, 2, 3]]]]
```

Функція `uniteigenvectors` відрізняється від функції `eigenvectors` тим, що повертає нормовані на одиницю власні вектори.

3.2.4 Ортогоналізація

Maxima включає спеціальну функцію для обчислення ортонормованого набору векторів із заданого. Використається стандартний алгоритм Грама-Шмідта.

Синтаксис виклику: `gramschmidt(x)` або `gschmidt(x)`.

Аргумент функції – матриця або список. Як компоненти системи векторів, на базі якої будується ортонормувана система, розглядаються рядки матриці `x` або підсписки списку `x`. Для використання даної функції необхідно явно завантажити пакет `eigen`.

Приклад:

```
(%i1) load("eigen");
(%o1) /usr/share/maxima/5.35.1/share/matrix/eigen.mac
(%i2) x:matrix([1,2,3],[4,5,6]);
(%o2)
```

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$

```
(%i3) y:gramschmidt(x);
(%o3)
```

$$[[1,2,3],[\frac{2\hat{}2\,3}{7},\frac{3}{7},-\frac{2\,3}{7}]]$$

```
(%i4) ratsimp(%[1].%[2]);
(%o4)
```

$$0$$

3.2.5 Перетворення матриці до трикутної форми

Перетворення матриці до трикутної форми здійснюється методом виключення Гауса за допомогою вбудованої функції `echelon(M)` (аналогічний результат дає функція `triangularize(M)`):

```
(%i1) a:matrix([1,2,3],[4,5,x],[6,7,y]);
```

Відмінності розглянутих функцій у тім, що `echelon` нормує діагональний елемент на 1, а `triangularize` – немає. Обидві функції використовують алгоритм виключення Гауса.

```
(%i1) x:matrix([1,2,3],[4,5,6]);
(%o1)
```

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & x \\ 6 & 7 & y \end{pmatrix}$$

```
(%i2) b:echelon(a);
(%o2)
```

$$\begin{pmatrix} 1 & 2 & 3 \\ 0 & 1 & -\frac{x-12}{3} \\ 0 & 0 & 1 \end{pmatrix}$$

3.2.6 Обчислення рангу і мінорів матриці

Для обчислення рангу матриці (порядку найбільшого невиродженого мінору матриці) використовується функція `rank`.

Приклад:

```
(%i1) a:matrix([1,2,3,4],[2,5,6,9]);
```

Матриця `a` – невироджена (два рядки, ранг дорівнює 2). Обчислимо ранг виродженої матриці, що містить лінійно залежні рядки.

```
(%i1) a:matrix([1,2,3,4],[2,5,6,9]);
(%o1)
```

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 5 & 6 & 9 \end{pmatrix}$$

```
(%i2) rank(a);
(%o2)
```

$$2$$

```
(%i3) b:matrix( [1,1] , [2,2] , [3,3] , [4,5]);
(%o3)
```

$$\begin{pmatrix} 1 & 1 \\ 2 & 2 \\ 3 & 3 \\ 4 & 5 \end{pmatrix}$$

```
(%i4) rank(b);
(%o4)
```

$$2$$

Мінор матриці обчислюється за допомогою функції `minor(M, i,j)`, де `M` – матриця, `i,j` – індекси елемента, для якого обчислюється мінор.

3.2.7 Розв'язування матричних рівнянь

Нехай дане матричне рівняння $AX = B$, де A – квадратна матриця розмірності n; B – матриця розмірності $n \times k$; X – невідома матриця розмірності $n \times k$. Нехай A – невироджена матриця (тобто $\det(A) = 0$, тоді існує єдине розв'язання цього рівняння. Розв'язок можна знайти за формулою $X = A^{-1}B$.

Приклад: Знайти розв'язок матричного рівняння $AX = B$, де

$A = \begin{pmatrix} 1 & 2 & 2 \\ -1 & -1 & 3 \\ 2 & 5 & 0 \end{pmatrix}$, $B = \begin{pmatrix} 10 & 0 \\ -2 & 5 \\ 1 & 4 \end{pmatrix}$

Спочатку задамо матриці A і B:

```
(%i1) A:matrix( [1, 2, 2],[ -1, -1, 3], [2, 5, 0]);
(%o1)
```

$$\begin{pmatrix} 1 & 2 & 2 \\ -1 & -1 & 3 \\ 2 & 5 & 0 \end{pmatrix}$$

```
(%i2) B:matrix( [10, 0],[-2, 5], [1, 4]);
(%o2)
```

$$\begin{pmatrix} 10 & 0 \\ -2 & 5 \\ 1 & 4 \end{pmatrix}$$

Перевіримо існування і одиничність розв'язання:

```
(%i3) determinant(A);
(%o3) −9
```

Матриця A невироджена, отже, розв'язок існує і єдиний. Знайдемо його:

```
(%i4) A1:invert(A); x:A1.B;
(%o4)
```

$$\begin{pmatrix} \frac{5}{3} & -\frac{10}{9} & -\frac{8}{9} \\ -\frac{2}{3} & \frac{4}{9} & \frac{5}{9} \\ \frac{1}{3} & \frac{1}{9} & -\frac{1}{9} \end{pmatrix}$$

```
(%o5)
```

$$\begin{pmatrix} 18 & -\frac{82}{9} \\ -7 & \frac{40}{9} \\ 3 & \frac{1}{9} \end{pmatrix}$$

Виконаємо перевірку:

```
(%i6) A.x-B;
(%o6)
```

$$\begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix}$$

Аналогічно розв'язується матричне рівняння $XA = B$, де A – квадратна матриця розмірності n, B – матриця розмірності $k \times n$, X – невідома матриця розмірності $k \times n$. Якщо A – невироджена матриця, то існує єдиний розв'язок $X = BA^{-1}$.

Приклад: Знайти розв'язок X матричного рівняння $XA = C$, де матриця A з попередньої задачі, C – задана матриця. Аналогічно попередньому прикладу, обчислюємо розв'язок:

```
(%i10) C:matrix([10,0,-2],[5,1,4]); x:C.A1; x.A-C;
(%o10)
```

$$\begin{pmatrix} 10 & 0 & -2 \\ 5 & 1 & 4 \end{pmatrix}$$

(%o11)

$$\begin{pmatrix} 16 & -\frac{34}{3} & -\frac{26}{3} \\ 9 & -\frac{14}{3} & -\frac{13}{3} \end{pmatrix}$$

(%o12)

$$\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

У загальному випадку (коли A – вироджена матриця, або A – не квадратна матриця) матричне рівняння $AX = B$ можна розв'язати за допомогою функції `solve`.

Синтаксис виклику: `solve([`eq_1`, `eq_2`, ..., `eq_n`], [`x_1`, `x_2`, ..., `x_m`])`, де `[`eq_1`, `eq_2`, ..., `eq_n`]` – список рівнянь, `[`x_1`, `x_2`, ..., `x_m`]` – список невідомих, щодо яких здійснюється розв'язання.

3.2.8 Спеціальні функції для розв'язання систем лінійних і поліноміальних рівнянь

Функція `linsolve([`$expr_1$`, `$expr_2$`, ..., `$expr_m$`], [`x_1`, `x_2`, ..., `x_n`])` розв'язує список лінійних рівнянь [$expr_1$, $expr_2$, ..., $expr_m$] відносно списку змінних [x_1, x_2, ..., x_n].

Вирази [$expr_1$, $expr_2$, ..., $expr_m$] можуть бути поліномами зазначених змінних і представлятися у вигляді рівнянь.

Приклад: Розв'язати систему лінійних рівнянь

$$\begin{cases} x + y + z + t = 6, \\ 2x - 2y + z + 3t = 2, \\ 3x - y + 2z - t = 8. \end{cases}$$

Розв'язування у **Maxima**:

```
(%i1) ex1:x+y+z+t=6;
ex2:2*x-2*y+z+3*t=2;
ex3:3*x-y+2*z-t=8;
linsolve([ex1,ex2,ex3],[x,y,z,t]);
(%o1)
```

$$z + y + x + t = 6$$

(%o2)

$$z - 2\,y + 2\,x + 3t = 2$$

(%o3)

$$2\,z - y + 3\,x - t = 8$$

(%o4)

$$[x = -\frac{3\%r1 - 14}{4},\ y = -\frac{\%r1 - 10}{4},\ z = \%r1,\ t = 0]$$

Таким чином, загальний розв'язок має вигляд: $x = (14 - 3c)/4$, $y = (10 - c)/4$, $z = c$, $t = 0$, де c – довільна стала. Їй можна надавати довільні дійсні значення. При кожному значенні c виходить частковий розв'язок. Наприклад, при $c = 1$ виходить частковий розв'язок

```
(%i5) ev(%),/r1=1;
(%o5)
```

$$[x = \frac{11}{4},\ y = \frac{9}{4},\ z = 1,\ t = 0]$$

Спосіб подання розв'язання залежить від прапорця `linsolve_params` (типово `true`). Якщо зазначений прапорець установлений у значення `true`, розв'язання недовизначених систем включає параметри `%r1`, `%r2` тощо. Якщо прапорець `linsolve_params` установлений у значення `false`, зв'язані змінні виражаються через вільні.

Багато в чому аналогічний результат надає змогу одержати функція `algsys` (фактично, це надбудова над `solve`).

Функція `algsys([`$expr_1$`, `$expr_2$`, ... ,`$expr_m$`], [`x_1`, `x_2`, ..., `x_n`])` розв'язує систему поліноміальних рівнянь, що визначаються виразами `[`$expr_1 = 0$`, `$expr_2 = 0$`, ..., `$expr_m = 0$`]`, щодо списку змінних `[`x_1`, ..., `x_n`]`.

Вирази `[`$expr_1$`, ..., `$expr_n$`]` можуть бути представлені і у вигляді рівнянь. Кількість рівнянь може перевищувати кількість невідомих і навпаки.

Приклад:

```
(%i6) e1:2*x*(1-a1)-2*(x-1)*a2; e2:a2-a1;
e3:a1*(-y-x^2+1); e4:a2*(y-(x-1)^2);
(%o6)
```

$$2(1-a1)x-2a2(x-1)$$

```
(%o7)
```

$$a2-a1$$

```
(%o8)
```

$$a1(-y-x^2+1)$$

```
(%o9)
```

$$a2\left(y-(x-1)^2\right)$$

```
(%i10) algsys([e1, e2, e3, e4], [x, y, a1, a2]);
(%o10)
```

$$[[x = 0,\ y = \%r2,\ a1 = 0,\ a2 = 0],\ [x = 1,\ y = 0,\ a1 = 1,\ a2 = 1]]$$

Для обчислення коренів окремих поліноміальних рівнянь використовується функція `realroots`.

Варіанти синтаксису:

`realroots(expr, bound);`
`realroots(eqn, bound);`
`realroots(expr);`
`realroots(eqn).`

Функція знаходить всі корені виразу `expr = 0` або рівняння `eqn`. Функція будує послідовність Штурма для ізоляції кожного кореня і використовує алгоритм ділення навпіл для уточнення кореня з точністю `bound` або з точністю, заданою типово.

Приклад:

```
(%i11) realroots(2-x+x^5, 5e-6);
(%o11)
```

$[x = -\frac{664361}{524288}]$

```
(%i12) float(%);
(%o12)
```

$[x = -1.267168045043945]$

```
(%i13) ev(2-x+x^5,%[1]);
(%o13)
```

$3.085850166506531910^{-6}$

Всі корені полінома (дійсні і комплексні) можна знайти за допомогою функції `allroots`. Спосіб подання розв'язку визначається змінною `polyfactor` (типово `false`; якщо встановити у `true`, то функція повертає результат факторизації). Алгоритм пошуку коренів є напівобчислювальним.

Приклад:

```
(%i1) eqn:x^4+1; soln:allroots (eqn);
(%o1)
```

x^4+1

```
(%o2)
```

$[x = 0.70710678118655\,i + 0.70710678118655,\ \ x = 0.70710678118655 - 0.70710678118655\,i,$
$x = 0.70710678118655\,i - 0.70710678118655,\ \ x = -0.70710678118655\,i - 0.70710678118655]$

Кількість дійсних коренів рівняння на деякому інтервалі повертає функція `nroots` (синтаксис `nroots(p,low, high)`).

Приклад: (знаходимо кількість коренів рівняння на відрізку $[-6, 9]$):

```
(%i1) p: x^10-2*x^4+1/2$ nroots (p, -6, 9);
(%o2) 4
```

Для перетворення рівнянь використовуються функції `lhs` і `rhs`, що надають змогу виділити ліву і праву частину рівняння відповідно.

Приклад:

```
(%i1) eqn:x^2+x+1=(x-1)^3;
(%o1)
```

$x^2+x+1=(x-1)^3$

```
(%i2) lhs(eqn);
(%o2)
```

x^2+x+1

```
(%i3) rhs(eqn);
(%o3)
```

$(x-1)^3$

Спрощення систем рівнянь виконується функцією `eliminate`, що надає змогу виключити ті або інші змінні.

Виклик `eliminate([`eqn_1`, ..., `eqn_n`], [`x_1`, ..., `x_k`])` виключає змінні `[`x_1`, ..., `x_k`]` із зазначених виразів.

Приклад:

```
(%i1) expr1: 2*x^2+y*x+z;
expr2: 3*x+5*y-z-1;
expr3: z^2+x-y^2+5;
```
(%o1) $z + xy + 2x^2$
(%o2) $-z + 5y + 3x - 1$
(%o3) $z^2 - y^2 + x + 5$
```
(%i4) eliminate ([expr3, expr_2, expr_1], [y, z]);
```
(%o4) $[7425x^8 - 1170x^7 + 1299x^6 + 12076x^5 + 22887x^4 - 5154x^3 - 1291x^2 + 7688x + 15376]$

3.3 Класифікація і основні властивості функцій

Функція називається явною (або заданою неявним чином), якщо вона задана формулою, у якій права частина не містить залежної змінної; наприклад, функція $y = x^3 + 7x + 5$.

Функція y аргументу x називається неявною (або заданою неявним чином), якщо вона задана рівнянням $F(x, y) = 0$, не розв'язним щодо залежної змінної. Наприклад, функція y ($y \geqslant 0$), задана рівнянням $x^3 + y^2 - x = 0$. Відзначимо, що останнє рівняння задає дві функції, $y = \sqrt{x - x^3}$ при $y > 0$, і $y = -\sqrt{x - x^3}$ при $y < 0$.

Обернена функція

Нехай $y = f(x)$ є функцією від незалежної змінної x, визначеної на проміжку X, з областю значень Y. Поставимо у відповідність кожному $y \in Y$ єдине значення $x \in X$, при якому $f(x) = y$. Тоді отримана функція $x = g(y)$, визначена на проміжку Y, з областю значень X називається оберненою щодо функції $y = f(x)$.

Наприклад, для функції $y = a^x$ оберненою буде функція $x = \log_a x$.

Складена функція

Нехай функція $y = f(u)$ є функцією від змінної u, визначеною на множині U з областю значень Y, а змінна u, у свою чергу, є функцією $u = \varphi(x)$ від змінної x, визначеною на множині X з областю значень U. Тоді задана на множині X функція $y = f[\varphi(x)]$ називається складеною функцією.

Наприклад, $y = \sin(x^5)$ – складена функція, тому що її можна представити у вигляді $y = \sin(u)$, де $u = x^5$.

Поняття елементарної функції. Основними елементарними функціями є:

а) степенева функція $y = x^r$, $r \in \mathbb{R}$;

б) показникова функція $y = a^x$ ($a > 0$, $a \neq 1$);

в) логарифмічна функція $y = \log_n x$ ($a > 0$, $a \neq 1$);

г) тригонометричні функції $y = \sin x$, $y = \cos x$, $y = \operatorname{tg} x$, $y = \operatorname{ctg} x$;

д) обернені тригонометричні функції $y = \arcsin x$, $y = \arccos x$, $y = \operatorname{arctg} x$, $y = \operatorname{arcctg} x$.

З основних елементарних функцій нові елементарні функції можуть бути отримані за допомогою: а) алгебраїчних дій; б) операцій утворення складених функцій.

Означення. Функції, побудовані з основних елементарних функцій за допомогою скінченної кількості алгебраїчних дій і скінченної кількості операцій утворення складеної функції, називаються елементарними.

Наприклад, функція

$$y = \frac{\sqrt{x} + \arcsin x^5}{\ln^3 x + x^3 + x^7}$$

є елементарною.

Прикладом неелементарної функції є функція $y = \operatorname{sign} x$.

3.3.1 Основні властивості функцій

Парність і непарність. Функція

$$y = f(x)$$

називається парної, якщо $f(-x) = f(x)$ і непарної, якщо $f(-x) = -f(x)$. У противному випадку функція називається функцією загального вигляду.

Наприклад, функція $y = x^2$ є парною, а функція $y = x^3$ – непарною. Функція $y = x^2 + x^3$ є функцією загального вигляду.

Графік парної функції симетричний щодо осі ординат, а графік непарної функції симетричний відносно початки координат.

Монотонність. Функція $y = f(x)$ називається монотонно зростаючою (спадною) на проміжку X, якщо для будь-яких x_1, x_2 ($x_1, x_2 \in X$) і $x_2 > x_1$ виконується нерівність $f(x_2) > f(x_1)$ ($f(x_2) < f(x_1)$). А якщо виконується нерівність $f(x_2) > f(x_1)$ ($f(x_2) < f(x_1)$), то функція називається неспадною (незростаючою).

Обмеженість. Функція $y = f(x)$ називається обмеженою на проміжку X, якщо існує таке додатне число M, що $|f(x)| < M$ для будь-якого $x \in X$.

Наприклад, функція $y = \sin x$ обмежена на всій числовій осі, тому що $|sinx| < 1$ для будь-якого $x \in \mathbb{R}$.

Періодичність. Функція $y = f(x)$ називається періодичної з періодом $T \neq 0$ на проміжку X, для будь-якого $x \in X$ виконується рівність $f(x + T) = f(x)$.

3.3.2 Границя функції і її властивості

3.3.2.1 Границя функції в нескінченності

Означення. Число A називається границею функції $f(x)$ при x, що прямує до нескінченності, якщо для будь-якого як завгодно малого додатного числа $\varepsilon > 0$, знайдеться таке додатне число $\delta > 0$, що для всіх x, що задовольняють умові $|x| > \delta$ виконується нерівність $|f(x) - A| < \varepsilon$.

Ця границя функції позначається у такий спосіб:

$$\lim_{x\to\infty} f(x) = A$$

або $f(x) \to A$ при $x \to \infty$.

3.3.2.2 Границя функції у точці

Нехай функція $y = f(x)$ визначена в деякому околі точки a, крім, можливо, самої точки a.

Означення. Число A називається границею функції $f(x)$ при x, що прямує до a (або у точці a), якщо для будь-якого як завгодно малого додатного числа $\varepsilon > 0$, знайдеться таке додатне число $\delta > 0$, що для всіх x, що задовольняють умові $0 < |x - a| < \delta$ виконується нерівність $|f(x) - A| < \varepsilon$. Умова $0 < |x - a|$ означає, що $x \neq a$.

Границя функції позначається у такий спосіб:

$$\lim_{x\to a} f(x) = A$$

або $f(x) \to A$ при $x \to a$.

3.3.2.3 Однобічні границі

Якщо $x > a$ і $x \to a$, то використовують запис $x \to a + 0$. Якщо $x < a$ і $x \to a$, то використовують запис $x \to a - 0$. Вирази $\lim\limits_{x\to a+0} f(x)$ і $\lim\limits_{x\to a-0} f(x)$ називаються відповідно границями функції $f(x)$ у точці a праворуч і ліворуч.

Якщо існує границя $\lim\limits_{x\to a} f(x)$, то існують і однобічні границі $\lim\limits_{x\to a+0} f(x)$ і $\lim\limits_{x\to a-0} f(x)$ і

$$\lim_{x\to a-0} f(x) = \lim_{x\to a} f(x) = \lim_{x\to a+0} f(x)$$

Ця рівність виконується також, якщо границі ліворуч і праворуч рівні між собою[1].

3.3.2.4 Теореми про границі

1. Границя суми двох функцій дорівнює сумі границь цих функцій, якщо ті існують, тобто

$$\lim_{x\to x_0} f(x) + \lim_{x\to x_0} \psi(x) = A + B,$$

де $A = \lim\limits_{x\to x_0} f(x)$, $B = \lim\limits_{x\to x_0} \psi(x)$.

2. Границя добутку двох функцій дорівнює добутку границь цих функцій.

$$\lim_{x\to x_0} [f(x) \cdot \psi(x)] = A \cdot B.$$

3. Границя частки двох функцій дорівнює частці границь цих функцій.

$$\lim_{x\to x_0} \frac{f(x)}{\psi(x)} = \frac{A}{B},$$

причому $B \neq 0$.

4. Якщо

$$\lim_{u\to u_0} f(u) = A;$$

$$\lim_{x\to x_0} \psi(x) = u_0,$$

то границя складеної функції

$$\lim_{x\to x_0} f[\psi(x)] = A.$$

[1] Можна довести, що якщо існують і рівні між собою однобічні границі, то існує і границя функції, що дорівнює однобічним границям.

3.3.2.5 Обчислення границь різних класів функцій

Границя виразу $f(x)$ при $x \to a$ обчислюється за допомогою функції `limit(f (x),x, a)`;
Розгляньмо **приклад**: обчислити границю $\lim\limits_{x\to 0} \frac{\sin x}{x}$..

Розв'язування: виконаємо команду

```
(%i1) limit(sin(x)/x,x,0);
```

Результат на екрані:

```
(%o1) 1
```

Більше складні варіанти обчислення границь ілюструють наступні кілька прикладів, що включають границі ліворуч, праворуч, при прямуванні до нескінченності тощо. Розгляньмо границі: $\lim\limits_{x\to+\infty} e^x$, $\lim\limits_{x\to-\infty} e^x$, $\lim\limits_{x\to 0-0} \frac{1}{x}$, $\lim\limits_{x\to 0+0} \frac{1}{x}$.

Границя необмеженої функції на нескінченності:

```
(%i2) limit (exp(x), x, inf);
(%o2) ∞
(%i3) limit (exp(x), x, minf);
(%o3) 0
```

Межі при $x \to 0$ ліворуч і праворуч:

```
(%i3) limit(1/x, x, 0, minus);
(%o3) −∞
(%i4) limit(1/x, x, 0, plus);
(%o4) ∞
```

3.3.2.6 Границя і неперервність функції

Обчислити границі $\lim\limits_{x\to 8} \sqrt[3]{x}$ і $\lim\limits_{x\to -8} \sqrt[3]{x}$.

```
(%i8) limit(x^(1/3), x, 8);
(%o8) 2
(%i9) limit(x^(1/3), x, -8);
(%o9) −2
```

3.3.2.7 Границі раціональних дробів

Обчислити границю $\lim\limits_{x\to -1} \frac{x^3-3x-2}{(x^2-x-2)^2}$.

```
(%i10) y(x):=(x^3-3*x-2)/(x^2-x-2)^2; limit(y(x), x,-1);
```

$$(\%o10)\quad y(x) := \frac{x^3-3x-2}{(x^2-x-2)^2}$$

```
(%o11) −3
```

При операціях з раціональними дробами і виділення носіїв нуля[2] доцільно використати факторизацію виразів, наприклад: обчислення границі безпосередньо

```
(%i16) limit((x^2-4)/(x^2-3*x+2), x, 2);
(%o16) 4
```

Обчислення границі після факторизаціи раціонального вираз:

```
(%i17) factor((x^2-4)/(x^2-3*x+2));
```

У чисельнику і знаменнику дробу скорочується носій нуля при $x \to 2$, тобто вираз $x - 2$.

$$(\%o17)\quad \frac{x+2}{x-1}$$

```
(%i18) limit(%,x,2);
(%o18) 4
```

[2]Під «носіями нуля» маємо на увазі вираз, що перетворюється у нуль у точці, у якій обчислюється границя.

3.3.2.8 Границі, що містять ірраціональні вираз

Обчислення границь даного класу багато у чому аналогічне обчисленню границь раціональних дробів, тому що зводиться до скорочення носіїв нуля в чисельнику і знаменнику аналізованих виразів, наприклад: обчислити границю виразу $\dfrac{\sqrt{x}-1}{x-1}$ при $x \to 1$. При обчисленні границі безпосередньо маємо:

```
(%i1) limit((sqrt(x)-1)/(x-1), x, 1);
(%o1) 1/2
```

Для спрощення і скорочення носіїв нуля використовується функція `radcan`:

```
(%i2) factor((sqrt(x)-1)/(x-1));
(%o2) (sqrt(x)-1)/(x-1)
(%i3) radcan(%);
(%o3) 1/(sqrt(x)+1)
```

3.3.2.9 Границі тригонометричних виразів

Першою визначною границею називається границя

$$\lim_{x\to 0}\frac{\sin x}{x}=1.$$

Розгляньмо приклади знаходження деяких границь з використанням першої визначної границі.

Приклад. Знайти границю

$$\lim_{x\to 0}\frac{\sin 5x}{x}.$$

$$\lim_{x\to 0}\frac{\sin 5x}{x}=\lim_{x\to 0}5\frac{\sin 5x}{5x}=5\lim_{t\to 0}\frac{\sin t}{t}=5,$$

де $t=5x$.

Обчислення з використанням **Maxima**:

```
(%i1) limit(sin(5*x)/x, x, 0);
(%o1) 5
```

Приклад. Знайти границю $\lim\limits_{x\to 0}\dfrac{1-2\cos 2x}{x^2}$.

$$\lim_{x\to 0}\frac{1-\cos 2x}{x^2}=\lim_{x\to 0}\frac{2\sin^2 x}{x^2}=2\left(\lim_{x\to 0}\frac{\sin x}{x}\right)^2=2.$$

Обчислення з використанням **Maxima**:

```
(%i4) limit((1-cos(2*x))/x^2, x, 0);
(%o4) 2
```

3.3.2.10 Межі експоненційних виразів

Другою визначною границею називається границя

$$\lim_{x\to\infty}\left(1+\frac{1}{x}\right)^x=e=2.718281828\ldots$$

Можна показати, що функція $y(x)=\left(1+\dfrac{1}{x}\right)^x$ при $x\to+\infty$ та при $x\to-\infty$ також має границю, рівну e.

$$e=\lim_{x\to\infty}\left(1+\frac{1}{x}\right)^x$$

Заміняючи x на $x=1/t$, одержимо ще один запис числа e

$$e=\lim_{t\to 0}(1+t)^{1/t}.$$

Число e (число Ейлера або неперове число) відіграє важливу роль у математичному аналізі.

Функція $y=e^x$ називається експонентою. Якщо показник експоненти громіздкий, то її прийнято записувати у вигляді: $\exp(x)$.

Логарифм за основою e називається натуральним. Його позначають символом ln, тобто $\log_e x = \ln x$.

Важливу роль у математичному аналізі відіграють також гіперболічні функції (гіперболічний синус, гіперболічний косинус, гіперболічний тангенс), що визначаються такими формулами:

$$\operatorname{sh} x = \frac{e^x - e^{-x}}{2}; \operatorname{ch} x = \frac{e^x + e^{-x}}{2}; \operatorname{th} x = \frac{\operatorname{sh} x}{\operatorname{ch} x}.$$

Розгляньмо приклади знаходження деяких границь з використанням другої визначної границі.

Приклад. Знайти границю $\lim\limits_{x\to 0} \dfrac{\ln(1+x)}{x}$.

$$\lim_{x\to 0} \frac{\ln(1+x)}{x} = \lim_{x\to 0} \ln(1+x)^{1/x} = \ln \lim_{x\to 0} (1+x)^{1/x} = \ln e = e.$$

Приклад. Знайти границю $\lim\limits_{x\to 0} \dfrac{a^x - 1}{x}$.

Нехай $a^x - 1 = u$. Тоді $a^x = 1 + u$; $x = \dfrac{\ln(1+u)}{\ln a}$.

$$\lim_{x\to 0} \frac{a^x - 1}{x} = \lim_{u\to 0} \frac{u \ln a}{\ln(1+u)} = \ln a \lim_{u\to 0} \frac{u}{\ln(1+u)} = \ln a \cdot 1 = \ln a.$$

Обчислення за допомогою **Maxima**:

```
(%i5) limit(log(1+x)/x, x, 0);
(%o5) 1
(%i6) limit((a^x-1)/x, x, 0);
(%o6) log  a
```

Знайдемо границю $\lim\limits_{x\to\infty} \left(\dfrac{x}{2+x}\right)^{3x}$

Аналітичний розрахунок дає такий результат:

$$\lim_{x\to\infty} \left(\frac{x}{2+x}\right)^{3x} = \exp\left[\lim_{x\to\infty} 3x \ln\left(1 - \frac{2}{2+x}\right)\right] = e^{-6}.$$

Використовуючи **Maxima**, одержуємо:

```
(%i7) limit((x/(2+x))^(3*x), x, inf);
(%o7) %e^-6
```

3.3.2.11 Нескінченно малі і нескінченно великі функції

Порівняння нескінченно малих функцій.

Розгляньмо границю частки від ділення двох нескінченно малих $\alpha(x)$ і $\beta(x)$ при $x \to a$.

Границя відносини двох нескінченно малих величин $= \lim\limits_{x\to a} \dfrac{\alpha(x)}{\beta(x)}$ може бути дорівнює нулю, кінцевому числу або ∞.

1. Якщо A скінченне, то $\alpha(x)$ і $\beta(x)$ називають нескінченно малими одного порядку і пишуть $\alpha(x) = O[\beta(x)]$ при $x \to a$.

   Якщо $A = 1$, то $\alpha(x)$ і $\beta(x)$ називають еквівалентними і пишуть $a(x) \sim \beta(x)$ при $x \to a$.

2. Якщо $A = 0$, то $\alpha(x)$ називають нескінченно малою вищого порядку, чим $\beta(x)$ і пишуть $\alpha(x) = o[\beta(x)]$ при $x \to a$.

   Якщо існує дійсне число $r > 0$ таке, що

   $$\lim_{x\to a} \frac{\alpha(x)}{[\beta(x)]^r} \neq 0,$$

   то $\alpha(x)$ називають нескінченно малою порядку r відносно $\beta(x)$ при $x \to a$.

3. Якщо $A \to \infty$ при $x \to a$, то у цьому випадку $\beta(x)$ називають нескінченно малою вищого порядку, чим $\alpha(x)$ і пишуть $\beta(x) = o[\alpha(x)]$.

Звичайно, може трапитися, що відношення двох нескінченно малих не прямує ні до якої границі, наприклад, якщо взяти $\alpha = x$ і $\beta = x \sin \frac{1}{x}$, то їхнє відношення, рівне $\sin \frac{1}{x}$, при $x \to 0$ границі не має. У такому випадку говорять, що дві нескінченно малі не порівняні між собою.

Приклад обчислень із **Maxima**:

Розгляньмо дві нескінченно малі функції при $x \to 0$

```
(%i1) f(x):=sin(3*x)*sin(5*x)$ g(x):=(x-x^3)^2$
```

Обчислимо границю відносини $f(x)/g(x)$ при $x \to 0$

```
(%i2) limit(f(x)/g(x), x, 0);
(%o2) 15
```

Результат, рівний сталому числу, свідчить про те, що розглянуті нескінченно малі одного порядку.

3.3.2.12 Еквівалентні нескінченно малі. Їхнє застосування до обчислення меж

При обчисленні границь корисно мати на увазі еквівалентність наступних нескінченно малих величин: $\sin x \sim x$; $\operatorname{tg} x \sim x$; $\arcsin x \sim x$; $\operatorname{arctg} x \sim x$; $\ln(1+x) \sim x$, при $x \to 0$.

Їх нескладно одержати, використовуючи правило Лопіталя (див. нижче).

Приклад: Порівняти нескінченно малі $\alpha(x) = x^2 \sin^2 x$ і $\beta(x) = x \operatorname{tg} x$ при $x \to 0$.

Замінимо $\sin^2 x$ і $\operatorname{tg} x$ на їхні еквівалентні нескінченно малі $\sin^2 x \sim x^2$ і $\operatorname{tg} x \sim x$. Одержимо

$$\lim_{x\to 0} \frac{\alpha(x)}{\beta(x)} = \lim_{x\to 0} \frac{x^2 \sin^2 x}{x \operatorname{tg} x} = \lim_{x\to 0} \frac{x^2 \cdot x^2}{x \cdot x} = \lim_{x\to 0} \frac{x^4}{x^2} = \lim_{x\to 0} x^2 = 0.$$

Таким чином, $\alpha(t) = o[\beta(t)]$ при $t \to 0$. Крім того, $\alpha(x)$ є нескінченно малою порядку 2 відносно $\beta(x)$.

Приклад: Визначити порядок малості $\alpha(x) = \sin(\sqrt{x+1} - 1)$ відносно $\beta(x) = x$ при $x \to 0$.

Оскільки

$$\sqrt{x+1} - 1 = \frac{(\sqrt{x+1} - 1)(\sqrt{x+1} + 1)}{\sqrt{x+1} + 1} = \frac{x}{\sqrt{x+1} + 1},$$

$$\lim_{x\to 0} \frac{\alpha(x)}{\beta(x)} = \lim_{x\to 0} \left[\frac{1}{x} \sin\left(\frac{x}{\sqrt{x+1} + 1}\right)\right] = \frac{1}{2}.$$

При обчисленнях з використанням **Maxima** природніше використати при обчисленні складних границь і порівнянні нескінченно малих розклад чисельника і знаменника у ряд Тейлора (докладне обговорення степеневих рядів – див. нижче)ю При обчисленні з використанням меню у меню «Аналіз → Знайти границю», установити пункт «Ряд Тейлора». Для обчислень використовується функція `tlimit`, робота якої заснована на заміні досліджуваних функцій рядом Тейлора (де це можливо). Типово прапорець заміни встановлено у `false`, тому для використання `tlimit` прапорець заміни встановлюється у `true`:

```
(%i1) tlimswitch=true;
(%o1) false = true
```

приклад обчислення з використанням `tlimit`:

```
(%i1) f(x):=(tan(x)-sin(x))/(x-sin(x));
(%o1) f(x) := (tan(x) - (x))/(x - sin(x))
(%i2) tlimit(f(x),x,0);
(%o2) 3
```

3.3.2.13 Нескінченно великі функції. Зв'язок між нескінченно малими і нескінченно великими величинами

Функція $f(x)$ називається нескінченно великою величиною при $x \to a$, якщо для будь-якого $\varepsilon > 0$ зайдеться таке $\delta > 0$, що для всіх x, що задовольняють умову $0 < |x - a| < \delta$, буде виконана нерівність $|f(x)| > \varepsilon$.

Запис того, що функція $f(x)$ нескінченно велика при $x \to a$ означає наступне: $\lim\limits_{x\to a} f(x) = \infty$ або $f(x) \to \infty$ при $x \to a$.

Приклад: $y = \operatorname{tg} x$ нескінченно велика при $x \to \pi/2$.

Зауваження: Функція може бути необмеженою, але не нескінченно великою. Наприклад, функція $y = x \sin x$ – не обмежена на $(-\infty, \infty)$, але не нескінченно велика при $x \to \infty$.

Якщо функція $\alpha(x)$ є нескінченно мала величина при $x \to a$ ($x \to \infty$), то функція $f(x) = \dfrac{1}{\alpha(x)}$ є нескінченно великою при $x \to a$ ($x \to \infty$).

І навпаки, якщо функція $f(x)$ нескінченно велика при $x \to a$ ($x \to \infty$), то функція $\alpha(x) = \dfrac{1}{f(x)}$ є величиною нескінченно малою при $x \to a$ ($x \to \infty$).

Наприклад, функція $y = \cos x$ – нескінченно мала при $x \to \pi/2$, тоді функція $\dfrac{1}{\cos x}$ – нескінченно велика. Функція $y = \dfrac{1}{2x - 7}$ – нескінченно мала при $x \to \infty$, тоді функція $y = 2x - 7$ – нескінченно велика при $x \to \infty$.

3.3.3 Неперервні функції

Поняття неперервності функції, так само як і поняття границі, є одним з основних понять математичного аналізу.

3.3.3.1 Неперервність функції у точці

Дамо два визначення поняття неперервності функції у точці.

Означення 1. *Функція $f(x)$ називається неперервною у точці a, якщо вона задовольняє трьом умовам: 1) $f(x)$ визначена в деякій околі точки $x = a$, 2) існує скінченна границя $\lim\limits_{x\to a} f(x)$, 3) ця границя дорівнює значенню функції $f(x)$ у точці a, тобто $\lim\limits_{x\to a} f(x) = f(a)$.* Очевидно, що неперервність функції у даній точці виражається неперервністю її графіка при проходженні даної точки.

Розгляньмо друге визначення неперервності функції у точці.

Надамо аргументу a приріст $\Delta x = 0$. Тоді функція $y = f(x)$ одержить приріст Δy, що визначається як різниця нарощеного і вихідного значення функції: $\Delta y = f(a + \Delta x) - f(a)$.

Означення 2. Функція $y = f(x)$ називається неперервної у точці a, якщо вона визначена у деякому околі точки $x = a$, і приріст її Δy у цій точці, що відповідає приросту Δx, прямує до нуля при прямуванні Δx до нуля: $\lim\limits_{\Delta x\to 0} \Delta y = 0$.

У підручниках з математичного аналізу доведено, що ці визначення рівносильні.

Приклад дослідження неперервності функції з **Maxima**:

Функція

$$f(x) := \frac{1}{1 + \exp\left(\frac{1}{1-x}\right)}$$

має можливу точку розриву при $x = 1$. Зіставимо границі цієї функції при прямуванні x до 1 ліворуч і праворуч:

```
(%i16) f(x) := 1/(1+exp(1/(1-x)));
```

(%o16) $f(x) := \dfrac{1}{1 + exp\left(\frac{1}{1-x}\right)}$.

```
(%i17) limit(f(x),x,1,plus);
(%o17) 1
(%i18) limit(f(x),x,1,minus);
(%o18) 0
```

Границі не збігаються, тому робимо висновок, що досліджувана функція розривна.

3.3.3.2 Властивості неперервних функцій

1. *Якщо функції $f(x)$ і $g(x)$ неперервні у точці a, то їхня сума $f(x) + g(x)$, добуток $f(x)g(x)$, і частка $f(x)$ (за умови, що $g(a) = 0$) є функціями, неперервними у точці a.*
2. *Якщо функція $y = f(x)$ неперервна у точці a і $f(a) > 0$, то існує такий окіл точки a, у якій $f(x) > 0$.*
3. *Якщо функція $y = f(u)$ неперервна у точці u_0, а функція $u = \psi(x)$ неперервна у точці $u_0 = \psi(x_0)$, то складена функція $y = f[\psi(x)]$ неперервна у точці x_0.*

Властивість 3 може бути записано у вигляді:

$$\lim_{x\to x_0} f[\psi(x)] = f\left[\lim_{x\to x_0} \psi(x)\right],$$

тобто *під знаком неперервної функції можна переходити до границі.*

Функція $y = f(x)$ називається неперервною на проміжку X, якщо вона неперервна у кожній точці цього проміжку. Можна довести, що всі елементарні функції неперервні в області їхнього визначення.

3.3.3.3 Точки розриву функцій і їхня класифікація

Точка a, що належить області визначення функції або є граничною для цієї області, називається точкою розриву функції $f(x)$, якщо у цій точці порушується умова неперервності функції.

Якщо існують скінченні границі

$$f(a - 0) = \lim_{x\to a-0} f(x) \text{ і } f(a + 0) = \lim_{x\to a+0} f(x),$$

причому не всі три числа $f(a)$, $f(a - 0)$, $f(a + 0)$ рівні між собою, то точка a називається *точкою розриву 1 роду* (існують кінцеві однобічні границі функції ліворуч і праворуч, не рівні одна одній.

Точки розриву 1 роду підрозділяються, у свою чергу, на точки усувного розриву (коли $f(a - 0) = f(a + 0) = f(a)$, тобто коли ліва і права границі функції $f(x)$ у точці a рівні між собою, але не дорівнюють значенню функції $f(x)$ у цій точці) і на точки стрибка (коли $f(a - 0) \neq f(a + 0)$, тобто коли ліва і права границі функції у точці a різні); в останньому випадку різниця $f(a + 0) - f(a - 0)$ називається стрибком функції $f(x)$ у точці a.

Точки розриву, що не є точками розриву 1 роду, називаються точками розриву 2 роду. У точках розриву 2 роду не існує хоча б одна з однобічних границь.

Розгляньмо попередній приклад. Функція

$$f(x) := \frac{1}{1+\exp\left(\frac{1}{1-x}\right)}$$

має точку розриву при $x = 1$.

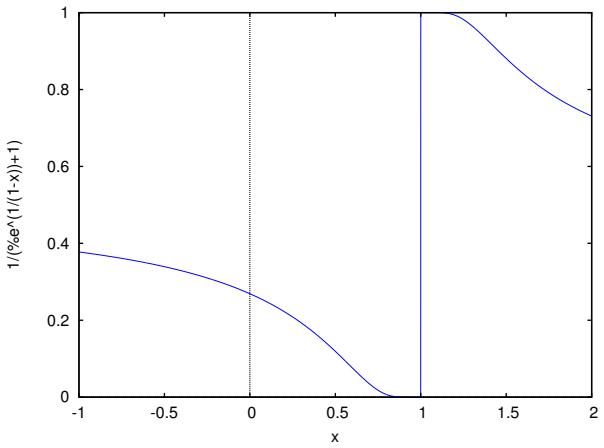


Рис. 3.1. Розрив досліджуваної функції

Оскільки границі $\lim\limits_{x\to 0-0} f(x)$ і $\lim\limits_{x\to 0+0} f(x)$ не збігаються, але обидві скінченні, робимо висновок про наявність точки розриву першого роду при $x = 1$.

Графічну ілюстрацію одержуємо за допомогою **wxMaxima** (див. рис. 3.1).

3.3.4 Диференціювання за допомогою пакета Maxima

Пакет **Maxima** надає потужні засоби для диференціювання функцій і обчислення диференціалів. Для обчислення найпростішій похідній треба у командному вікні після запрошення **Maxima** увести команду наступного вигляду: `diff(<функція>, <змінна>)`; де `<функція>` – вираз, що задає функцію (не обов'язково однієї змінної), наприклад `x^2+2*x+1`; `<змінна>` – назва змінної, за якою виконуватиметься диференціювання, наприклад x.

Прикладом обчислення похідній може служити така команда: `diff(x^2+2*x+1,x);`.

За допомогою команди `diff` можна обчислювати похідні вищих порядків. При цьому команда має такий формат: `diff(<функція>,<змінна>,<порядок>);` де `<порядок>` – порядок обчислюваної похідної.

У розв'язаннях деяких прикладів цієї глави за допомогою **Maxima** будуть використані додаткові команди **Maxima**:

- `ratsimp(<вираз>)`, `radcan(<вираз>)` – спрощення алгебраїчного виразу.
- `trigsim(<вираз>)`, `trigexpand(<вираз>` – спрощення або підстановка тригонометричного виразу.
- `factor(<вираз>);` – розкласти `<вираз>` на множники.
- `at(<вираз>,<old>=<new>);` – підставити вираз `<new>` на місце `<old>` в <виразі>.
- `<змінна>:solve(<вир1>=<значення>,<вир2>);` – надати `<змінній>` значення виразу `<вир2>`, отримане дозволом рівняння `<вир1>`(`<вир2>`)=<значення>.
- `taylor(f(x),x,x0,n);` – розкласти функцію `f(x)` за формулою Тейлора із центром у точці `x0` до порядку `n` включно.

3.3.4.1 Обчислення похідних і диференціалів

Для обчислення похідної функції використовується функція `diff`, для обчислення похідних різного порядку зручно створити користувацьку функцію (у прикладі нижче – `f(x)`):

```
(%i3) f(x):=sin(9*x^2);
(%o3) f(x) := sin(9x^2)
(%i4) d1:diff(f(x),x,1);
(%o4) 18 x cos(9x^2)
(%i5) d2:diff(f(x),x,2);
(%o5) 18 cos (9x^2) - 324x^2 sin (9x^2)
(%i6) d3:diff(f(x),x,3);
(%o6) -972x sin (9x^2) - 5832x^3 cos (9x^2)
```

Приклад обчислення диференціала (`del(x)` рівноцінно `dx`, не зазначена явно змінна диференціювання):

```
(%I8) diff(log(x));
```
$$\text{(\%o8)}\quad \frac{del(x)}{x}$$

Аналогічний підхід застосуємо і для функції декількох змінних. Функція `diff` з єдиним аргументом – диференційованою функцією – повертає повний диференціал.

Приклад:

```
(%i9) diff(exp(x*y))
```
$$\text{(\%o9)}\quad x\%e^{x}del(y)+y\%e^{xy}del(x)$$

Приклад:

```
(%I10) diff(exp(x*y*z));
```
$$\text{(\%o10)}\quad xy\%e^{xyz}del(z)+xz\%e^{xyz}del(y)+yz\%e^{xyz}del(x)$$

Якщо вказати апостроф перед символом `diff`, то похідна не обчислюється і спрощення, звичайно передбачене типово, не здійснюється.

Приклад:

Створюємо функцію f (x, z):

```
(%i18) f(x,z):=x^2*z+z^2*x;
```
$$\text{(\%o18)}\quad f(x,z):=x^2z+z^2x$$

Обчислюємо диференціальний вираз:

```
(%i19) diff(f(x,z),x,2)+diff(f(x,z),z,3)+diff(f(x,z),x)*x^2;
```
$$\text{(\%o19)}\quad x^2(z^2+2xz)+2z$$

Виконуємо формальне диференціювання, не обчислюючи безпосередньо результат:

```
(%i20) 'diff(f(x,z),x,2)+'diff(f(x,z),z,3)+'diff(f(x,z),x)*x^2;
```
$$\text{(\%o20)}\quad \frac{d^3}{d\,z^3}\,f\,(x,z)\;+\frac{d^2}{d\,x^2}\,f\,(x,z)\;+x^2\left(\frac{d}{d\,x}\,f\,(x,z)\;\right)$$

3.4 Екстремуми функцій

3.4.1 Відшукання максимумів і мінімумів

Точки, де досягається найбільше або найменше значення функції називаються відповідно точками максимуму або мінімуму функції.

Означення 1. *Точка x_0 називається точкою максимуму функції $f(x)$, якщо в деякій околі точки x_0 виконується нерівність $f(x)\geqslant f(x_0)$.*

Означення 2. *Точка x_1 називається точкою мінімуму функції $f(x)$, якщо в деякому околі точки x_1 виконується нерівність $f(x)\leqslant f(x_1)$.*

Значення функції у точках x_0 і x_1 називаються відповідно *максимумом і мінімумом функції*. Максимум і мінімум функції поєднуються загальною назвою екстремуму функції.

Екстремум функції часто називають *локальним* екстремумом, підкреслюючи тим самим, що поняття екстремуму зв'язане лише з досить малим околом точки x_0. Так що на одному проміжку функція може мати декілька екстремумів, причому може трапитися так, що мінімум в одній точці більше максимуму в іншій.

Наявність максимуму (або мінімуму) в окремій точці проміжку X зовсім не означає, що у цій точці функція $f(x)$ приймає найбільше (найменше) значення на цьому проміжку (або, як говорять має глобальний максимум (мінімум)).

3.4.1.1 Теорема Ферма

Теорема Ферма. *Якщо диференційована на проміжку X функція $y=f(x)$ досягає найбільшого або найменшого значення у внутрішній точці x_0, то тоді похідна функції у цій точці дорівнює нулю, тобто $f'(x_0)=0$.*

Нехай функція $y=f(x)$ диференційована на проміжку X і у точці $x_0\in X$ приймає найменше значення.

Тоді

$$f(x_0+\Delta x)\geqslant f(x_0),$$

якщо $x_0+\Delta x\in X$ і, отже

$$\Delta y=f(x_0+\Delta x)-f(x_0)\geqslant 0$$

при досить малих Δx і незалежно від знаку Δx.

Тому

$$\frac{\Delta y}{\Delta x}\geqslant 0 \text{ при } \Delta x>0 \text{ (праворуч від } x_0),$$

$$\frac{\Delta y}{\Delta x} \leqslant 0 \text{ при } \Delta x < 0 \text{ (ліворуч від } x_0).$$

Переходячи до границі праворуч і ліворуч отримаємо

$$\lim_{\Delta x \to 0+} \frac{\Delta y}{\Delta x} \geqslant 0 \text{ і } \lim_{\Delta x \to 0-} \frac{\Delta y}{\Delta x} \leqslant 0$$

Оскільки функція диференційована на проміжку X, то границі праворуч і ліворуч рівні

$$\lim_{\Delta x \to 0+} \frac{\Delta y}{\Delta x} = \lim_{\Delta x \to 0-} \frac{\Delta y}{\Delta x}.$$

Звідси $f'(x_0) = 0$.

Аналогічну послідовність міркувань можна побудувати і для максимуму.

Теорему Ферма часто називають необхідною умовою екстремуму **диференційованої** функції.

Геометричний зміст теореми Ферма: *у точці екстремуму, що досягається усередині проміжку X, дотична до графіка функції паралельна осі абсцис.*

3.4.1.2 Необхідна умова екстремуму

Якщо у точці x_0 диференційована функція $f(x)$ має екстремум, то в деякій околі цієї точки виконуються умови теореми Ферма, і отже, похідна функції у цій точці дорівнює нулю, тобто $f'(x_0) = 0$. Але функція може мати екстремум і у точках, у яких вона не диференційована. Так, наприклад, функція $y = |x|$ має екстремум (мінімум) у точці $x = 0$, але не диференційована у ній. Функція $y = \sqrt[3]{x^2}$ також має у точці $x = 0$ мінімум, а її похідна у цій точці нескінченна: $y' = \frac{2}{3\sqrt[3]{x}}$, $y'(0) = \infty$.

Тому необхідна умова екстремуму може бути сформульоване у такий спосіб.

Для того щоб функція $y = f(x)$ мала екстремум у точці x_0, необхідно, щоб її похідна у цій точці рівнялася нулю ($f'(x_0) = 0$) або не існувала.

Точки, у яких виконана необхідна умова екстремуму, називаються *критичними* (або *стаціонарними*). Але *критична точка не обов'язково є точкою екстремуму.*

Приклад. Знайти критичні точки функції і переконатися у наявності або відсутності екстремуму у цих точках:
1. $y = x^2 + 1$; 2. $y = x^3 - 1$.

1. $y' = 2x$. $y'(x) = 0$ при $x = 0$. У точці $x = 0$ функція $y = x^2 + 1$ має мінімум.
2. $y' = 3x^2$. $y'(x) = 0$ при $x = 0$. У точці $x = 0$ функція $y = x^3 - 1$ не має екстремуму. Функція $y = x^3 - 1$ зростає на всій числовій осі.

Отже, для знаходження екстремумів функції потрібне додаткове дослідження критичних точок.

Приклад: Дослідити на наявність екстремуму наступну функцію

$$y(x) = x^3 - 3x^2 + 3x + 2.$$

Задаємо досліджувану функцію

```
(%i1) f(x):=x^3-3*x^2+3*x+2;
(%o1) f(x) := x^3 - 3x^2 + 3x + 2
```

Похідну у формі функції визначаємо **явно**, використовуючи функцію `define`

```
(%i2) define(df(x),diff(f(x),x));
(%o2) df(x) := 3x^2 - 6x + 3
```

Розв'язуючи рівняння $df(x) = 0$ (тобто $f'(x) = 0$, знаходимо критичні точки

```
(%i3) solve(df(x)=0,x);
(%o3) [x = 1]
```

У цьому випадку критична точка одна – $x = 1$.

3.4.1.3 Перша достатня умова екстремуму

Теорема. *Якщо при переході через точку x_0 похідна диференційованої функції $y = f(x)$ міняє свій знак із плюса на мінус, то точка x_0 є точка максимуму функції $y = f(x)$, а якщо з мінуса на плюс, то – точка мінімуму.*

Нехай похідна міняє знак із плюса на мінус, тобто у деякому інтервалі (a, x_0) похідна додатна ($f'(x) > 0$), а в деякому інтервалі (x_0, b) – від'ємна ($f'(x) < 0$). Тоді відповідно до достатньої умови монотонності функція $f(x)$ зростає на інтервалі (a, x_0) і спадає на інтервалі (x_0, b).

За визначенням зростаючої функції $f(x_0) \geqslant f(x)$ при всіх $x \in (a, x_0)$, а за визначенням спадної функції $f(x) \leqslant f(x_0)$ при всіх $x \in (x_0, b)$, тобто $f(x_0) \geqslant f(x)$ при всіх $x \in (a, b)$, отже, x_0 – точка максимуму функції $y = f(x)$.

Аналогічно розглядається випадок, коли похідна міняє знак з мінуса на плюс.

Відзначимо, що диференційованість функції в самій точці x_0 не використовувалася при доведенні теореми. Насправді вона і не потрібна – достатньо, щоб функція була неперервна у точці x_0.

Якщо зміна знаку похідної не відбувається, то екстремуму немає. Однак при роботі із системами комп'ютерної математики зручнішою є друга достатня умова екстремуму.

3.4.1.4 Друга достатня умова екстремуму

Теорема. Якщо перша похідна $f'(x)$ двічі диференційованої функції $y = f(x)$ дорівнює нулю у деякій точці x_0, а друга похідна у цій точці $f''(x_0)$ додатна, то x_0 є точкою максимуму функції $y = f(x)$; якщо $f''(x_0)$ від'ємна, то x_0 – точка максимуму.

Нехай $f'(x_0) = 0$, а $f''(x_0) > 0$. Це значить, що

$$f''(x) = (f'(x))' > 0$$

також і у деякому околі точки x_0, тобто $f'(x)$ зростає на деякому інтервалі (a, b), що містить точку x_0.

Але $f'(x_0) = 0$, отже, на інтервалі (a, x_0) $f'(x) < 0$, а на інтервалі (x_0, b) $f'(x) > 0$, тобто $f'(x)$ при переході через точку x_0 міняє знак з мінуса на плюс, тобто x_0 – точка мінімуму.

Аналогічно розглядається випадок $f'(x_0) = 0$ і $f''(x_0) < 0$. Продовжимо дослідження функції

$$y(x) = x^3 - 3x^2 + 3x + 2.$$

Як установлено вище, є одна критична точка: $x = 1$. Задаємося функцією `d2f(x)`

```
(%i4) define(d2f(x),diff(df(x),x));
(%o4) d2f(x) := 6x - 6
```

Обчислюємо значення другої похідної у критичній точці:

```
(%i5) map(d2f,%o3);
(%o5) [6x - 6 = 0]
```

У даному прикладі неможливо визначити, чи є точка $x = 1$ екстремумом досліджуваної функції, оскільки друга похідна у ній виявилася рівною 0. Варто звернути увагу на спосіб обчислення – функція `d2f(x)` застосовується до всіх елементів списку, отриманого при розв'язанні рівняння $f'(x) = 0$ (використовується вбудована функція **Maxima** `map`).

Скористаємося першою достатньою ознакою наявності екстремуму

```
(%i6) df(0);
(%o6) 3
(%i7) df(2);
(%o7) 3
```

Як видно з наведеного результату, перша похідна не змінює знак у критичній точці, що свідчить про відсутність екстремуму у ній.

Отриманий результат ілюструється графіком досліджуваної функції і її похідних (див. рис. 3.2).

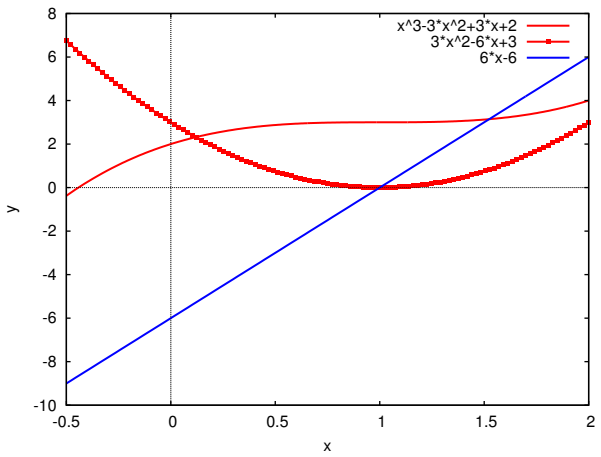


Рис. 3.2. Приклад дослідження функції

3.4.1.5 Схема дослідження функції y=f(x) на екстремум

1. Знайти похідну $y' = f'(x)$.
2. Знайти критичні точки функції, у яких похідна $f'(x) = 0$ або не існує.
3. Дослідити знак похідній ліворуч і праворуч від кожної критичної точки і зробити висновок про наявність екстремумів функції.

   Або
4. Знайти другу похідну $f''(x)$ і визначити її знак у кожній критичній точці.
5. Знайти екстремуми (екстремальні значення) функції.

Приклад. Дослідити на екстремум функцію $y = x(x-1)^3$.

1. $y' = (x-1)^3 + 3x(x-1)^2 = (x-1)^2(4x-1)$.
2. Критичні точки $x_1 = 1$ і $x_2 = \dfrac{1}{4}$.
3. Зміна знака похідній при переході через точку x_1 не відбувається, тому у цій точці немає екстремуму.
   $y'' = 2(x-1)(4x-1) + 4(x-1)^2 = 2[(x-1)(6x-3)]$.
   $y''(x_2) > 0$, тому у цій точці спостерігається мінімум функції $y = x(x-1)^3$.
4. $y_{min} = y\left(\dfrac{1}{4}\right) = -\dfrac{27}{256}$.

Виконаємо той же розрахунок за допомогою **Maxima**

```
(%i13) f(x):=x*(x-1)^3;
(%o13) f(x) := x(x-1)^3
(%i14) define(df(x),diff(f(x),x));
(%o14) df(x) := 3(x-1)^2x + (x-1)^3
(%i15) solve(df(x)=0,x);
(%o15) [x = 4, x = 1]
(%i16) define(d2f(x),diff(df(x),x));
(%o16) d2f(x) := 6(x-1)x + 6(x-1)^2
(%i17) map(d2f,%o15);
(%o17) [6(x-1)x + 6(x-1)^2 = 9/4, 6(x-1)x + 6(x-1)^2 = 0]
```

У точці $x = 1$ друга похідна дорівнює 0, тому обчислюємо значення першої похідної ліворуч і праворуч від $x = 1$:

```
(%i18) df(2);
(%o18) 7
(%i19) df(1/3);
(%o19) 4/27
```

Похідна в околі точки $x = 1$ не міняє знак, тому екстремум у досліджуваної функції один – точка $x = 1$. Оскільки $d2f(\frac{1}{4}) > 0$, $x = \frac{1}{4}$ – точка мінімуму. Ілюстрація отриманого результату – на рисунку 3.3.

3.4.1.6 Знаходження найбільших і найменших значень функції

Найбільше або найменше значення функції на деякому відрізку може досягатися як у точках екстремуму, так і у точках на кінцях відрізка.

Нехай функція $y = f(x)$ визначена на деякому відрізку $[a, b]$.

Знаходження найбільших і найменших значень функцій відбувається за наступною схемою.

1. Знайти похідну $f'(x)$.
2. Знайти критичні точки функції, у яких $f'(x_0) = 0$ або не існує.
3. Знайти значення функції у критичних точках і на кінцях відрізка і вибрати з них найбільше f_{MAX} і найменше f_{MIN} значення. Це і будуть найбільше і найменше значення функції на досліджуваному відрізку.

Приклад. Знайти найбільше і найменше значення функції $y = 3x^2 - 6x$ на відрізку $[0, 3]$. Аналітичний розрахунок:

1. $y' = 6x - 6$; $y'' = 6$.
2. $x_0 = 1$.

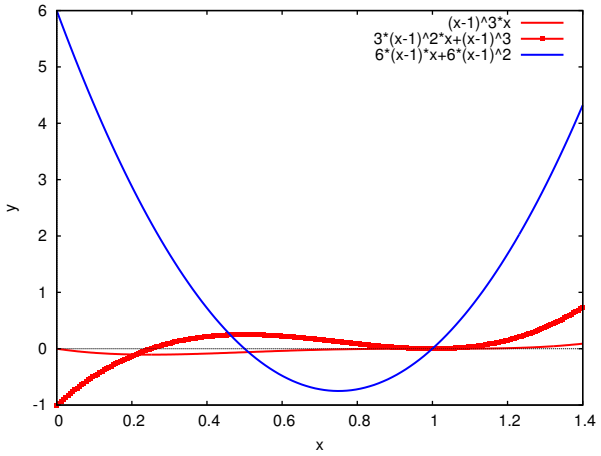


Рис. 3.3. Приклад дослідження функції на екстремум

3. $y(1) = -3$; $y(0) = 0$; $y(3) = 9$.

У точці $x = 1$ найменше значення функції, а у точці $x = 3$ – найбільше.
Розрахунок з використанням **Maxima**:
Знаходимо критичні точки досліджуваної функції

```
(%i29) f(x):=3*x^2-6*x;
(%o29) f(x) := 3x^2 - 6x
(%i30) define(df(x),diff(f(x),x));
(%o30) df(x) := 6x - 6
(%i31) solve(df(x)=0,x);
(%o31) [x = 1]
```

Результат розрахунку – список, що містить один елемент (`[x = 1]`). Створюємо новий список, що включає граничні значень і критичні точки:

```
(%i32) L:[%o31[1],x=0,x=3] ;
(%o32) [x = 1, x = 0, x = 3]
```

Застосовуємо функцію $f(x)$ до кожного елемента списку L:

```
(%i33) map(f,L);
(%o33) [3x^2 - 6x = -3, 3x^2 - 6x = 0, 3x^2 - 6x = 9]
```

Результат – найбільші і найменші значення – знаходимо у списку отриманих значень.

3.4.2 Опуклість функції

Означення. *Графік функції $y = f(x)$ називається опуклим в інтервалі (a, b), якщо він розташований нижче дотичної, проведеної в будь-якій точці цього інтервалу* (див. рис. 3.4a).

Графік функції $y = f(x)$ називається ввігнутим в інтервалі (a, b), якщо він розташований вище дотичній, проведеної в будь-якій точці цього інтервалу (див. рис. 3.4b).

3.4.2.1 Необхідні і достатні умови опуклості (увігнутості) функції

Для визначення опуклості (увігнутості) функції на деякому інтервалі можна використати наступні теореми.

Теорема 1. *Нехай функція $f(x)$ визначена і неперервна на інтервалі X і має скінченну похідну $f'(x)$. Для того, щоб функція $f(x)$ була опуклою (увігнутою) в X, необхідно і достатньо, щоб її похідна $f'(x)$ спадала (зростала) на цьому інтервалі.*

Теорема 2. *Нехай функція $f(x)$ визначена і неперервна разом зі своєю похідною $f'(x)$ на X і має усередині X неперервну другу похідну $f''(x)$. Для опуклості (увігнутості) функції $f(x)$ в X необхідно і достатньо, щоб усередині X*

$$f''(x) \leqslant 0 (f''(x) \geqslant 0).$$

Доведемо теорему 2 для випадку опуклості функції $f(x)$.

Необхідність. Візьмемо довільну точку $x_0 \in X$. Розкладемо функцію $f(x)$ поблизу точки x_0 у ряд Тейлора

$$f(x) = f(x_0) + f'(x_0)(x - x_0) + r_1(x),$$

$$r_1(x) = \frac{(x - x_0)}{2} f''(x_0 + \theta(x - x_0))(0 < \theta < 1).$$

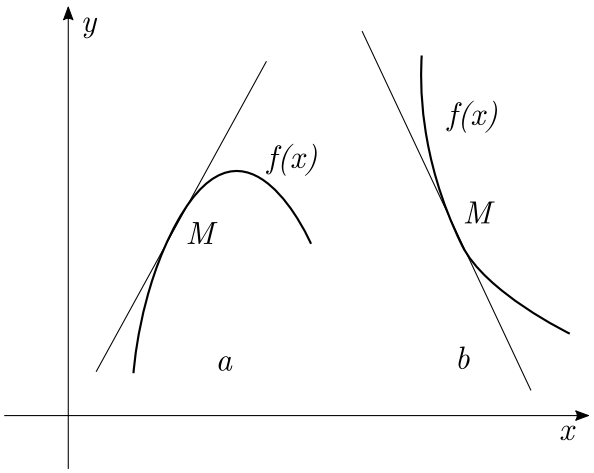


Рис. 3.4. Опуклі і увігнуті функції.

Рівняння дотичної до кривій $f(x)$ у точці, що має абсцису x_0:

$$Y(x) = f(x_0) + f'(x_0)(x - x_0).$$

Тоді перевищення кривої $f(x)$ над дотичною до неї у точці x_0 дорівнює

$$f(x) - Y(x) = r_1(x).$$

Таким чином, залишок $r_1(x)$ дорівнює величині перевищення кривої $f(x)$ над дотичною до неї у точці x_0. Оскільки $f''(x)$ є неперервною, якщо $f''(x_0) > 0$, то і $f''(x_0 + \theta(x - x_0)) > 0$ для x_0, що належать досить малому околу точки x_0, а тому, очевидно, і $r_1(x) > 0$ для будь-якого відмінного від x_0 значення x, що належить до зазначеного околу.

Виходить, графік функції $f(x)$ лежить вище дотичній $Y(x)$ і крива $f(x)$ опукла у довільній точці $x_0 \in X$.

Достатність. Нехай крива $f(x)$ опукла на проміжку X. Візьмемо довільну точку $x_0 \in X$.

Аналогічно попередньому розкладемо функцію $f(x)$ поблизу точки x_0 у ряд Тейлора

$$f(x) = f(x_0) + f'(x_0)(x - x_0) + r_1(x),$$

$$r_1(x) = \frac{(x - x_0)}{2} f''(x_0 + \theta(x - x_0))(0 < \theta < 1).$$

Перевищення кривої $f(x)$ над дотичною до неї у точці, що має абсцису x_0, що визначається виразом $Y(x) = f(x_0) + f'(x_0)(x - x_0)$, дорівнює

$$f(x) - Y(x) = r_1(x).$$

Оскільки перевищення є додатним для досить малого околу точки x_0, то додатною є і друга похідна $f''(x_0 + \theta(x - x_0))$. При прямуванні $x \to x_0$ одержуємо, що для довільної точки x_0 $f''(x_0) > 0$.

Приклад. Дослідити на опуклість (увігнутість) функцію $y = x^2 - 16x + 32$.

Її похідна $y' = 2x - 16$ зростає на всій числовій осі, значить за теоремою 1 функція увігнута на $(-\infty, \infty)$.

Її друга похідна $y'' = 2 > 0$, тому за теоремою 2 функція увігнута на $(-\infty, \infty)$.

3.4.2.2 Точки перегину

Означення. ***Точкою перегину*** *графіка неперервної функції називається точка, що розділяє інтервали, у яких функція опукла і увігнута.*

Із цього визначення випливає, що точки перегину – це точки точки екстремуму першої похідної. Звідси випливають наступні твердження для необхідної і достатньої умов перегину.

Теорема (необхідна умова перегину). *Для того щоб точка x_0 була точкою перегину двічі диференційованої функції $y = f(x)$, необхідно, щоб її друга похідна у цій точці дорівнювала нулеві ($f''(x_0) = 0$) або не існувала.*

Теорема (достатня умова перегину). *Якщо друга похідна $f''(x)$ двічі диференційованої функції $y = f(x)$ при переході через деяку точку x_0 міняє знак, то x_0 є точка перегину.*

Відзначимо, що у самій точці друга похідна $f''(x_0)$ може не існувати.

Геометричну інтерпретацію точок перегину проілюстровано на рис. 3.5.

В околі точки x_1 функція опукла і графік її лежить *нижче* дотичної, проведеної у цій точці. В околі точки x_2 функція увігнута і графік її лежить вище дотичної, проведеної у цій точці. У точці перегину x_0 дотична розділяє графік функції на області опуклості і увігнутості.

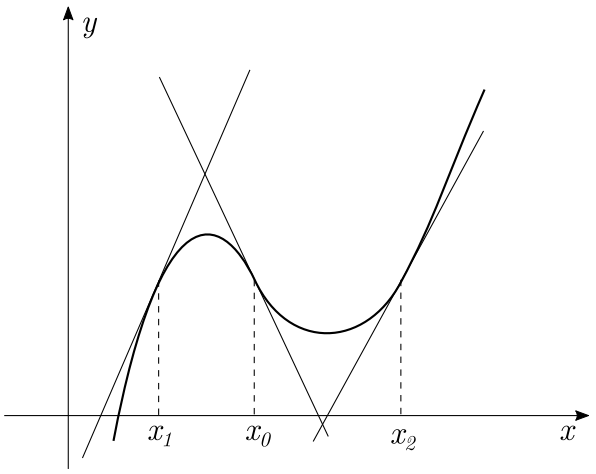


Рис. 3.5. Точки перегину.

3.4.2.3 Дослідження функції на опуклість і наявність точок перегину

1. Знайти другу похідну $f''(x)$.
2. Знайти точки, у яких друга похідна $f''(x) = 0$ або не існує.
3. Дослідити знак другій похідної ліворуч і праворуч від знайдених точок і зробити висновок про інтервали опуклості або увігнутості і наявності точок перегину.

Приклад. Дослідити функцію $y(x) = 2x^3 - 6x^2 + 15$ на опуклість і наявність точок перегину.

1. $y' = 6x^2 - 12x$; $y'' = 12x - 12$.
2. Друга похідна дорівнює нулю при $x_0 = 1$.
3. Друга похідна $y''(x)$ міняє знак при $x_0 = 1$, значить точка $x_0 = 1$ – точка перегину.

На інтервалі $(-\infty, 1)$ $y''(x) < 0$, значить функція $y(x)$ опукла на цьому інтервалі.
На інтервалі $(1, \infty)$ $y''(x) > 0$, значить функція $y(x)$ увігнута на цьому інтервалі.

3.4.2.4 Загальна схема дослідження функцій і побудови графіка

При дослідженні функції і побудові її графіка рекомендуємо використати наступну схему:

1. Знайти область визначення функції.
2. Дослідити функцію на парність-непарність. Нагадаємо, що графік парної функції симетричний щодо осі ординат, а графік непарної функції симетричний відносно початку координат.
3. Знайти вертикальні асимптоти.
4. Дослідити поведінку функції на нескінченності, знайти горизонтальні або похилі асимптоти.
5. Знайти екстремуми і інтервали монотонності функції.
6. Знайти інтервали опуклості функції і точки перегину.
7. Знайти точки перетину з осями координат.

Дослідження функції виконується одночасно з побудовою її графіка.

Приклад. Дослідити функцію $y(x) = f(x) = \dfrac{1 + x^2}{1 - x^2}$ і побудувати її графік.

1. Область визначення функції – $(-\infty, -1) \cup (-1, 1) \cup (1, \infty)$.
2. Досліджувана функція – парна $y(x) = y(-x)$, тому її графік симетричний щодо осі ординат.
3. Знаменник функції перетворюється у нуль при $x = \pm 1$, тому графік функції має вертикальні асимптоти $x = -1$ і $x = 1$.

   Точки $x = \pm 1$ є точками розриву другого роду, оскільки границі ліворуч і праворуч у цих точках дорівнюють ∞.

$$\lim_{x \to -1-0} y(x) = \lim_{x \to -1+0} y(x) = \infty; \lim_{x \to 1-0} y(x) = \lim_{x \to 1+0} y(x) = -\infty.$$

4. Поведінка функції на нескінченності.
$$\lim_{x\to\pm\infty} y(x) = -1,$$
тому графік функції має горизонтальну асимптоту $y = -1$.

5. Екстремуми і інтервали монотонності. Знаходимо першу похідну
$$y'(x) = \frac{4x}{(1-x^2)^2}.$$
$y'(x) < 0$ при $x \in (-\infty, -1) \cup (-1, 0)$, тому у цих інтервалах функція $y(x)$ спадає.
$y'(x) > 0$ при $x \in (0, 1) \cup (1, \infty)$, тому у цих інтервалах функція $y(x)$ зростає.
$y'(x) = 0$ при $x = 0$, тому точка $x_0 = 0$ є критичною точкою.
Знаходимо другу похідну
$$y''(x) = \frac{4(1+3x^2)}{(1-x^2)^3}.$$
Оскільки $y''(0) > 0$, то точка $x_0 = 0$ є точкою мінімуму функції $y(x)$.

6. Інтервали опуклості і точки перегину.
Функція $y''(x) > 0$ при $x \in (-1, 1)$, значить на цьому інтервалі функція $y(x)$ увігнута.
Функція $y''(x) < 0$ при $x \in (-\infty, -1) \cup (1, \infty)$, значить на цих інтервалах функція $y(x)$ опукла.
Функція $y''(x)$ ніде не перетворюється у нуль, значить точок перегину немає.

7. Точки перетинання з осями координат.
Рівняння $f(0) = y$, має розв'язок $y = 1$, значить точка перетину графіка функції $y(x)$ з віссю ординат $(0, 1)$.
Рівняння $f(x) = 0$ не має розв'язку, значить точок перетину з віссю абсцис немає.

З урахуванням проведеного дослідження можна будувати графік функції
$$y(x) = \frac{1+x^2}{1-x^2}.$$
Схематично графік функції зображений на рис. 3.6.

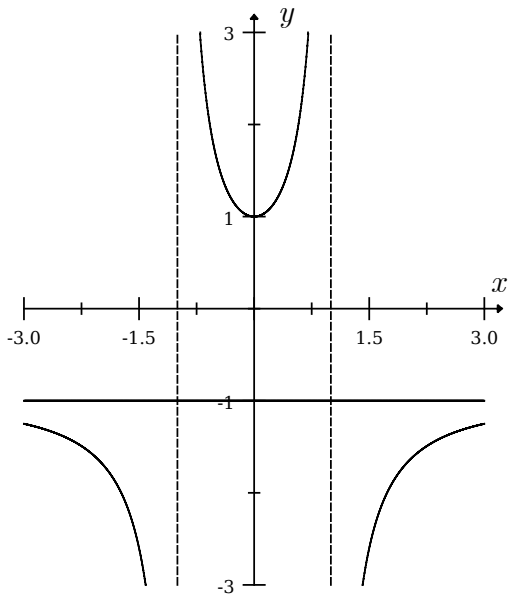
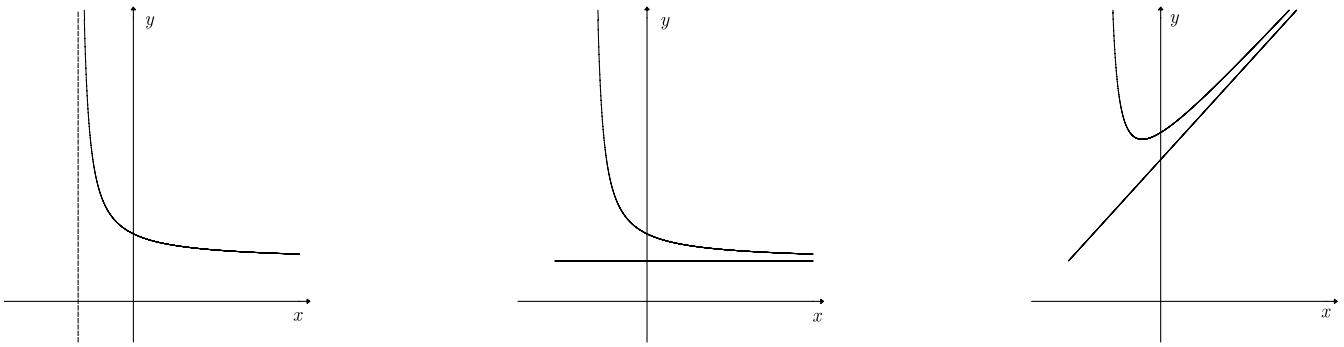


Рис. 3.6. Графік функції $y(x) = \frac{1+x^2}{1-x^2}$

3.4.2.5 Асимптоти графіка функції

Означення. ***Асимптотою*** *графіка функції $y = f(x)$ називається пряма, що має таку властивість, що відстань від точки $(x, f(x))$ до цієї прямої прямує до 0 при необмеженому видаленні точки графіка від початку координат.*

Асимптоти бувають 3 типів: вертикальні (див. рис. 3.7a), горизонтальні (див. рис. 3.7b) і похилі (див. рис. 3.7c).
Асимптоти знаходять, використовуючи наступні теореми:



a *b* *c*

Рис. 3.7. Асимптоти

Теорема 1. *Нехай функцію $y = f(x)$ визначено у деякій околі точки x_0 (крім, можливо, самої цієї точки) і хоча б одна з границь функції при $x \to x_0 - 0$ (ліворуч) або $x \to x_0 + 0$ (праворуч) дорівнює нескінченності. Тоді пряма $x = x_0$ є вертикальною асимптотою графіка функції $y = f(x)$.*

Вертикальні асимптоти $x = x_0$ слід шукати у точках розриву функції $y = f(x)$.

Теорема 2. *Нехай функцію $y = f(x)$ визначено при досить великих x і існує скінченна границя функції на нескінченності $\lim\limits_{x\to\pm\infty} f(x) = b$. Тоді пряма $y = b$ є горизонтальна асимптот графіка функції $y = f(x)$.*

Теорема 3. *Нехай функція $y = f(x)$ визначена при досить великих x і існують скінченні границі*

$$\lim_{x\to\pm\infty} \frac{f(x)}{x} = k$$

і

$$\lim_{x\to\pm\infty} [f(x) - kx] = b.$$

Тоді пряма $y = kx + b$ є похилою асимптотою графіка функції $y = f(x)$.

Приклад. Знайти асимптоти графіка дробово-раціональної функції

$$y(x) = \frac{ax + b}{cx + d}; c \neq 0; ad - bc \neq 0.$$

Якщо $c = 0$, то дробово-раціональна функція стає лінійною

$$y(x) = \frac{a}{d} + \frac{b}{d}$$

Особлива точка – $x = -d/c$. Знайдемо границю $\lim\limits_{x\to -d/c} f(x)$.

Перепишемо дробово-раціональну функцію у вигляді:

$$y(x) = \frac{ax + b}{c(x + d/c)}$$

Оскільки $ad - bc = 0$, то при $x \to -d/c$ чисельник дробово-раціональної функції не прямує до нуля. Тому пряма $x = -d/c$ — асимптота графіка дробово-раціональної функції.

Знайдемо границю $\lim\limits_{x\to\pm\infty} f(x)$.

$$\lim_{x\to\pm\infty} \frac{ax + b}{cx + d} = \lim_{x\to\pm\infty} \frac{a + b/x}{c + d/x} = \frac{a}{c}$$

$y = a/c$ є горизонтальної асимптотою дробово-раціональної функції.

Приклад. Знайти асимптоти кривої $y(x) = \dfrac{x^3}{x^2 + 1}$. Тому $k = 1$. Тепер шукаємо b.

$$\lim_{x\to\pm\infty} \left[\frac{x^3}{x^2 + 1} - x\right] = \lim_{x\to\pm\infty} \left[\frac{-x}{x^2 + 1}\right] = 0$$

Функція $y(x) = \dfrac{x^3}{x^2 + 1}$ має похилу асимптоту $y = x$.

3.4.2.6 Властивості функцій, неперервних на відрізку. Теореми Веєрштраса

1. *Якщо функція $y = f(x)$ неперервна на відрізку $[a, b]$, то вона обмежена на цьому відрізку, тобто існують такі сталі і скінченні числа m і M, що*
$$m < f(x) < M \text{ при } a < x < b.$$
2. *Якщо функція $y = f(x)$ неперервна на відрізку $[a, b]$, то вона досягає на цьому відрізку найбільшого значення M і найменшого значення m.*
3. *Якщо функція $y = f(x)$ неперервна на відрізку $[a, b]$, і значення її на кінцях відрізка $f(a)$ і $f(b)$ мають протилежні знаки, то усередині відрізка найдеться точка $\xi \in (a, b)$, така, що $f(\xi) = 0$.*

3.4.3 Диференціювання функцій декількох змінних

Для визначення набору частинних похідних функції декількох змінних (компонентів градієнта) використовується функція `gradef` у форматі `gradef(f(`x_1`, ..., `x_n`), `g_1`, ..., `g_m`)` або `gradef(a, x, expr)`.

Вираз `gradef(f(`x_1`, ..., `x_n`), `g_1`, ..., `g_m`)` визначає g_1, g_2,... g_n як частинні похідні функції `f(`x_1`, `x_2`, ..., `x_n`)` за змінними x_1, x_2, ..., x_n відповідно.

Залежності між змінними можна явно вказати за допомогою функції `depends`, що надає змогу декларувати, що змінна залежить від однієї або декількох інших змінних. Наприклад, якщо залежності f і x немає, вираз `diff(f, x)` повертає 0. Якщо декларувати її за допомогою `depends(f, x)`, вираз `diff(f, x)` повертає символьну похідну.

Приклад:

```
(%i1) depends(y,x);
(%o1) [y(x)]
(%i2) gradef(f(x,y),x^2,g(x,y));
(%o2) f(x,y)
(%i3) diff(f(x,y),x);
```
(%o3) $g(x,y)\left(\frac{d}{dx}y\right)+x^2$
```
(%i4) diff(f(x,y),y);
```
(%o4) $g(x,y)$

Інша форма звертання до `gradef` фактично встановлює залежність a від x. За допомогою `gradef` можна визначити похідні деякої функції, навіть якщо вона сама невідома, за допомогою `diff` визначити похідні вищих порядків.

Для прямих обчислень, пов'язаних з операціями векторного аналізу, слід завантажити пакет `vect`. Крім того, для застосування операторів `div`, `curl`, `grad`, `laplasian` до деякого виразу використовується функція `express`.

Приклад: Обчислення градієнта функції трьох змінних

```
(%i2) grad(x^2 + 2*y^2 + 3*z^2);
(%o2) grad(3z^2+2y^2+x^2)
(%i3) express(%);
```
(%o3) $[\frac{d}{dx}(3z^2+2y^2+x^2),\ \frac{d}{dy}(3z^2+2y^2+x^2),\ \frac{d}{dz}(3z^2+2y^2+x^2)]$
```
(%i4) ev(%,diff);
```
(%o4) $[2x,\ 4y,\ 6z]$

Обчислення дивергенції

```
(%i5) div([x^2,2*y^2,3*z^2]);
(%o5) div([x^2, 2y^2, 3z^2])
(%i6) express(%);
```
(%o6) $\frac{d}{dz}(3z^2)+\frac{d}{dy}(2y^2)+\frac{d}{dx}x^2$
```
(%i7) ev(%,diff);
```
(%o7) $6z+4y+2x$

Обчислення вихору:

```
(%i8) curl([x^2,2*y^2,3*z^2]);
(%o8) curl([x^2, 2y^2, 3z^2])
(%i9) express(%);
```
(%o9) $[\frac{d}{dy}(3z^2)-\frac{d}{z}(2y^2),\ \frac{d}{dz}x^2-\frac{d}{dx}(3z^2),\ \frac{d}{dx}(2y^2)-\frac{d}{dy}x^2]$
```
(%i10) ev(%,diff);
```
(%o10) $[0,\ 0,\ 0]$

Обчислення оператора Лапласа:

```
(%i13) laplacian(x^2+2*y^2+3*z^2);
(%o13) laplacian (3 z^2 + 2 y^2 + x^2)
(%i14) express(%);
(%o14) d^2/dz^2 (3 z^2 + 2 y^2 + x^2) + d^2/dy^2 (3 z^2 + 2 y^2 + x^2) + d^2/dx^2 (3 z^2 + 2 y^2 + x^2)
(%i15) ev(%,diff);
(%o15) 12
```

За допомогою пакета `vect` можна також обчислювати векторний добуток векторів (для цього використовується символ $\sim$):

```
(%i1) load(vect)$
(%i2) a:[1,2,-1]$ b:[3,4,0]$
(%i4) a~b$
(%i5) express(%);
(%o5) [4, −3, −2]
```

Розгляньмо приклад дослідження функції декількох змінних: дослідити на екстремум функцію $f(x,y) = y^2 - 4y + x^3 - \frac{9x^2}{2} + 6x - 12$.

Завантажуємо пакет `vect`

```
(%i1) load(vect)$
```

Визначаємо досліджуваний вираз і обчислюємо його градієнт:

```
(%i2) f:x^3-9/2*x^2+6*x+y^2-4*y-12;
(%o2) y^2 − 4y + x^3 − 9x^2/2 + 6x − 12
(%i3) grad(f);
(%o3) grad (y^2 − 4y + x^3 − 9x^2/2 + 6x − 12)
(%i4) express(%);
(%o4) [d/dx (y^2 − 4y + x^3 − 9x^2/2 + 6x − 12), d/dy (y^2 − 4y + x^3 − 9x^2/2 + 6x − 12), d/dz (y^2 − 4y + x^3 − 9x^2/2 + 6x − 12)]
(%o5) [3x^2 − 9x, 2y − 4, 0]
```

Виокремлюємо з отриманого списку частинні похідні і розв'язуємо систему $f_x(x,y) = 0;\ f_y(x,y) = 0$

```
(%i6) dfdx:%o5[1];
(%o6) 3x^2 − 9x + 6
(%i7) dfdy:%o5[2];
(%o7) 2y − 4
(%i8) solve([dfdx=0,dfdy=0],[x,y]);
(%o8) [[x = 1, y = 2], [x = 2, y = 2]]
```

У результаті розв'язання знаходимо дві критичні точки $M_1(1,2)$ і $M_2(2,2)$. Для перевірки, чи досягається у критичних точках екстремум, використаємо достатню умову екстремуму:

```
(%i9) A:diff(dfdx,x);
(%o9) 6x − 9
(%i10) C:diff(dfdy,y);
(%o10) 2
(%i11) B:diff(dfdx,y);
(%o11) 0
(%i12) A*C-B^2;
(%o12) 2(6x − 9)
```

Оскільки $AC - B^2 > 0$ тільки у точці $M_2(2,2)$, то досліджувана функція має єдиний екстремум. З огляду на, що у точці $M_2(2,2)$ $A > 0$, точка M_2 – точка мінімуму. Результат ілюструємо графічно (рис. 3.8).

3.5 Аналітичне і обчислювальне інтегрування

3.5.1 Основні команди

Невизначений інтеграл $\int f(x)\mathrm{d}x$ обчислюється за допомогою команди `integrate(f, x)`, де `f` – підінтегральная функція, `x` – змінна інтегрування.

Для обчислення визначеного інтеграла $\int\limits_a^b f(x)\mathrm{d}x$ у команді `integrate` додаються межі інтегрування, наприклад,

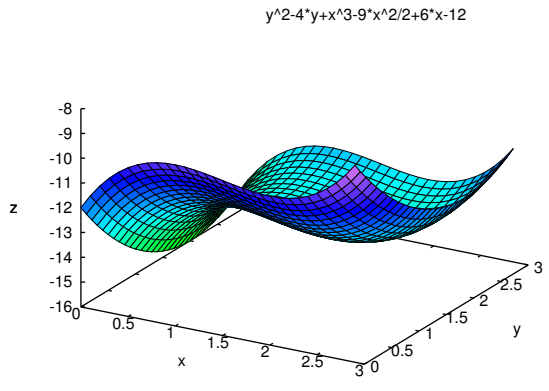


Рис. 3.8. Пошук екстремуму функції декількох змінних

```
integrate((1+cos(x))^2, x, 0, %pi);
```

$$\int_0^{\pi} (1+\cos x)^2 dx = \frac{3\pi}{2}$$

Невласні інтеграли з нескінченними межами інтегрування обчислюються, якщо у параметрах команди `integrate` указувати, наприклад, `x, 0, %inf`.

Числове інтегрування виконується функцією `romberg` (див. нижче) або за допомогою функцій пакета `quadpack`.

3.5.2 Інтеграли, що залежать від параметра. Обмеження для параметрів

Якщо потрібно обчислити інтеграл, що залежить від параметра, то його значення може залежати від знака цього параметра або якихось інших обмежень. Розгляньмо як приклад інтеграл $\int_{+\infty} 0e^{-ax}dx$, який, як відомо з математичного аналізу, збігається при $a > 0$ і розбігається при $a < 0$. Якщо обчислити його відразу, то вийде:

```
(%i1) integrate(exp(-a*x),x,0,inf);
Is a positive, negative, or zero? p;
```
(%o1) $\dfrac{1}{a}$

Результат аналітичного інтегрування:

$$\int_0^{+\infty} e^{-ax}dx = \lim_{x\to\infty} -\frac{e^{-ax}-1}{a} = \frac{1}{a}.$$

Для одержання явного аналітичного результату обчислень варто зробити якісь припущення про значення параметрів, тобто накласти на них обмеження. Це можна зробити за допомогою команди `assume(`$expr_1$`)`, де $expr_1$ – нерівність.

Опис накладених обмежень параметра a можна викликати командою `properties(a)`.

```
(%i1) assume (a > 1)$ integrate (x**a/(x+1)**(5/2), x, 0, inf);
```
Is $\dfrac{2a+2}{5}$ an integer? no;
Is $2a-3$ positive, negative, or zero? neg;
(%o2) $\beta\left(a+1,\ \frac{3}{2}-a\right)$
```
(%i3) properties(a);
```
(%o3) $[database\ info,\ a > 1]$

Повернемося до обчислення інтеграла з параметром $\int_0^{+\infty} e^{-ax}dx$, яких слід виконувати у такому порядку:

```
(%i1) assume(a>0); integrate(exp(-a*x),x,0,inf);
```
(%o1) $[a > 0]$ (%o2) $\frac{1}{a}$

Скасувати прийняті обмеження на значення параметрів можна, використовуючи функцію `forget`.

Приклад:

```
(%i1) assume(n+1>0); integrate((a+b)*x^(n+1),x);
```
(%o1) $[n > -1]$ (%o2) $\frac{(b+a)x^{n+2}}{n+2}$

Скасування обмеження спричиняє питання про значення параметрів підінтегральної функції:

```
(%i3) forget(n+1>0); integrate((a+b)*x^(n+1),x);
```
(%o3) $[n > -1]$
```
Is n + 2 zero or nonzero? zero;
```
(%o4) $(b+a)log(x)$

Отриманий результат зовсім інший!

3.5.3 Основні прийоми інтегрування

У **Maxima** є функція, призначена для виконання розрахунків крок за кроком, що здійснює заміну змінної, `changevar`.

Формулу інтегрування частинами:

$$\int u(x)v'(x)dx = u(x)v(x) - \int u'(x)v(x)dx$$

доведеться застосовувати вручну. У **Maxima** (на відміну від, наприклад, **Maple**), функція інтегрування частинами не виділена явно, хоча в окремих випадках цей спосіб використовується `integrate`.

Для обчислення первісних диференціальних виразів використовується пакет `antid` (основні функції пакета – `antidiff` і `antid`). Функція `antidiff` виконує інтегрування виразів із довільними функціями (у тому числі невизначеними), перед її першим викликом варто завантажити пакет (`antid` відрізняється від її форматом виведеного результату).

Приклад:

```
(%i1) load("antid");
(%i2) expr: exp(z(x))*diff(z(x),x)*sin(x);
(%o2) e^z(x) sin(x)(d/dx z(x))
(%i3) a1: antid(expr, x, z(x));
(%o3) [e^z(x) sin(x), -e^z(x) cos(x)]
```

За допомогою пакета `antid` можна виконати формальне інтегрування частинами, наприклад:

```
(%i1) expr:u(x)*diff(v(x),x);
(%o1) u(x) (d/dx v(x))
(%i2) a:antid(expr,x,v(x));
(%o2) antid (u(x) (d/dx v(x)), x, v(x))
(%I3) b:antidiff(expr,x,v(x));
(%o3) antidiff (u(x) (d/dx v(x)), x, v(x))
```

Якщо в інтегралі потрібно зробити заміну змінних, використовується функція `changevar`.

Синтаксис виклику цієї функції: `changevar(`*expr*`, f(x,y), y, x)`. Функція здійснює заміну змінної відповідно до рівняння `f(x, y)=0` у всіх інтегралах, що зустрічаються у виразі *expr* (передбачається, що y – нова змінна, x – вихідна). При використанні разом з `changevar` часто використовується відкладене обчислення інтеграла (одинарні лапки перед функцією `integrate`).

Приклад:

```
(%i5) assume(a > 0)$ 'integrate (%e**sqrt(a*y), y, 0, 4);
(%o6) ∫_0^4 e^(sqrt(a) sqrt(y)) dy
```

Цей інтеграл не обчислюється аналітично безпосередньо, тому виконуємо заміну:

```
(%i7) changevar (%, y-z^2/a, z, y);
```

$$(\%o7)\ -\frac{2\int_{-2\sqrt{a}}^{0} z e^{|z|}}{a}$$

Початковий інтеграл було записано з ознакою відкладеного обчислення, тому приводимо результат у «завершену» форму (виконуємо `ev` із ключем `nouns`).

```
(%i8) ev(%,nouns);
```

$$(\%o8)\ -\frac{2\left(-2\sqrt{a}e^{2\sqrt{a}} + e^{2\sqrt{a}} - 1\right)}{a}$$

Не завжди можна обчислити інтеграл (як визначений, так і невизначений) до кінця лише за рахунок використання функції `integrate`. У цьому випадку функція повертає вираз з відкладеним обчисленням вкладеного (можливо, простішого за формою) інтеграла.

Приклад:

```
(%i10) expand((x-4)*(x^3+2*x+1));
(%o10) x^4 - 4x^3 + 2x^2 - 7x - 4
(%i11) integrate (1/%, x);
```

Не знаючи коренів знаменника, неможливо повністю обчислити інтеграл від раціонального виразу, тому один з компонентів результату – невизначений інтеграл, для остаточного обчислення якого необхідно знайти корінь знаменника (наприклад, використовуючи `allroots`).

$$(\%o11)\ \frac{\log(x-4)}{73} - \frac{\int \frac{x^2+4x+18}{x^3+2x+1}dx}{73}$$

Можливим розв'язанням є спрощення інтеграла, що супроводжується зниженням степеня раціонального виразу у знаменнику. При цьому необхідно встановити у `true` значення змінної `integrate_use_rootsof`. Однак при цьому результат може бути досить важко інтерпретувати.

Розгляньмо попередній приклад, виконавши попередньо факторизацію знаменника:

```
(%i1) f:expand ((x-4) * (x^3+2*x+1));
(%o1) x^4-4*x^3+2*x^2-7*x-4
(%i2) polyfactor:true$ ffact:allroots(f);
(%o3) 1.0*(x-3.999999999999997)*(x+0.4533976515164)*(x^2-0.45339765151641*x+2.205569430400593)
(%i4) float(integrate(1/ffact,x));
```

Отриманий результат однаково важко назвати однозначно прийнятним, тому що він включає одночасно дуже великі і дуже малі величини. Причина у тім, що корені знаменника представлялися раціональними числами. Для того, щоб одержати компактний результат, бажано для коефіцієнтів вигляду $r = \frac{m}{n}$ зменшити m і n.

Інтеграли від тригонометричних і логарифмічних функцій **Maxima** обчислює досить успішно. Розгляньмо кілька прикладів.

```
(%i1) integrate(sin(x)*sin(2*x)*sin(3*x),x);
(%o1) cos(6*x)/24-cos(4*x)/16-cos(2*x)/8
(%i2) integrate(1/cos(x)^3,x);
(%o2) log(sin(x)+1)/4-log(sin(x)-1)/4-sin(x)/(2*sin(x)^2-2)
(%i3) integrate(x^3*log(x),x);
(%o3) x^4*log(x)/4-x^4/16
```

3.5.4 Перетворення Лапласа

Пряме і обернене перетворення Лапласа обчислюються за допомогою функцій `laplace` і `ilt` відповідно.

Синтаксис виклику функції `laplace`: `laplace(`*expr*`, t, s)`.

Функція обчислює перетворення Лапласа виразу *expr* відносно змінної t. Образ виразу *expr* буде включати змінну s.

Функція `laplace` розпізнає у виразі *expr* функції `delta`, `exp`, `log`, `sin`, `cos`, `sinh`, `cosh`, і `erf`, а також похідні, інтеграли, суми і обернене перетворення Лапласа (`ilt`). При наявності інших функцій обчислення перетворення може і не вдатися.

Крім того, обчислення перетворення Лапласа можливе і для диференціальних рівнянь і інтегралів типу згортки.

```
(%i1) laplace(c,t,s);
(%o1) c/s
(%i2) laplace(erf(t),t,s);
(%o2) %e^(s^2/4)*(1-erf(s/2))/s
(%i3) laplace(sin(t)*exp(-a*t),t,s);
(%o3) 1/(s^2+2*a*s+a^2+1)
```

Функція `ilt(`*expr*`, t, s)` обчислює зворотне перетворення Лапласа щодо змінної `t` з параметром `s`.

Приклад:

```
(%i1) laplace(c,t,s);
(%o1) c/s
(%i2) ilt( %,s,t);
(%o2) c
(%i3) laplace(sin(2*t)*exp(-4*t),t,s);
(%o3) 2/(s^2+8*s+20)
(%i4) ilt(%,s,t);
(%o4) %e^(-4*t)*sin(2*t)
```

3.6 Методи теорії наближення в обчислювальному аналізі

Курс вищої математики для студентів технічних вишів містить первинні основи обчислювальних методів як свою складову частину. Для фахівців інженерного профілю вкрай важливим є одночасне знаходження розв'язку у замкненій аналітичній формі і одержання обчислювальних значень результату. Подання функції у вигляді степеневого ряду

надає змогу звести вивчення властивостей функції, наближення якої виконується, до простішої задачі вивчення цих властивостей у відповідного поліноміального розкладу апроксимації.

Цим пояснюється важливість усіляких аналітичних і обчислювальних додатків поліноміальних наближень для апроксимації і обчислення функції. Заміна функцій на їхні степеневі розклади і поліноміальні наближення допомагає вивченню границь, аналізу збіжності і розбіжності рядів та інтегралів, наближеному обчисленню інтегралів і розв'язуванню диференціальних рівнянь. Степеневі ряди і розклади за многочленами Чебишова широко використовуються при обчисленні значень функції із заданим порядком точності. Вони є ефективним обчислювальним засобом при розв'язуванні широкого кола науково-технічних задач.

3.6.1 Наближене обчислення математичних функцій

Нехай функція $f(x)$ задана на інтервалі $(x_0 - R, x_0 + R)$ і нам потрібно обчислити значення функції $f(x)$ при $x = x_1 \in (x_0 - R, x_0 + R)$ із заданою точністю $\varepsilon > 0$.

Припустивши, що функція $f(x)$ в інтервалі $x \in (x_0 - R, x_0 + R)$ розкладається у степеневий ряд

$$f(x) = \sum_{i=0} \infty u_i(x) = \sum_{i=0}^{\infty} a_i(x - x_0)^i = a_0 + a_1(x - x_0) + a_2(x - x_0)^2 + \cdots + a_n(x - x_0)^n + \dots$$

ми отримаємо, що точне значення $f(x_i)$ дорівнює сумі цього ряду при $x = x_1$

$$f(x_1) = \sum_{i=0}^{\infty} a_i(x_1 - x_0)^i = a_0 + a_1(x_1 - x_0) + a_2(x_1 - x_0)^2 + \cdots + a_n(x_1 - x_0)^n + \dots,$$

а наближене – частковій сумі $S_n(x_1)$

$$f(x_1) \approx S_(x_1) = \sum_{i=0} n a_i(x_1 - x_0)^i = a_0 + a_1(x_1 - x_0) + a_2(x_1 - x_0)^2 + \cdots + a_n(x_1 - x_0)^n.$$

Для похибки наближення ми маємо вираз у вигляді залишку ряду

$$f(x_1) - S_n(x_1) = r_n(x_1),$$

де

$$r_n(x_1) = \sum_{i=1} \infty x_1^{n+i} = a_{n+1}x_1^{n+1} + a_{n+2}X_1^{n+2} + \dots$$

Для знакочергових рядів із послідовно спадними членами

$$|r_n(x)| = \left| \sum_{i=1}^{\infty} u_{n+i}(x_1) \right| < |u_{n+1}(x_1)|$$

Точність апроксимації, як правило, зростає зі зростанням степеня розкладу наближення, і покращується із наближенням точки x_1 до точки x_0. Для рівномірної апроксимації на інтервалі найбільш зручними виявляються розклади за поліномами Чебишова.

Для наближеного знаходження значень функції за допомогою степеневих рядів, як правило, використовуються її розклад у вигляді рядів Тейлора.

Ряд Тейлора для функції $f(x)$ – це степеневий ряд вигляду

$$\sum_{k=0}^{\infty} \frac{f^{(k)}(x_0)}{k!} (x - x_0)^k,$$

де числова функція f вважається визначеною у деякому околі точки x_0, що має у цій точці похідні всіх порядків.

Многочленами Тейлора для функції $f(x)$, порядку n відповідно, називаються частинні суми ряду Тейлора

$$\sum_{k=0}^{n} \frac{f^{(k)}(x_0)}{k!} (x - x_0)^k.$$

Якщо ми розпишемо цю формулу, то одержимо наступний вираз

$$f(x_0) + \frac{f'(x_0)}{1!} (x - x_0) + \frac{f''(x_0)}{2!} (x - x_0)^2 + \cdots + \frac{f^{(n)}(x_0)}{n!} (x - x_0)^n.$$

Формула Тейлора для функції $f(x)$ – це подання функції у вигляді суми її многочлена Тейлора степеня n ($n = 0, 1, 2, \dots$) і залишкового члена. Іншими словами це називають розкладом функції $f(x)$ за формулою Тейлора в околі точки x_0. Якщо дійсна функція f однієї змінної має n похідних у точці x_0, то її формула Тейлора має вигляд

$$f(x) = P_n(x) + r_n(x),$$

де

$$P_n(x) = \sum_{k=0}^{n} \frac{f^{(k)}(x_0)}{k!} (x - x_0)^k$$

– многочлен Тейлора степеня n, а залишковий член може бути записаний у формі Пеано

$$r_n(x) = o\left((x - x_0)^n\right), x \to x_0.$$

Отримуємо, що

$$P_n(x) = f(x_0) + \frac{f'(x_0)}{1!} (x - x_0) + \frac{f''(x_0)}{2!} (x - x_0)^2 + \cdots + \frac{f^{(n)}(x_0)}{n!} (x - x_0)^n .$$

Якщо функція f диференційована $n+1$ разів у деякому околі точки x_0, $(x_0 - \delta, x_0 + \delta)$, $\delta > 0$, то залишковий член у цій околі може бути записаний у формі Лагранжа

$$r_n(x) = \frac{f^{(n+1)}(x_0 + \theta(x - x_0))}{(n+1)!}(x - x_0)^{n+1},$$

$$0 < \theta < 1, x \in (x_0 - \delta, x_0 + \delta)$$

Зазначимо, що при $n = 1$ вираз для $P_1(x) = f(x_0) + f'(x_0)(x - x_0)$ збігається із формулою Лагранжа скінченних приростів для функції $f(x)$.

Формула Тейлора для многочленів. Нехай є довільний многочлен $f(x) = a_0x^n + a_1x^{n-1} + \cdots + a_n$. Тоді за будь-яких x і h має місце наступна формула:

$$f(x + h) = a_0(x + h)^n + a_1(x + h)^{n-1} + \cdots + a_n =$$
$$= f(x) + f'(x)h + \frac{f''(x)}{2!}h^2 + \cdots + \frac{f^{(k)}(x)}{k!}h^k + \cdots + \frac{f^{(n)}(x)}{n!}h^n$$

Рядом Маклорена для функції $f(x)$ називається її ряд Тейлора у точці 0 початку координат, тобто це степеневої ряд вигляду

$$f(x) = \sum_{k=0}^{\infty} \frac{f^{(k)}(0)}{k!}x^k.$$

У такий спосіб формула Маклорена є частинним випадком формули Тейлора. Припустимо, що функція $f(x)$ має n похідних у точці $x = 0$. Тоді у деякому околі цієї точки $(-\delta, \delta)$, $\delta > 0$, функцію $f(x)$ можна представити у вигляді

$$f(x) = \sum_{k=0}^{n} \frac{f^{(k)}(0)}{k!}x^k + r_n(x), x \in (-\delta, \delta),$$

де $r_n(x)$ – залишковий член n-го порядку у формі Пеано.

Наведемо розклади за формулою Маклорена для основних елементарних математичних функцій:

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots + \frac{x^n}{n!} + o(x^n),$$

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} + \cdots + (-1)^n \frac{x^{2n-1}}{(2n-1)!} + o(x^{2n}),$$

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} + \cdots + (-1)^n \frac{x^{2n}}{(2n)!} + o(x^{2n+1}),$$

$$(1 + x)^\alpha = 1 + \alpha x + \frac{\alpha(\alpha - 1)}{2!}x^2 + \cdots + \frac{\alpha(\alpha - 1)\ldots(\alpha - n + 1)}{n!}x^n + o(x^n),$$

$$\ln(1 + x) = x - \frac{x^2}{2} + \frac{x^3}{3} + \cdots + (-1)^{n-1} \frac{x^n}{n} + o(x^n).$$

У **Maxima** існує спеціальна команда, що надає змогу обчислювати ряди і многочлени Тейлора: `taylor(`*expr*`, x, a, n)`. Тут *expr* – вираз, що розкладається у ряд, `a` – значення `x`, в околі якого виконується розклад (за степенями `x - a`), `n` – параметр, що вказує на порядок розкладу і представлений цілим додатним числом. Якщо `a` вказується просто у вигляді імені змінної, то виконується обчислення ряду і многочлена Маклорена.

Приклад: Знайти многочлен Тейлора 9-ої степеня експоненційної функції e^x у початку координат.

```
(%i1) taylor(exp(x),x,0,9);
```

(%o1) $1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \frac{x^4}{24} + \frac{x^5}{120} + \frac{x^6}{720} + \frac{x^7}{5040} + \frac{x^8}{40320} + \frac{x^9}{362880} + \ldots$

Многочлени Тейлора дають найбільш точну апроксимацію наближуваної функції поблизу точки x_0. У міру видалення від точки x_0 похибка зростає. Для наближення доводиться використовувати многочлени Тейлора вищого степеня, але іноді і вони не допомагають через нагромадження обчислювальної похибки.

Цікаво простежити цей процес графічно. Пакет **Maxima** надає таку можливість за допомогою команди `plot`.

Приклад: Знайти число e з точністю до 0.001. Покладемо $x = 1$. Тоді щоб обчислити значення e, необхідно виконати серію команд:

Будуємо розклад функції e^x у ряд Тейлора (до 8 порядку включно)

```
(%i1) t:taylor(exp(x),x,0,8);
```
(%o1) $1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \frac{x^4}{24} + \frac{x^5}{120} + \frac{x^6}{720} + \frac{x^7}{5040} + \frac{x^8}{40320} + \dots$

Обчислюємо часткову суму ряду при $x = 1$:

```
(%i2) ev(t,x=1);
```
(%o2) $\frac{109601}{40320}$

Значення e у формі із рухомою крапкою знаходимо, використовуючи функцію `float`:

```
(%i3) float(%);
(%o3) 2.71827876984127
```

Цікаво виконати обчислення і порівняти результати, що виходять для числа e при різних степенях використовуваного многочлена Тейлора. Виходять такі результати:

$k = 1$, $e_1 = 1$, $k = 2$, $e_2 = 2$, $k = 3$, $e_3 = 2.5$, $k = 4$, $e_4 = 2.666666667$, $k = 5$, $e_5 = 2.708333333$, $k = 6$, $e_6 = 2.716666667$, $e_7 = 2.718055556$, $k = 8$, $e_8 = 2.718253968$, $k = 9$, $e_9 = 2.718281526$, $e_10 = 2.718281801$.

Звідси видно, що значення e з точністю 0.001 обчислюється при використанні многочлена Тейлора степеня не нижче 7-го. Також треба, що число e з точністю 0.000001 або, що те саме, 10^{-6} обчислюється за допомогою многочлена Тейлора 9-го або вищого степеня.

Оцінку залишку ряду зробимо за формулою залишкового члена ряду Маклорена

$$|f(x_1) - s_n(x_1)| = |r_n(x_1)| = \left|\frac{f^{(n+1)}(c)}{(n+1)!}\right|,$$

де c перебуває між 0 і x_1. Отже, $r_n(1) = \frac{e^c}{(n+1)!}$, $0 < c < 1$. Оскільки $e^c < e < 3$, то $r_n(1) < \frac{3}{(n+1)!}$. При $n = 7$ маємо $r_7 < \frac{3}{7!} < 0.001$, $e \approx 2.718$.

Поряд з командою `taylor` для розкладу функцій і виразів у ряди використовується команда `powerseries(вираз, x, a)` (будується розклад для заданого виразу за змінною `x` в околі `a`). Результатом виконання команди `powerseries` може бути побудова ряду Тейлора для функції у загальній формі, наприклад:

```
(%i1) powerseries(sin(x),x,0);
```
(%o1) $\sum_{i1=0}^{\infty} \frac{(-1)^{i1} \cdot x^{2 \cdot i1+1}}{(2 \cdot i1+1)!}$

```
(%i2) powerseries(sin(x^2),x,0);
```
(%o2) $\sum_{i2=0}^{\infty} \frac{(-1)^{i2} \cdot x^{2(2 \cdot i1+1)}}{(2 \cdot i2+1)!}$

Для отримання розкладу у ряд Тейлора функції декількох змінних використовується функція taylor із вказування списку змінних у формі: `taylor(expr, [x1, x2, ... ], [a1, a2, ... ], [n1, n2,. .. ])`

Приклад: Знайти многочлен Тейлора 6-го степеня від функції $\frac{x}{1+x}$.

```
(%i1) f(x):=x/(1+x);
```
(%o1) $f(x) := \frac{x}{1+x}$

```
(%i2) powerseries(f(x),x,0);
```
(%o2) $x \sum_{i1=0}^{\infty} (-1)^{i1} x^{i1}$

```
(%i3) taylor(f(x),x,0,6);
```
(%o3) $x - x^2 + x^3 - x^4 + x^5 - x^6 + \dots$

Приклад: Знайти розкладу функції $\arccos x$ у ряд Маклорена.

```
(%i6) taylor(acos(x),x,0,12);
```
(%o6) $\frac{-63}{2816} \cdot x^{11} + \frac{-35}{1152} \cdot x^9 + \frac{-5}{112} \cdot x^7 + \frac{-3}{40} \cdot x^5 + \frac{-1}{6} \cdot \ x^3 + (-1) \cdot x + \frac{1}{2} \cdot \pi + ...$

Приклад: Знайти розклад функції $\exp(x) + 1$ за формулою Тейлора 5-го степеня в околі точки $x = 2$.

```
(%i7) taylor(exp(x)+1,x,2,5);
```
(%o7) $1 + e^2 + e^2(x-2) + \frac{e^2(x-2)^2}{2} + \frac{e^2(x-2)^3}{6} + \frac{e^2(x-2)^4}{24} + \frac{e^2(x-2)^5}{120} + \dots$

Приклад: Знайти розклад гіперболічного косинуса у ряд Маклорена 8-го степеня.

```
taylor(cosh(x),x,10);
```

Одержуємо

$$1+\frac{1}{2}x^2+\frac{1}{24}x^4+\frac{1}{720}x^6+\frac{1}{40320}x^8+O(x^10).$$

Зазначимо, що для аналітичних функцій розклад у ряд Тейлора існує завжди. Наведемо приклад функції, що не має розкладу у ряд Тейлора і для якої команда taylor не дає результату: $f(x)=1/x^2+x$.

```
(%i8) taylor(1/x^2+x,x,0,7);
(%o8) 1/x^2 + x + ...
```

У результаті виконання команди `taylor` або `powerseries` одержуємо вихідний вираз $x^{-2}+x$. У той же час в околі інших точок, наприклад точки $x=2$, формула Тейлора обчислюється

```
(%i13) taylor(1/x^2+x,x,2,2);
(%o13) (2 2^2+1)/2^2 - ((2-2 2^2) (x-2))/(2 2^2) + ((2^2+2) (x-2)^2)/(8 2^2) + ...
(%i14) ratsimp(%);
(%o14) 2^(-2-3) ((2^2+2) x^2 + (-4 2^2-8 2+2^2+3) x + 4 2^2+12 2+8)
```

Пакет **Maxima** дає можливість як знаходження розкладів математичних функцій у ряди Тейлора, так і графічної інтерпретації точності цих розкладів. Подібна графічна візуалізація допомагає у розумінні збіжності многочленів Тейлора до самої наближуваної функції.

Розгляньмо приклади такої графічної візуалізації для функції $\cos x$. Порівняємо графіки самої функції $\cos x$ із графіками її розкладів Тейлора різних степенів.

Приклад: Порівняємо функцію $\cos x$ з її розкладом Маклорена 4-го степеня на інтервалі $[-5,5]$.

Побудуємо розклад

```
(%i15) appr:taylor(cos(x),x,0,5);
(%o15) 1 - x^2/2 + x^4/24 + ...
```

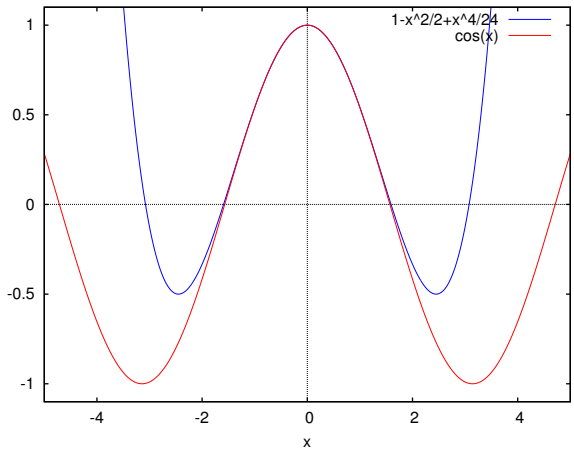


Рис. 3.9. Зіставлення розкладання в ряд Маклорена і функції

Побудуємо графік (екранна форма, у форматі **wxMaxima**)

```
( %i16) wxplot2d([appr,cos(x)], [x,-5,5], [y,-1.1,1.1], [nticks, 100]);
```

Виведемо графік до файла:

```
(%i17) plot2d([appr,cos(x)], [x,-5,5], [y,-1.1,1.1],
[gnuplot_preamble, "set grid;"], [gnuplot_term, ps],
[gnuplot_out_file, "appr.eps"])$
```

Легко помітити, що при невеликих значеннях x графіки самої функції і її наближувального розкладу, практично збігаються, однак при зростанні x починають відрізнятися.

Приклад: Порівняємо функцію $\cos x$ з її розкладом Маклорена 8-го степеня на інтервалі $[-5,5]$. Зіставимо результат з попереднім прикладом.

Побудуємо розклад вищого степеня:

```
(%i18) appr1:taylor(cos(x),x,0,9);
(%o18) 1 - x^2/2 + x^4/24 - x^6/720 + x^8/40320 + ...
```

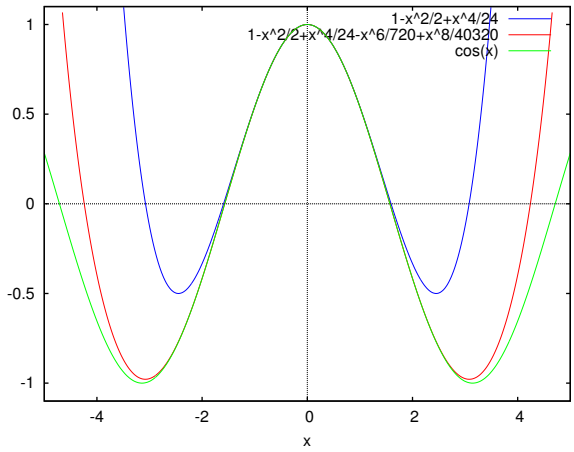


Рис. 3.10. Зіставлення двох розкладів у ряд Маклорена і функції $y = \cos x$

Приклад показує, що при використанні розкладу Тейлора вищого степеня точність наближення зростає і вдається досягти задовільного наближення на ширшому інтервалі. Однак помітимо, що степінь розкладу Тейлора не можна підвищувати необмежено у зв'язку з нагромадженням обчислювальної похибки.

Розклад у ряд Тейлора може використовуватися і для обчислення границь (функція `tlimit`, за синтаксисом аналогічна до `limit`).

3.6.2 Наближене обчислення визначених інтегралів

Степеневі ряди ефективні і зручні при наближеному обчисленні визначених інтегралів, що не виражаються у скінченному вигляді через елементарні функції. Для обчислення $\int_0^x f(t)dt$ підінтегральна функція $f(t)$ розкладається у степеневий ряд. Якщо

$$f(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n + \ldots, |x| < R,$$

то при $|x| < R$ степеневої ряд можна інтегрувати почленно. Отримуємо спосіб обчислення інтеграла $\int_0^x f(t)dt$ з будь-якою наперед заданою точністю

$$\int_0^x f(t)dt = a_0 x + a_1 \frac{x^2}{2} + a_2 \frac{x^3}{3} + \cdots + a_n \frac{x^{n+1}}{n+1} + \ldots$$

Приклад: Наближене обчислення інтеграла ймовірностей

$$\phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-x}^{x} e^{-t^2/2} dt = \frac{2}{\sqrt{2\pi}} \int_0^x e^{-t^2/2} dt.$$

Оскільки

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots, |x| < \infty,$$

то

$$e^{-x^2/2} = 1 - \frac{x^2}{2} + \frac{x^4}{2^2 2!} - \frac{x^6}{2^3 3!} + \cdots$$

Підставивши цей ряд під знак інтеграла і виконавши почленне інтегрування отримуємо

$$\phi(x) = \frac{2}{\sqrt{2\pi}} \int_0^x e^{-t^2/2} dt = \frac{2}{\sqrt{2\pi}} \left(x - \frac{x^3}{3 \cdot 2} + \frac{x^5}{5 \cdot 2^2 \cdot 2!} - \frac{x^7}{7 \cdot 2^3 \cdot 3!} + \ldots \right).$$

Оскільки це знакочерговий ряд з послідовно спадними за модулем доданками, похибка обчислення інтеграла послідовно спадає і не перевищує останнього доданка.

Розгляньмо приклад наближеного подання інтеграла у вигляді полінома деякого степеня у тому випадку, коли він не обчислюється у замкненій аналітичній формі.

Приклад: Обчислити $\int_0^1 e^{-x^2/2} dx$, оцінити досягнуту точність

Скористаємося розкладом підінтегральної функції у ряд. Підставляючи у отриманий раніше вираз $x = 1$, обчислюємо шуканий інтеграл. Оскільки досліджуваний ряд знакочерговий, похибка заміни нескінченної суми скінченним виразом за абсолютною величиною не перевищує першого відкинутого члена.

```
(%i1) f(x):=exp(-x^2/2);
```

(%o1) $f(x) := exp\left(\frac{-x^2}{2}\right)$

```
(%i2) taylor(f(x),x,0,8);
```

(%o2) $1 - \frac{x^2}{2} + \frac{x^4}{8} - \frac{x^6}{48} + \frac{x^8}{384} + \ldots$

Інтегруючи у межах від 0 до 1, одержуємо числовий результат:

```
(%i3) integrate(%,x,0,1);
(%o3) 103499/120960
(%i4) float (%);
(%o4) 0.85564649470899
```

Точність розрахунку оцінюємо, інтегруючи в межах від 0 до a:

```
(%i5) integrate(%o2,x,0,a);
(%o5)
```

$$\frac{120960\cdot\ a-20160\cdot\ a^3+3024\cdot\ a^5-360\cdot\ a^7+35\cdot\ a^9}{120960}$$

При $a=1$ знаходимо:

```
(%i6) expand(%);
(%o6) a^9/3456 - a^7/336 + a^5/40 - a^3/6 + a
(%i7) float(1/3456);
(%o7) 2.8935185185185184 * 10^-4
```

Таким чином, точність розрахунку значення інтеграла $\int_0^1 e^{-x^2/2}dx$ є не меншою за $0,0003$. Остаточно

$$\int_0^1 e^{-x^2/2}dx = 2.8935 \pm 0.0003.$$

3.7 Перетворення степеневих рядів

Пакет **Maxima** надає змогу не лише будувати розклади різних функцій у степеневі ряди, але і подавати їх у вигляді дробово-раціональної функції (апроксимації Паде) або ланцюгового дробу.

Апроксимацією Паде для функції $f(x)=\sum_{k=0}^{\infty}a_kx^k$, заданої степеневим рядом, називається така дробово-раціональна функція $R(x)=\frac{\sum_{k=0}^{L}p_kx^k}{1+\sum_{l=1}^{M}q_lx^l}$, чий розклад у степеневої ряд збігається зі степеневим рядом $f(x)$ з точністю до коефіцієнта при x^{L+M}.

Паде-апроксимант задається значенням функції в заданій точці і $M+L$ значеннями її похідних у цій же точці. Ця ж інформація може слугувати основою для розкладу функції у степеневий ряд, так у чому ж відмінність? Головна відмінність у тому, що задавши $M+L+1$ член степеневого ряду, ми відкидаємо інші члени ряду, дорівнюючи їх до нуля. Паде-апроксимант не є поліномом, тому задавши $M+L+1$ членів розкладу Паде-апроксиманта у степеневої ряд, ми в неявній формі задаємо і інші члени.

Чому ці додаткові члени будуть рівні? Це питання, на яке немає однозначної відповіді. В одних випадках вони дозволять нам побудувати точнішу апроксимацію, у інших – ні. Немає способу, що дозволив би сказати, наскільки точною виявиться Паде-апроксимація і у якому околі і з якою точністю можна одержати результати.

Ще одним недоліком цього методу є те, що він вимагає інформації не про значення функції, а про значення її похідних вищих порядків, які можуть бути значно більшими за абсолютною величиною, чим самі значення функції.

Паде-апроксимація найбільш ефективна для функцій, що мають полюси на комплексній площині на околицях точки розкладання. Наприклад, функція $f(x)=\dfrac{1}{1+\sin^2 x}$ неперервна на дійсній осі, але має полюси на комплексній площині. Тому вона неефективно апроксимується степеневим рядом (до шостого степеня включно), але добре апроксимується за Паде зі степенями чисельника і знаменника рівними 4 і 2.

Функція `pade`, представлена у пакеті **Maxima**, апроксимує частину ряду Тейлора, що містить доданки до n-го порядку включно, дробово-раціональною функцією. Її аргументи – ряд Тейлора, порядок чисельника n, порядок знаменника m. Зрозуміло, кількість відомих коефіцієнтів ряду Тейлора повинна збігатися із загальною кількістю коефіцієнтів у дробово-раціональній функції мінус один, оскільки чисельник і знаменник визначені з точністю до спільного множника.

Синтаксис виклику функції `pade`:

`pade(ряд Тейлора, степінь чисельника, степінь знаменника)`

Замість ряду Тейлора може використовуватися ряд Лорана. У цьому випадку степені чисельника і знаменника можуть бути і нескінченними (inf). У цьому випадку розглядаються всі раціональні функції, сумарний степінь яких менше або дорівнює довжині степеневого ряду. **Приклад:**

```
(%i1) t:taylor(exp(x),x,0,3);
(%o1) 1 + x + x^2/2 + x^3/6 + ...
(%i2) pade(t,1,2);
```

```
(%o2) [(2 x + 6)/(x^2 - 4 x + 6)]
(%i3) taylor(sin(x)/x,x,0,7);
(%o3) 1 - x^2/6 + x^4/120 - x^6/5040 + ...
(%i4) pade(%,2,4);
(%o4) [-(620 x^2 - 5880)/(11 x^4 + 360 x^2 + 5880)]
(%i5) taylor(1/(cos(x)-sec(x))^3,x,0,5);
(%o5) -1/x^6 + 1/(2 x^4) + 11/(120 x^2) - 347/15120 - 6767 x^2/604800 - 15377 x^4/7983360 + ...
(%i6) pade(%,3,inf);
(%o6) [-120/(41 x^10 + 60 x^8 + 120 x^6), (8806092 x^2 - 16847160)/(1353067 x^10 - 382512 x^8 + 16847160 x^6)]
```

Специфічнішою є функція `cf`, що обчислює коефіцієнти ланцюгового дробу, що апроксимує заданий вираз. Синтаксис виклику — `cf(expr)`. Вираз `expr` має складатися з цілих чисел, квадратних коренів із цілих чисел і знаків арифметичних дій. Функція повертає список коефіцієнтів (неперервний дріб $a + 1/(b + 1/(c + ...))$ представляється списком `[a,b,c,... ]`). Прапорець `cflength` визначає кількість періодів ланцюгового дробу. Від початку встановлене значення 1. Функція `cfdisrep` перетворює список (як правило виведений функцією `''cf`) у власне ланцюговий дріб вигляду $a + 1/(b + 1/(c + ...))$.

Приклади використання функцій `cf` і `cfdisrep`:

```
(%i1) cf([1,2,-3]+[1,-2,1]);
(%o1) [1,1,1,2]
(%i2) cfdisrep(%);
(%o2) 1 + 1/(1 + 1/(1 + 1/2))
(%i3) cflength:3;
(%o3) 3
(%i4) cf(sqrt(3));
(%o4) [1,1,2,1,2,1,2]
(%i5) cfdisrep(%);
(%o5) 1 + 1/(1 + 1/(2 + 1/(1 + 1/(2 + 1/(1 + 1/2)))))
```

3.8 Розв'язування диференціальних рівнянь у Maxima

3.8.1 Основні визначення

Диференціальним рівнянням називається рівняння вигляду $F(x, y, y', \ldots, y^{(n)}) = 0$, де $F(t_0, t_1, \ldots, t_{n+1})$ – функція, визначена у деякій області D простору $\mathbb{R}^{n+2}$, x – незалежна змінна, y – функція від x, y', $\ldots$, $y^{(n)}$ – її похідні.

Порядком рівняння n називається найвищий з порядків похідних y, що входять у рівняння. Функція $f(x)$ називається розв'язком диференціального рівняння на проміжку $(a; b)$, якщо для всіх x з $(a; b)$ виконується рівність: $F(x, f(x), f'(x), \ldots, f^{(n)}(x)) = 0$.

Диференціальному рівнянню задовольняє нескінченна множина функцій, але за деяких умов розв'язок такого рівняння єдиний. Інтегральна крива – це графік розв'язку диференціального рівняння, тобто графік функції, що задовольняє цьому рівнянню.

Якщо диференціальне рівняння містить одну незалежну змінну, то воно називається звичайним диференціальним рівнянням, якщо ж незалежних змінних дві або більше, то таке диференціальне рівняння називається диференціальним рівнянням у частинних похідних.

Приклад: Розв'язати рівняння $y' = 0$.

Очевидно, що його розв'язок $f(x) = const$ визначений на $(-\infty, \infty)$. Зазначимо, що ця стала – довільна і розв'язок – не єдиний: існує нескінченна множина розв'язків.

Приклад: Розв'язати рівняння $y' = \frac{y}{x}$ або $\frac{dy}{dx} = \frac{y}{x}$.

Перетворюючи рівняння, отримуємо: $\frac{dy}{y} = \frac{dx}{x}$. Інтегруючи обидві частини рівняння, отримуємо: $\int \frac{dy}{y} = \int \frac{dx}{x} \Rightarrow \ln y = \ln x + \ln C$ або $y = Cx$. Загальний розв'язок зображається серією лінійних інтегральних кривих, що проходять через точку ;(0,0);. При цьому через будь-яку точку, що не збігається з $(0, 0)$, проходить тільки одна інтегральна крива (розв'язок).

Загальним розв'язкомк – множиною розв'язків диференціального рівняння $y' = f(x,y)$ – є сукупність функцій $F(x,y,C) = 0$, $C = const$. Частинний розв'язок отримують підставлянням конкретного значення сталої у загальний розв'язок. Особливі розв'язки не входять у загальні розв'язки, і через кожну точку особливого розв'язання проходить більше однієї інтегральної кривої. Особливі розв'язки не можна одержати із загального розв'язку ні за яких значень сталої C. Якщо побудувати сімейство інтегральних кривих диференціального рівняння, то особливий розв'язок буде зображуватися лінією, що у кожній своїй точці дотикається до принаймні однієї інтегральної кривої.

Приклад: Розгляньмо рівняння $y' = -\frac{x}{y}$. Перетворюючи його, знайдемо: $\frac{dy}{dx} = -\frac{x}{y} \Rightarrow 2ydy + 2xdx = 0 \Leftrightarrow d(x^2+y^2) = 0$. Інтегруючи, отримуємо $x^2 + y^2 = C$.

Приклад: Диференціальне рівняння $y' = 2\sqrt{y}$ має загальний розв'язок $y = (x-C)^2$ і особливий розв'язок $y = 0$. При конкретному значенні C (наприклад, $C = 1$) отримуємо частинний розв'язок: $y = (x-1)^2$.

Геометрично множина розв'язків диференціального рівняння представляється у вигляді поля напрямків. У кожній точці області, у якій визначене поле напрямків, задається пряма з кутовим коефіцієнтом, рівним похідній розв'язку. Дотична до всіх подібних прямих і дає інтегральну криву.

Можливість однозначного розв'язання диференціального рівняння визначається теоремою єдиності:

Нехай $f(x,y)$ – неперервна функція в області $D = (x,y; a < x < b; c < y < d)$, причому частинна похідна $f(x,y)$ також неперервна у D. Тоді існує єдиний розв'язок $y = y(x)$ диференціального рівняння $y' = f(x,y)$ з початковою умовою $y(x_0) = y_0$, $(x_0, y_0) \in D$. Отже, через точку $(x_0, y_0) \in D$ проходить тільки одна інтегральна крива.

3.8.2 Функції для розв'язання диференціальних рівнянь у Maxima

В **Maxima** передбачені спеціальні засоби розв'язання задачі Коші для систем звичайних диференціальних рівнянь, заданих як у явній формі $\frac{dx}{dt} = F(t,x)$, так і в неявній $M\frac{dy}{dt} = F(t,x)$, де M – матриця, – т. зв. розв'язувач ЗДР (*ODE solver*), що забезпечує користувачеві можливість вибору методу, завдання початкових умов тощо.

Функція `ode2` надає змогу розв'язувати звичайні диференціальні рівняння першого і другого порядків.

Синтаксис виклику `ode2(eqn,dvar,ivar)`, де `eqn` – вираз, що визначає саме диференціальне рівняння, залежна змінна – `dvar`, незалежна змінна – `ivar`.

Диференціальне рівняння представляється у формі з «замороженою» похідною (тобто з похідною, обчислення якої заборонене за допомогою одинарних лапок: `'diff(y, x)`). Інший варіант явно вказати залежність $y = y(x)$ – використати функцію `depends` (у цьому випадку можна не використовувати початковий апостроф див. приклад). Якщо `ode2` не може одержати розв'язок, вона повертає значення `false`.

За допомогою функції `ode2` можуть бути розв'язані наступні типи ЗДР першого порядку: лінійні, ЗДР з відокремлюваними змінними, однорідні ЗДР, рівняння у повних диференціалах, рівняння Бернуллі, узагальнені однорідні рівняння.

Крім того, за допомогою функції `ode2` можуть бути розв'язані наступні типи рівнянь другого порядку: зі сталими коефіцієнтами; у повних диференціалах; лінійні однорідні зі змінними коефіцієнтами, які можуть бути зведені до рівнянь зі сталими коефіцієнтами; рівняння Ойлера; рівняння, розв'язні методом варіації сталих; рівняння, вільні від незалежної змінної, що допускають зниження порядку.

Тип використовуваного методу зберігається у змінній `method`. При використанні інтегрувального множника він зберігається у змінній `intfactor`. Частинний розв'язання неоднорідного рівняння зберігається у змінній `yp`.

Для відшукання частинних розв'язків задач Коші з початковими умовами використовуються функції `ic1` (для рівнянь першого порядку) і `ic2` (для рівнянь другого порядку). Частинні розв'язки граничних задач для рівнянь другого порядку використовують функцію `bc2`.

Розгляньмо приклади використання функції `ode2`.

Варіант використання відкладеного диференціювання (`'diff`):

```
(%i1) ode2('diff(y,x)=2*y+exp(x),y,x);
(%o1) y = (%c - e^-x)e^2x
```

Варіант із явним зазначенням залежності $y = y(x)$:

```
(%i1) depends(y,x);
(%o1) [y(x)]
(%i2) ode2(diff(y,x)=2*y+exp(x),y,x);
(%o2) y = (%c - e^-x)e^2x
```

Параметр `%c` – стала інтегрування для рівняння першого порядку.

Розв'язування рівняння другого порядку:

```
(%i4) ode2('diff(y,x,2)-3*'diff(y,x)+2*y=0,y,x);
(%o4) y = %k1%e^2x + %k2%e^x
```

Параметри `%k1` і `%k2` – сталі інтегрування для рівнянь другого порядку.

Розгляньмо варіанти обчислення частинних розв'язків: для рівняння першого порядку

```
(%i5) ic1(%o1,x=1,y=1);
(%o5) y = %e^-2 ((%e + 1)%e^2x - %e^x+2)
```

для рівняння другого порядку

```
(%i6) ic2(%o4,x=0,y=1,diff(y,x)=1);
```
(%o6) $y = \%e^x$

3.8.3 Розв'язування основних типів диференціальних рівнянь

3.8.3.1 Рівняння з відокремлюваними змінними

Рівняннями з відокремлюваними змінними називаються рівняння виду $y' = f(x) \cdot g(y)$, де $f(x)$ – функція, неперервна на деякому інтервалі (a, b), а функція $g(y)$ – функція, неперервна на інтервалі (c, d), причому $g(y) = 0$ на (c, d)).

Перетворюючи рівняння, одержуємо: $\frac{dy}{dx} = f(x) \cdot g(y) \Leftrightarrow \frac{dy}{g(y} = f(x)dx$.

Інтегруючи обидві частини, отримуємо $\int \frac{dy}{g(y)} = \int f(x)dx$. Позначаючи $G(y)$ будь-яку первісну для $\frac{1}{g(y)}$, а $F(x)$ – будь-яку первісну для $f(x)$, одержуємо загальний розв'язок диференціального рівняння у вигляді неявно вираженої функції $G(y) = F(x) + C$.

Приклад розв'язання у **Maxima**:

Відшукуємо загальний розв'язок:

```
(%i1) difr1:'diff(y,x)=sqrt(1-y^2)/sqrt(1-x^2);
```
(%o1) $\frac{d}{dx}y = \frac{\sqrt{1-y^2}}{\sqrt{1-x^2}}$
```
(%i2) rez:ode2(difr1,y,x);
```
(%o2) $asin(y) = asin(x) + \%c$

Відшукуємо різні варіанти приватних рішень:

```
(%i3) ic1(rez,x=0,y=0);
```
(%o3) $asin(y) = asin(x)$
```
(%i4) ic1(rez,x=0,y=1);
```
(%o4) $asin(y) = \frac{2asin(x) + \pi}{2}$

3.8.3.2 Однорідні рівняння

Під однорідними рівняннями розуміємо рівняння виду $y = f(\frac{y}{x})$. Для їхнього розв'язування використовується заміна виду $y = u \cdot x$, після підставляння якої виходить рівняння з відокремлюваними змінними: $y' = u'x + u \Rightarrow u'x + u = f(u)$. Розділяючи змінні і інтегруючи, отримуємо: $x\frac{du}{dx} = f(u) - u \Rightarrow \int \frac{du}{f(u)-u} = \int \frac{dx}{x}$.

Приклад розв'язання у **Maxima**:

Знаходимо загальний розв'язок:

```
(%i1) homode:'diff(y,x) = (y/x)^2 + 2*(y/x);
```
(%o1) $\frac{d}{dx}y = \frac{y^2}{x^2} + \frac{2y}{x}$
```
(%i2) ode2(homode,y,x);
```
(%o2) $-\frac{xy + x^2}{y} = \%c$

Знаходимо частинний розв'язок:

```
(%i3) ic1(%,x=2,y=1);
```
(%o3) $-\frac{xy + x^2}{y} = -6$

Загальнішим варіантом диференціальних рівнянь, рівняння типу: $y' = \frac{a_1x+b_1y+c_1}{a_2x+b_2y+c_2}$ – зводимо їх до однорідного. **Maxima** не здатна розв'язувати такі рівняння за допомогою `ode2` безпосередньо, а лише після необхідного перетворення.

3.8.3.3 Лінійні рівняння першого порядку

Диференціальне рівняння називається лінійним щодо невідомої функції і її похідної, якщо воно може бути записане у вигляді:

$$y' + P(x) \cdot y = Q(x)$$

при цьому, якщо права частина $Q(x)$ дорівнює нулю, то таке рівняння називається *лінійним однорідним диференціальним рівнянням*, якщо права частина $Q(x)$ не дорівнює нулю, то таке рівняння називається *лінійним неоднорідним диференціальним рівнянням*. При цьому $P(x)$ і $Q(x)$ – функції неперервні на деякому проміжку $x \in (a, b)$.

Розгляньмо розв'язування лінійного диференціального рівняння у **Maxima**:

```
(%i1) lineq1:'diff(y,x)-y/x=x;
```
(%o1) $\frac{d}{dx}y - \frac{y}{x} = x$
```
(%i2) ode2(lineq1,y,x);
```
(%o2) $y = x(x + \%c)$

При роботі з **Maxima** не потрібно приводити диференціальне рівняння до стандартної форми

$$y' + P(x)y = Q(x)$$

```
(%i3) lineq2:y^2-(2*x*y+3)*'diff(y,x)=0;
```
(%o3) $y^2 - (2xy + 3)\left(\frac{d}{dx}y\right) = 0$
```
(%i4) ode2(lineq2,y,x);
```
(%o4) $\frac{xy+1}{y^3} = \%c$

3.8.3.4 Рівняння у повних диференціалах

Диференціальне рівняння першого порядку вигляду:

$$P(x, y)dx + Q(x, y)dy = 0$$

називається рівнянням у повних диференціалах, якщо ліва частина цього рівняння являє собою повний диференціал деякої функції $u = F(x, y)$. Дане диференціальне рівняння є рівнянням у повних диференціалах, якщо виконується умова

$$\frac{\partial P}{\partial y} = \frac{\partial Q}{\partial x}.$$

Загальний інтеграл рівняння має вигляд $U(x, y) = 0$.

Якщо рівняння $P(x, y)dx + Q(x, y)dy = 0$ не є рівнянням у повних диференціалах, але виконуються умови теореми єдиності, то існує функція $\mu = \mu(x, y)$ (інтегрувальний множник) така, що

$$\mu(Pdx + Qdy) = dU.$$

Функція μ задовольняє умові:

$$\frac{\partial(\mu P)}{\partial y} = \frac{\partial(\mu Q)}{\partial x}.$$

Приклади розв'язання у **Maxima**:

Для розв'язання у **Maxima** диференціальне рівняння представляють у формі

$$P(x, y) + Q(x, y)\frac{dy}{dx} = 0.$$

Рівняння, що зводиться до рівняння у повних диференціалах

```
(%i1) deq:(2*x*y+x^2*y+y^3/3)+(x^2+y^2)*'diff(y,x)=0;
```
(%o1) $\left(y^2 + x^2\right)\left(\frac{d}{dx}y\right) + \frac{y^3}{3} + x^2y + 2xy = 0$
```
(%i2) ode2(deq,y,x);
```
(%o2) $\frac{\%e^x y^3 + 3x^2 \%e^x y}{3} = \%c$

Вказівка на інтегрувальний множник

```
(%i3) intfactor;
```
(%o3) $\%e^x$

Вказівка на використаний метод

```
(%i4) method;
```
(%o4) *exact*

Рівняння у повних диференціалах

```
(%i5) deq1:(3*x^2+6*x*y^2)+(6*x^2*y+4*y^3)*'diff(y,x)=0;
```
(%o5) $(4y^3 + 6x^2y)\left(\frac{d}{dx}y\right) + 6xy^2 + 3x^2 = 0$
```
(%i6) ode2(deq1,y,x);
```
(%o6) $y^4 + 3x^2y^2 + x^3 = \%c$

Вказівка на використаний метод

```
(%i7) method;
```
(%o7) *exact*

3.8.3.5 Рівняння Бернуллі

Рівнянням Бернуллі називається рівняння вигляду

$$y' + P(x) \cdot y = Q(x) \cdot y^{\alpha},$$

де P і Q – функції від x або сталі числа, а α – стале число, не рівне 0 і 1[3].

Для розв'язання рівняння Бернуллі застосовують підстановку $z = \frac{1}{y^{\alpha-1}}$, за допомогою якої, рівняння Бернуллі приводиться до лінійного.

Приклад розв'язання рівняння Бернуллі за допомогою **Maxima**:

```
(%i1) deq:'diff(y,x)=4/x*y+x*sqrt(y);
```
(%o1) $\frac{d}{dx}y = \frac{4y}{x} + x\sqrt{y}$
```
(%i2) ode2(deq,y,x);
```
(%o2) $y = x^4\left(\frac{\log(x)}{2} + \%c\right)^2$
```
(%i3) method;
```
(%o3) *bernoulli*
```
(%i4) de1:'diff(y,x)+y/x=-x*y^2;
```
(%o4) $\frac{d}{dx}y + \frac{y}{x} = -xy^2$
```
(%i5) ode2(de1,y,x);
```
(%o5) $y = \frac{1}{x(x+\%c)}$

3.8.3.6 Рівняння вищих порядків

У **Maxima** за допомогою функції `ode2` можливе пряме розв'язування лише лінійних диференціальних рівнянь другого порядку. При розв'язуванні виконується перевірка, чи є задане рівняння лінійним, тобто чи можливо його перетворення до форми $y'' + p(x)y + q(x)y = r(x)$.

Спочатку відшукується розв'язок однорідного рівняння вигляду $y'' + p(x)y + q(x) = 0$ у формі $y = k_1y_1 + k_2y_2$ (k_1, k_2 – довільні сталі). Якщо $r(x) \neq 0$, відшукується частинний розв'язок неоднорідного рівняння методом варіації сталих...

3.8.3.7 Рівняння зі сталими коефіцієнтами

Розв'язування однорідних рівнянь вигляду $y'' + ay' + by = 0$ відшукуються за результатами розв'язання характеристичного рівняння $r^2 + ar + b = 0$. Можливі наступні варіанти комбінацій його коренів r_1, r_2:

- r_1, r_2 – дійсні і різні. Розв'язок представляється у формі $y = k_1 \cdot e^{r_1 \cdot x} + k_2 \cdot e^{r_2 \cdot x}$.
- $r_1 = r_2$ – корені дійсні і однакові. Розв'язок представляється у формі $y = (k_1 + k_2 \cdot x)e^{r_1 \cdot x}$.
- r_1, r_2 – комплексні (спряжені). Якщо $r_1 = \alpha + \beta i$, $r_2 = \alpha - \beta i$, то розв'язок представляється у вигляді $y = e^{\alpha x}(k_1 \cos\beta x + k_2 \sin\beta x)$.

Загальний розв'язок неоднорідного рівняння зі сталими коефіцієнтами представляється у вигляді суми загального розв'язку відповідного однорідного рівняння і якогось частинного розв'язку неоднорідного.

Приклади розв'язання ЗДР другого порядку зі сталими коефіцієнтами:

Неоднорідне рівняння загального вигляду:

```
(%i1) de1:2*'diff(y,x,2)-'diff(y,x)-y=4*x*exp(2*x);
```
(%o1) $2\left(\frac{d^2}{dx^2}y\right) - \frac{d}{dx}y - y = 4x\ \ \%e^{2x}$
```
(%i2) ode2(de1,y,x);
```
(%o2) $y = \frac{(20x-28)\%e^{2x}}{25} + \%k1\%e^{x} + \ \ \%k2\%e^{-\frac{x}{2}}$

Частинний розв'язок неоднорідного рівняння зберігається у змінній *yp*:

```
(%i3) yp;
```
(%o3) $\frac{(20x-28)\ \ \%e^{2x}}{25}$

Неоднорідне рівняння із кратними коренями характеристичного рівняння:

[3]При $\alpha = 0$ одержуємо неоднорідне, а при $\alpha = 1$ – однорідне лінійне рівняння.

```
(%i1) de2:'diff(y,x,2)-2*'diff(y,x)+y=x*exp(x);
```
$$(\%o1)\quad \frac{d^2}{dx^2}y-2\left(\frac{d}{dx}y\right)+y=x\ \ \%e^x$$
```
(%i2) ode2(de2,y,x);
```
$$(\%o2)\quad y=\frac{x^3\%e^x}{6}+(\%k2x+\ \ \%k1)\ \ \%e^x$$
```
(%i3) yp;
```
$$(\%o3)\quad \frac{x^3\%e^x}{6}$$

Неоднорідне рівняння з комплексними коренями:

```
(%i4) de3:'diff(y,x,2)+y=x*sin(x);
```
$$(\%o4)\quad \frac{d^2}{dx^2}y+y=x\sin(x)$$
```
(%i5) ode2(de3,y,x);
```
$$(\%o5)\quad y=\frac{2x\sin(x)+\left(1-2x^2\right)\cos(x)}{8}+\%k1\sin(x)+\%k2\cos(x)$$
```
(%i6) yp;
```
$$(\%o6)\quad \frac{2x\sin(x)+\left(1-2x^2\right)\cos(x)}{8}$$

3.8.3.8 Рівняння зі змінними коефіцієнтами

Аналогічно рівнянню зі сталими коефіцієнтами, загальний розв'язок однорідного рівняння $y''+p(x)y'+q(x)y=0$ має вигляд $y=C_1y_1+C_2y_2$, де y_1, y_2 – лінійно незалежні розв'язки однорідного ЗДР (фундаментальна система розв'язків).

Загальний розв'язок неоднорідного рівняння $y''+p(x)y'+q(x)y=f(x)$ з неперервними коефіцієнтами і правою частиною має вигляд $y=y_0+Y$, де y_0 – загальний розв'язок відповідного однорідного рівняння, Y – частинний розв'язок неоднорідного.

Якщо відома фундаментальна система розв'язків однорідного рівняння, загальний розв'язок неоднорідного може бути представлений у формі:

$$y=C_1(x)\cdot y_1+C_2(x)\cdot y_2,$$

де $C_1(x)$, $C_2(x)$ визначаються методом варіації довільних сталих.

Приклад:

```
(%i3) difr:x^2*'diff(y,x,2)-x*'diff(y,x)=3*x^3;
```
$$(\%o3)\quad x^2\left(\frac{d^2}{dx^2}y\right)-x\left(\frac{d}{dx}y\right)=3x^2$$
```
(%i4) ode2(difr,y,x);
```
$$(\%o4)\quad y=x^3+\%k2x^2-\frac{\%k1}{2}$$

Приклад:

```
(%i3) difr1:x*'diff(y,x,2)+'diff(y,x)=x^2;
```
$$(\%o3)\quad x\left(\frac{d^2}{dx^2}y\right)+\frac{d}{dx}y=x^2$$
```
(%i4) ode2(difr1,y,x);
```
$$(\%o4)\quad y=\%k1\cdot\log(x)\ \ +\frac{x^3}{9}+\%k2$$

3.8.3.9 Рівняння Ойлера

Однорідне рівняння $x^2y''+axy'+by=0$ називається рівнянням Ойлера. Його загальний розв'язок має вигляд $y=k_1x^{r_1}+k_2x^{r_2}$, де r_1 і r_2– розв'язки рівняння $r(r-1)+ar+b=0$.

У випадку, коли рівняння $r(r-1)+ar+b=0$ має подвійний корінь r, розв'язок представляється у формі $y=k_1x^r+k_2\ln x x^r$.

Неоднорідне рівняння типу Ойлера зводиться до однорідного зі сталими коефіцієнтами шляхом відповідної заміни.

Приклад:

```
(%i1) dr:x^2*'diff(y,x,2)+x*'diff(y,x)+y=1;
```
$$(\%o1)\quad x^2\left(\frac{d^2}{dx^2}y\right)+x\left(\frac{d}{dx}y\right)+y=1$$
```
(%i2) ode2(du,y,x);
```
$$(\%o2)\quad y=\sin(\log(x))^2+\%k1\sin(\log(x))+\cos(\log(x))^2+\%k2\cos(\log(x))$$

3.8.3.10 Граничні задачі

Для задання граничних умов при інтегруванні ЗДР другого порядку використовується функція bc2.

Синтаксис виклику: `bc2 (solution, xval1, yval1, xval2, yval2)`, де `xval1` – значення x у першій граничній точці, `yval1` – значення розв'язку y у тій же точці (обидві величини задаються у формі $x = a$, $y = b$). Приклад використання `ode2` і `bc2`:

```
(%i1) 'diff(y,x,2) + y*'diff(y,x)^3 = 0;
```
(%o1) $\frac{d^2}{dx^2}y + y\left(\frac{d}{dx}y\right)^3 = 0$
```
(%i2) ode2(%,y,x);
```
(%o2) $\frac{y^3 + 6\%k1y}{6} = x + \%k2$
```
(%i3) bc2(%,x=0,y=1,x=1,y=3);
```
(%o3) $\frac{y^3 - 10y}{6} = x - \frac{3}{2}$

3.8.4 Операторний метод розв'язування

Для розв'язування систем звичайних лінійних диференціальних рівнянь у **Maxima** є функція `desolve`. Робота функції `desolve` заснована на перетворенні Лапласа заданих диференціальних рівнянь.

Нехай задана функція дійсного змінного $f(t)$, що задовольняє наступним умовам:

1. однозначна і неперервна разом зі своїми похідними n-го порядку для всіх $t > 0$, крім тих, де вона і її похідні мають розриви 1-го роду. При цьому у кожному кінцевому інтервалі зміни є кінцеве число точок розриву;
2. $f(t) = 0$ для всіх $t > 0$;
3. зростає повільніше деякої експоненційної функції $M \cdot e^{at}$, де M і a – деякі додатні величини, тобто завжди можна вказати такі M і a, щоб при будь-якому $t > 0$ виконувалася нерівність $|f(t)| < Me^{at}$.

Розглянутій функції $f(t)$ ставиться у відповідність нова функція, що визначається рівністю

$$F(s) = \mathcal{L}\{f(t)\} = \int_0^\infty e^{-st} f(t)dt,$$

де s – додатне дійсне число або комплексне число з додатною дійсною частиною.

Функція $f(t)$ при цьому називається оригіналом, а $F(s)$ – зображенням функції $f(t)$ за Лапласом. Перехід від оригіналу до зображення називається перетворенням Лапласа. Відповідно, зворотний перехід від зображення до оригіналу називається зворотним перетворенням Лапласа.

Для перетворення Лапласа виконується теорема єдиності: якщо дві неперервні функції $f(x)$ і $g(x)$ мають те саме зображення за Лапласом $F(p)$, то вони тотожно рівні.

За допомогою операційного числення можна порівняно просто розв'язувати різні задачі, що зводяться до інтегрування лінійних диференціальних рівнянь. Перехід від вихідних функцій до їхніх зображень надає змогу замінити розв'язування системи диференціальних рівнянь розв'язуванням системи алгебраїчних рівнянь (але при цьому знаходження зворотного перетворення Лапласа може бути досить складною задачею).

При обчисленні перетворення Лапласа похідні заміняються алгебраїчними виразами наступного вигляду:

$$pF(p) - f(0) = \mathcal{L}\{f'(t)\};$$

$$p^2F(p) - pf(0) - f'(0) = \mathcal{L}\{f''(t)\}$$

тощо, тому використання перетворення Лапласа для розв'язування систем ЗДР вимагає задання початкових умов. Використання `desolve` обмежується однією із властивостей перетворення Лапласа: якщо $\mathcal{L}\{(t)\} = F(s)$, то $\mathcal{L}\{tf(t)\} = -F'(s)$. Тому `desolve` припускає, що розв'язується система ЗДР зі сталими коефіцієнтами.

Синтаксис виклику `desolve`: `desolve(delist, fnlist)`, де `delist` – список розв'язуваних диференціальних рівнянь, `fnlist` – список шуканих функцій. При використанні `desolve` необхідно явно задавати функціональні залежності (замість `'diff(y, x)` використовувати запис `diff(y(x),x)`).

Приклади використання `desolve`:

Система ЗДР першого порядку:

```
(%i1) de1:diff(f(x),x)=diff(g(x),x)+sin(x);
```
(%o1) $\frac{d}{dx}f(x) = \frac{d}{dx}g(x) + \sin(x)$
```
(%i2) de2:diff(g(x),x,2)=diff(f(x),x)-cos(x);
```
(%o2) $\frac{d^2}{dx^2}g(x) = \frac{d}{dx}f(x) - \cos(x)$
```
(%i3) desolve([de1,de2],[f(x),g(x)]);
```
(%o3) $[f(x) = \ \%e^x\left(\left.\frac{d}{dx}g(x)\right|_{x=0}\right) - \left.\frac{d}{dx}g(x)\right|_{x=0} + f(0),\ \ g(x) = \ \%e^x\left(\left.\frac{d}{dx}g(x)\right|_{x=0}\right) - \left.\frac{d}{dx}g(x)\right|_{x=0} + \cos x + g(0) - 1]$

Одиничне диференціальне рівняння другого порядку:

```
(%i1) de3:diff(f(x),x,2)+f(x)=2*x;
```
$$\text{(\%o1)}\quad \frac{d^2}{dx^2}f(x)+f(x)=2x$$
```
(%i2) desolve(de3,f(x));
```
$$\text{(\%o2)}\quad f(x)=\sin(x)\left(\left.\frac{d}{dx}f(x)\right|_{x=0}-2\right)+f(0)\cos(x)+2x$$

Для задання початкових умов використовується функція `atvalue`. Синтаксис виклику:

`atvalue(expr, [`$x_1=a_1$`, ..., `$x_m=a_m$`],c)`

`atvalue(expr, `$x_1=a_1$`, c)`

Функція `atvalue` надає значення c виразу `expr` у точці $x=a$. Вираз `expr` – функція $f(x_1,...,x_m)$ або похідній `diff(`$f(x_1,...,x_m)$`, `x_1`, `n_1`, ..., `x_m`, `n_m`)`. Тут n_i -порядок диференціювання за змінною x_i.

Приклад використання `desolve` і `atvalue`:

```
(%i1) de1:diff(f(x),x)=diff(g(x),x)+sin(x);
```
$$\text{(\%o1)}\quad \frac{d}{dx}f(x)=\frac{d}{dx}g(x)+\sin(x)$$
```
(%i2) de2:diff(g(x),x,2)=diff(f(x),x)-cos(x);
```
$$\text{(\%o2)}\quad \frac{d^2}{dx^2}g(x)=\frac{d}{dx}f(x)-\cos(x)$$
```
(%i3) atvalue(f(x),x=0,1);
(%o3) 1
(%i4) atvalue(g(x),x=0,2);
(%o4) 2
(%i5) atvalue(diff(g(x),x),x=0,3);
(%o5) 3
(%i6) desolve([de1,de2],[f(x),g(x)]);
```
(%o6) $[f(x)=3e^x-2,\ \ g(x)=\cos(x)+3e^x-2]$

Керування початковими умовами здійснюється за допомогою функцій `properties` і `prontprops`. Функція `properties` (синтаксис виклику – `properties(a)`) друкує властивості змінної (атома a), а функція `printprops` друкує інформацію про задану властивість змінної. Крім того, функція `at` обчислює значення виразу у заданій точці з урахуванням властивості `atvalue`.

Синтаксис виклику `printprops`:

`printprops(a, i)`

`printprops ([`a_1`, ..., `a_n`], i)`

`printprops(all, i)`

Ця функція надає змогу переглянути властивості атома a (або групи атомів **Lisp**, зазначених у списку), визначені індикатором i.

Скасування встановленого `atvalue`, виконується функцією `remove` (вилучення властивості `p` в атомів a_1, ..., a_n здійснюється викликом `remove(`a_1`, `p_1`, ..., `a_n`, `p_n`)`; видалення списку властивостей – викликом `remove([`a_1`, ..., `a_m`], [`p_1`, ..., `p_n`], ...)`).

Приклад синтаксису і використання розглянутих функцій:

```
(%i1) eq1:'diff(f(x),x)='diff(g(x),x)+sin(x);
```
$$\text{(\%o1)}\quad \frac{d}{dx}f(x)=\frac{d}{dx}g(x)+\sin(x)$$
```
(%i2) eq2:'diff(g(x),x,2)='diff(f(x),x)-cos(x);
```
$$\text{(\%o2)}\quad \frac{d^2}{dx^2}g(x)=\frac{d}{dx}f(x)-\cos(x)$$
```
(%i3) atvalue('diff(g(x),x),x=0,a);
(%o3) a
(%i4) atvalue(f(x),x=0,1);
(%o4) 1
(%i5) properties(f);
(%o5) [atvalue]
(%i6) printprops(f,atvalue);
```
$f(0)=1$
```
(%o6) done
(%i7) desolve([eq1,eq2], [f (x), g(x)]);
```
(%o7) $[f(x)=a\%e^x-a+1,\ \ g(x)=\cos(x)+a\%e^x-a+g(0)-1]$
```
(%i8) at(%, [x=1]);
```
(%o8) $[f(1)=\ \ \%ea-a+1,\ \ g(1)=ea-a+\cos(1)+g(0)-1]$

Ще один приклад аналізу властивостей:

```
(%i9) atvalue(f(x,y), [x=0, y=1], a^2);
(%o9) a^2
(%i10) atvalue('diff(f(x,y), x), x=0, 1+y);
(%o10) @2+1
(%i11) printprops (all, atvalue);
```
$\frac{d}{d@1}\ g(@1)|_{[@1=0]} = a$
$\frac{d}{d@1}\ f(@1, @2)|_{[@1=0]} = @2 + 1$
$f(0, 1) = a^2$
$f(0) = 1$
```
(%o11) done
```

3.8.5 Додаткові можливості розв'язання ЗДР

3.8.5.1 Пакет contrib_ode

Як видно з наведеного вище опису можливостей **Maxima**, можливості основної функції для аналітичного розв'язування ЗДР – функції `ode2` – досить обмежені. Для розширення можливостей розв'язання ЗДР першого і другого порядку в останніх версіях **Maxima** існує пакет розширення `contrib_ode`. За допомогою `contrib_ode` можливе розв'язання рівнянь Клеро, Лагранжа, Ріккаті тощо. У загальному випадку результат – список розв'язків. Для деяких рівнянь (зокрема Ріккаті) розв'язок представляється у формі іншого ЗДР – результату заміни змінних. Функція `contrib_ode` реалізує методи факторизації (`factorization`), Клеро (`Clairault`), Лагранжа (`Lagrange`), Ріккаті (`Riccati`), Абеля (`Abel`) і метод симетрії Лі (`Lie symmetry method`).

Для використання пакет `contrib_ode` слід завантажити:

```
(%i1) load("contrib_ode")$
```

Приклад розв'язання ЗДР з використанням функції `contrib_ode`:

```
(%i2) eqn:x*'diff(y,x)^2-(1+x*y)*'diff(y,x)+y=0;
```
(%o2) $x\left(\frac{d}{dx}y\right)^2 - (xy+1)\left(\frac{d}{dx}y\right) + y = 0$
```
(%i3) contrib_ode(eqn,y,x);
```
(%t3) $x\left(\frac{d}{dx}y\right)^2 - (xy+1)\left(\frac{d}{dx}y\right) + y = 0$
```
first order equation not linear in y'
```
(%o3) $[y = \log(x) + \%c,\ \ y = \%c\%e^x]$
```
(%i4) method;
(%o4) factor
```

Перевага `contrib_ode` – можливість розв'язання нелінійних ЗДР першого порядку, оскільки вони можуть мати у загальному випадку кілька розв'язків, результат представляється у вигляді списку.

Синтаксис виклику `contrib_ode` не відрізняється від синтаксису виклику `ode2`.

Розгляньмо приклади розв'язання інших типів рівнянь.

3.8.5.2 Рівняння Клеро і Лагранжа

Рівняння Клеро

```
(%i1) load("contrib_ode")$
(%i2) eqn:'diff(y,x)^2+x*'diff(y,x)-y=0;
```
(%o2) $\left(\frac{d}{dx}y\right)^2 + x\left(\frac{d}{dx}y\right) - y = 0$
```
(%i3) contrib_ode(eqn,y,x);
```
(%t3) $\left(\frac{d}{dx}y\right)^2 + x\left(\frac{d}{dx}y\right) - y = 0$
```
first order equation not linear in y'
```
(%o3) $[y = \%cx + \%c^2,\ \ y = -\frac{x^2}{4}]$
```
(%i4) method;
(%o4) clairault
```

Рівняння Лагранжа

```
(%i5) leq:y=(1+'diff(y,x))*x+('diff(y,x))^2;
```
(%o5) $y = \left(\frac{d}{dx}y\right)^2 + x\left(\frac{d}{dx}y + 1\right)$
```
(%i6) contrib_ode(leq,y,x);
```

(%t6) $y = \left(\frac{d}{dx}y\right)^2 + x\left(\frac{d}{dx}y + 1\right)$
first order equation not linear in y'
(%o6) $[[x = \%e^{-\%t}\left(\%c - 2(\%t-1)\%e^{\%t}\right),\ y = (\%t+1)x + \%t^2]]$
(%i7) method;
(%o7) *lagrange*

У деяких випадках розв'язання можливе тільки у параметричній формі. Приклад (%t – параметр):

(%i8) eqn:'diff(y,x)=(x+y)^2;
(%o8) $\frac{d}{dx}y = (y+x)^2$
(%i9) contrib_ode(eqn,y,x);
(%o9) $[[x = \%c - atan\left(\sqrt{\%t}\right),\ y = -x - \sqrt{\%t}],\ [x = atan\left(\sqrt{\%t}\right) + \%c,\ y = \sqrt{\%t} - x]]$
(%i10) method;
(%o10) *lagrange*

3.8.5.3 Інші задачі з використанням contrib_ode

Пакет contrib_ode надає змогу розв'язувати диференціальні рівняння, які є нерозв'язними за допомогою ode2 безпосередньо. Приклад – узагальнені однорідні рівняння (див. вище). Представлені розв'язання використовують методи Абеля і симетрії Лі.

(%i11) eqn:(2*x-y+4)*'diff(y,x)+(x-2*y+5)=0;
(%o11)

$$(-y+2x+4)\left(\frac{d}{dx}y\right) - 2y + x + 5 = 0$$

(%i12) contrib_ode(eqn,y,x);
(%o12) $[\frac{\log\left(3 - \frac{2(2x+4)-x-5}{-y+2x+4}\right) - 3\log\left(1 - \frac{2(2x+4)-x-5}{-y+2x+4}\right) + 2\log\left(-\frac{2(2x+4)-x-5}{-y+2x+4}\right)}{2} = \log(x+1) + \%c]$
(%i13) method;
(%o13) *abel2*
(%i14) eqn1:'diff(y,x)=(1-3*x-3*y)/(1+x+y);
(%o14) $\frac{d}{dx}y = \frac{-3y-3x+1}{y+x+1}$
(%i15) contrib_ode(eqn1,y,x);
(%o15) $[\frac{2\log(y+x-1) + y + 3x}{2} = \%c]$
(%i16) method;
(%o16) *lie*

3.8.5.4 Розв'язування однорідних лінійних рівнянь

Інші корисні функції пакета contrib_ode: odelin і ode_check. Функція odelin розв'язує однорідні лінійні рівняння першого і другого порядку, і повертає фундаментальний розв'язок ЗДР.

Приклад:

```
(%i4) odelin(x*(x+1)*'diff(y,x,2)+(x+5)*'diff(y,x,1)+(-4)*y,y,x);
...trying factor method...
solving 7 equations in 4 variables...
trying the Bessel solver...solving 1 equations in 2 variables...
trying the F01 solver...
solving 1 equations in 3 variables...
trying the spherodial wave solver...
solving 1 equations in 4 variables...
trying the square root Bessel solver...
solving 1 equations in 2 variables...
trying the 2F1 solver...
solving 9 equations in 5 variables
```

(%o4) $\frac{gauss\_a(-6,-2,-3,-x)}{x^4},\ \frac{gauss\_b(-6,-2,-3,-x)}{x^4}$

Примітка: функції gauss_a і gauss_b – спеціальні функції, що являють розв'язки гіпергеометричного рівняння.
Функція ode_check надає змогу підставити в ЗДР знайдений розв'язок.

Приклад:

```
(%i1) load("contrib_ode")$
(%i2) eqn:(1+x^2)*'diff(y,x,2)-2*x*'diff(y,x);
```
(%o2) $\left(x^2+1\right)\left(\frac{d^2}{dx^2}y\right)-2x\left(\frac{d}{dx}y\right)$
```
(%i3) odelin(eqn,y,x);
...trying factor method...solving 7 equations in 4 variables
```
(%o3) $1, x\left(x^2+3\right)$
```
(%i4) ode_check(eqn,y=x*(x^2+3));
```
(%o4) 0
```
(%i5) ode_check(eqn,y=1);
```
(%o5) 0

3.8.6 Обчислювальні методи розв'язання ЗДР

Однак у ряді випадків відшукати символьний розв'язок ЗДР у досить компактному вигляді неможливо. У цьому випадку доцільно використати обчислювальні методи. **Maxima** включає пакет розширення `dynamics`, що надає змогу проінтегрувати систему ЗДР методом Рунґе-Кутти.

Починаючи з версії 5.12, **Maxima** включає пакет `dynamics` (його необхідно завантажувати перед використанням). Крім методу Рунґе-Кутти, пакет `dynamics` включає ряд функцій для побудови різних фракталів.

Метод Рунґе-Кутти реалізує функція `rk`. Синтаксис її виклику: `rk([eq], [vars], [init], [`t_{range}`])`, де `eq` – список правих частин рівнянь; `vars` – список залежних змінних; `init` – список початкових значень; t_{range} – список `[`t`,` t_0`,` t_{end}`,` ht`]`, що містить символьне позначення незалежної змінної (t), її початкове значення (t_0), кінцеве значення (t_{end}), крок інтегрування (ht).

Приклад: Розв'язати ЗДР

$$\frac{dx}{dt}=4x^2-4y^2; \qquad \frac{dy}{dt}=y^2-x^2+1;$$

при $t=[0\ldots4]$, $x(0)=-1,25$, $y(0)=0,75$.

Використаємо пакет `dynamics`.

```
(%i1) load("dynamics")$
```

Вибираємо крок інтегрування $0,02$.

```
(%i2) sol:rk([4*x^2-4*y^2,y^2-x^2+1], [x,y], [-1.25,0.75], [t,0,4,0.02]);
```

У результаті розв'язання одержуємо список значень у форматі $[[t,x,y]]$.

```
(%i1) load("dynamics")$
(%i2) rp1:4*x^2-4*y^2;
```
(%o2) $4x^2-4y^2$
```
(%i3) rp2:y^2-x^2+1;
```
(%o3) y^2-x^2+1
```
(%i4) sol:rk([rp1,rp2], [x,y], [-1.25,0.75], [t,0,4,0.02])$
```

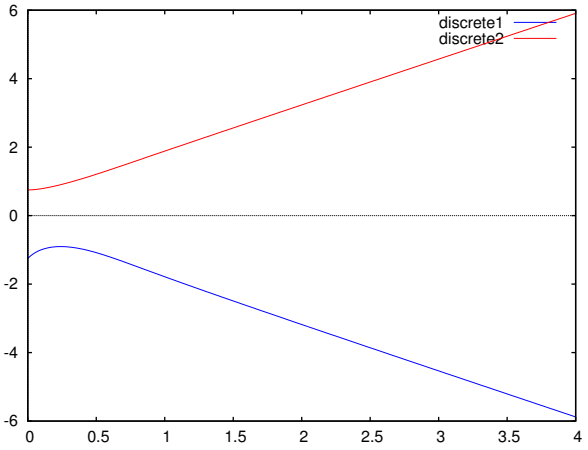


Рис. 3.11. Приклад графічного розв'язання системи ЗДР обчислювальним методом

Список `sol` не виводимо на екран (він досить довгий, тому завершуємо введення команди символом $).

Для побудови графіка розв'язку перетворимо отриманий список, побудувавши окремо список значень t (список xg у прикладі), x (список $yg1$), y (список $yg2$). При побудові графіка використаємо параметр `discrete`.

```
(%i5) len:length(sol);
(%o5) 201
(%i6) xg:makelist(sol[k][1],k,1,len)$
(%i7) yg1:makelist(sol[k][2],k,1,len)$
(%i8) yg2:makelist(sol[k][3],k,1,len)$
(%i9) plot2d([[discrete,xg,yg1],[discrete,xg,yg2]]);
```

Результат розв'язання представлений на рис. 3.11.
Аналогічний, хоча і трохи складніший приклад – моделювання атрактора Лоренца (див. стор. 143).

3.9 Ряди Фур'є за ортогональними системами

Пакет **Maxima** включає досить широкі можливості для роботи як із класичними тригонометричними рядами Фур'є, так і з рядами Фур'є за іншими ортогональними системами. Розгляньмо короткий вступ, необхідний для розуміння наведених прикладів.

3.9.1 Поняття ряду Фур'є

Нехай дані дві функції $f(x)$ і $g(x)$, добуток яких інтегрувальне на відрізку $[a,b]$. Функції $f(x)$ і $g(x)$, називають ортогональними на $[a,b]$, якщо виконується умова

$$\int_a^b f(x)g(x)\rho(x)dx = 0$$

де $\rho(x)$ – вагова функція.
Функціональна послідовність $\{\varphi_n(x)\} = \{\varphi_0(x), \varphi_1(x), \dots, \varphi_n(x), \dots\}$ називається ортогональною на $[a,b]$, якщо виконується умова:

$$\int_a^b \varphi_n(x)\varphi_m(x)\rho(x)dx = 0, \forall n \neq m.$$

Функціональна послідовність $\{\varphi_n(x)\}$ називається ортонормованою на $[a,b]$, якщо

$$\int_a^b \varphi_n(x)\varphi_m(x)\rho(x)dx = \left\{ \begin{array}{lll} 1, & \text{якщо} & n = m; \\ 0, & \text{якщо} & n \neq m; \end{array} \right.$$

Часто використовувана послідовність із тригонометричних функцій $1, \cos x, \sin x, \cos 2x, \sin 2x, \dots, \cos nx, \sin nx, \dots$ ортогональна на відрізку $[-\pi,\pi]$ з ваговою функцією $\rho(x) = 1$.
Перевіримо властивість ортогональности, обчислюючи відповідні інтеграли. При m = n отримуємо:

$$\int_{-\pi}^{\pi} 1 \cdot \sin nxdx = -\left.\frac{\cos nx}{n}\right|_{-\pi}^{\pi} = 0,$$

$$\int_{-\pi}^{\pi} 1 \cdot \cos nxdx = \left.\frac{\sin nx}{n}\right|_{-\pi}^{\pi} = 0,$$

$$\int_{-\pi}^{\pi} \sin mx \cdot \sin nxdx = \frac{1}{2}\int_{-\pi}^{\pi} (\cos(m-n)x - \cos(m+n)x)\,dx = \frac{1}{2}\left(\frac{\sin(m-n)x}{m-n} - \frac{\sin(m+n)x}{m+n}\right)\Bigg|_{-\pi}^{\pi} = 0,$$

$$\int_{-\pi}^{\pi} \cos mx \cdot \cos nxdx = \frac{1}{2}\int_{-\pi}^{\pi} (\cos(m-n)x + \cos(m+n)x)\,dx = \frac{1}{2}\left(\frac{\sin(m-n)x}{m-n} + \frac{\sin(m+n)x}{m+n}\right)\Bigg|_{-\pi}^{\pi} = 0,$$

Якщо ж $m = n$, то

$$\int_{-\pi}^{\pi} \cos^2 mxdx = \frac{1}{2}\int_{-\pi}^{\pi} (1 + \cos 2mx)\,dx = \frac{1}{2}\left(x + \frac{\sin 2mx}{2m}\right)\Bigg|_{-\pi}^{\pi} = \pi,$$

Отже, $\int_{-\pi}^{\pi} \cos mx \cos nxdx = \left\{ \begin{array}{l} 0, m \neq n, \\ \pi, m = n. \end{array} \right.$ Аналогічним чином, $\int_{-\pi}^{\pi} \sin mx \sin nxdx = \left\{ \begin{array}{l} 0, m \neq n, \\ \pi, m = n. \end{array} \right.$
Залишається обчислити інтеграл $\int_{-\pi}^{\pi} \cos mx \sin nxdx$.
Оскільки підінтегральна функція є непарною, то

$$\int_{-\pi}^{\pi} \cos mx \sin nxdx = 0.$$

Як випливає з наведених рівностей, будь-які дві різні функції тригонометричної послідовності ортогональні на відрізку $[-\pi,\pi]$.

Іншою широко використовуваною послідовністю ортогональних функцій є послідовність поліномів Лежандра. Поліном Лежандра степеня n можна представити через формулу Родріга у вигляді:

$$P_n(z) = \frac{1}{2^n n!}\frac{d^n}{dz^n}\left(z^2-1\right)^n.$$

Ці поліноми також можна обчислити за допомогою рекурентної формули:

$$P_{n+1}(x) = \frac{2n+1}{n+1}xP_n(x) - \frac{n}{n+1}P_{n-1}(x)$$

Поліноми Лежандра ортогональні на відрізку $[-1,1]$ з вагою $\rho(x)=1$:

$$\int_{-1}^{1} P_k(x)P_l(x)dx = \begin{cases} \frac{2}{2k+1}, & \text{якщо} \quad k=l; \\ 0, & \text{якщо} \quad k\neq l. \end{cases}$$

Ще однією важливою послідовністю ортогональних функцій є послідовність поліномів Чебишова. Поліноми Чебишова першого роду $T_n(x)$ степеня n можна визначити за допомогою рівності:

$$T_n(\cos\theta) = \cos n\theta,$$

або, що майже еквівалентно,

$$T_n(z) = \cos(n\arccos z).$$

Вони також можуть бути обчислені за рекурентною формулою:

$$T_0(x)=1; T_1(x)=x; T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x).$$

Поліноми Чебишова ортогональні на відрізку $[-1,1]$ з вагою $\rho(x) = \dfrac{1}{\sqrt{1-x^2}}$

$$\int_{-1}^{1} T_k(x)T_l(x)\frac{1}{\sqrt{1-x^2}}dx = \begin{cases} \frac{\pi}{2}, & \text{якщо} \quad k=l\neq 0; \\ \pi, & \text{якщо} \quad k=l=0; \\ 0, & \text{якщо} \quad k\neq l. \end{cases}$$

3.9.2 Обчислення коефіцієнтів тригонометричних рядів Фур'є

Члени тригонометричного ряду $\dfrac{a_0}{2} + \sum\limits_{n=1}^{\infty}(a_n\cos nx + b_n\sin nx)$ є періодичними функціями із загальним періодом 2π, тому і сума цього ряду $S(x)$ також буде періодичною функцією з періодом 2π.

Припустімо, що 2π-періодичну функцію $f(x)$ можна розкласти у тригонометричний ряд, що рівномірно збігається на відрізку $[-\pi,\pi]$.

$$f(x) = \frac{a_0}{2} + \sum_{n=1}^{\infty}(a_n\cos nx + b_n\sin nx)$$

Розгляньмо питання про визначення коефіцієнтів a_0, a_n і b_n $(n=1,2,\dots)$. Для цього застосуємо теорему про почленне інтегрування функціонального ряду. Проінтегруємо обидві частини рівності в межах від $-\pi$ до π:

$$\int_{-\pi}^{\pi} f(x)dx = \frac{a_0}{2}\int_{-\pi}^{\pi}dx + \sum_{n=1}^{\infty}\left(a_n\int_{-\pi}^{\pi}\cos nxdx + b_n\int_{-\pi}^{\pi}\sin nxdx\right)$$

З результатів обчислення інтегралів, наведених вище, випливає, що всі доданки, що зустрічаються у правій частині під знаком суми, дорівнюють нулю, тому

$$\int_{-\pi}^{\pi} f(x)dx = \pi a_0.$$

Отже,

$$a_0 = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)dx.$$

Для того щоб знайти a_n $(n=1,2,\dots)$, обидві частини цієї рівності помножимо на $\cos mx$ і проінтегруємо на відрізку $[-\pi,\pi]$. Оскільки система тригонометричних функцій ортогональна, то

$$\int_{-\pi}^{\pi}\cos mx\cos nxdx = 0; \int_{-\pi}^{\pi}\cos mx\sin nxdx = 0$$

для $\forall m,\ n\in N$, якщо $m\neq n$.

Це означає, що всі інтеграли, що зустрічаються у правій частині, будуть дорівнюють нулю. Винятком буде інтеграл, що виходить при $m = n$. Цей інтеграл дорівнює π. Тому

$$\int_{-\pi}^{\pi} f(x)\cos nxdx = a_n \int_{-\pi}^{\pi} \cos^2 nxdx = \pi a_n.$$

звідки $a_n = \int_{-\pi}^{\pi} f(x)\cos nxdx$, $n = 1, 2, \dots$

Аналогічно, помноживши обидві частини рівності на $\sin mx$ і проінтегрувавши на відрізку $[-\pi; \pi]$, одержуємо, що $b_n = \int_{-\pi}^{\pi} f(x)\sin nxdx$, $n = 1, 2, \dots$

Отже, якщо функцію $f(x)$ можна представити у вигляді тригонометричного ряду, то коефіцієнти a_0, a_n, b_n обчислюються за наведеними формулами і називаються коефіцієнтами Фур'є для функції $f(x)$ (а ряд, відповідно, рядом Фур'є для $f(x)$).

Проміжок інтегрування $[-\pi, \pi]$ для періодичної з періодом 2π функції можна замінити будь-яким проміжком $[a, a + 2\pi]$, $a \in \mathbb{R}$, довжина якого дорівнює 2π.

Функція $f(x)$ називається кусково-гладкою на відрізку $[a, b]$ якщо функція $f(x)$ і її похідна на $[a, b]$ мають скінченну кількість точок розриву першого роду.

Достатні умови розкладності функції в ряд Фур'є дає теорема Діріхле: якщо $f(x)$ – періодична з періодом 2π кусково-гладка на $[-\pi; \pi]$ функція, то її ряд Фур'є збігається у будь-якій точці цього відрізка і його сума дорівнює:

1. значенню функції $f(x)$, коли x – точка неперервності функції $f(x)$;
2. $\dfrac{f(x-0)+f(x+0)}{2}$, якщо x – точка розриву функції $f(x)$, при цьому

$$\frac{f(x-0)+f(x+0)}{2} = \frac{a_0}{2} + \sum_{n=1}^{\infty} (a_n \cos nx + b_n \sin nx).$$

Зазначимо, що на практиці найчастіше зустрічаються функції, які задовольняють умовам теореми Діріхле.

Приклад: періодичну з періодом 2π функцію $f(x) = x$, $-\pi < x < \pi$ розкласти у ряд Фур'є.

Обчислимо коефіцієнти Фур'є (використаємо **Maxima**):

```
(%i1) n:5;
(%o1) 5
(%o2) f(x):= x;
(%o2) f(x) := x
(%i3) a0:1/%pi*integrate(f(x),x,-%pi,%pi);
(%o3) 0
(%i4) for k:1 thru n do a[k]:1/%pi*integrate(f(x)*cos(k*x),x,-%pi,%pi);
(%o4) done
(%i5) for k:1 thru n do b[k]:1/%pi*integrate(f(x)*sin(k*x),x,-%pi,%pi);
(%o5) done
(%i6) for k:1 thru n do display(a[k],b[k]);
```

$a_1 = 0$ $b_1 = 2$ $a_2 = 0$ $b_2 = -1$ $a_3 = 0$ $b_3 = \frac{2}{3}$ $a_4 = 0$ $b_4 = -\frac{1}{2}$ $a_5 = 0$ $b_5 = \frac{2}{5}$

```
(%o6) done
(%i7) fun(x):=a0/2+sum(a[k]*cos(k*x),k,1,n)+sum(b[k]*sin(k*x),k,1,n);
(%o7) fun(x) := a0/2 + sum(a[k]*cos(k*x),k,1,n) + sum(b[k]*sin(k*x),k,1,n)
(%i8) wxplot2d([f(x),fun(x)], [x,-5,5], [nticks,20]);
```

Дана функція $f(x)$ задовольняє умовам теореми Діріхле, її графік у порівнянні із графіком частинної суми ряду Фур'є `fun(x)` зображений на рис. 3.12.

3.9.3 Ряди Фур'є для парних і непарних функцій

Припустімо, що $f(x)$ – непарна 2π-періодична функція. У цьому випадку $f(x)\cos nx$ – парна функція, оскільки виконується рівність $f(-x)\cos(-nx) = f(x)\cos nx$, а $f(x)\sin nx$ – непарна функція, оскільки $f(-x)\sin(-nx) = -f(x)\sin nx$. Тому коефіцієнти ряду Фур'є a_n, b_n рівні:

$$a_n = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)\cos nxdx = \frac{2}{\pi}\int_{0}^{\pi} f(x)\cos nxdx, (n = 0, 1, \dots),$$

$$b_n = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)\sin nxdx = 0 (n = 1, 2, \dots)$$

Отже, ряд Фур'є парної функції містить тільки косинуси, тобто $f(x) = \dfrac{a_0}{2} + \sum\limits_{n=1}^{\infty} a_n \cos nx$. Аналогічно, якщо $f(x)$ – непарна функція, то $f(x)\cos nx$ – непарна, а $f(x)\sin nx$ — парна функція. Тому $a_n = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)\cos nxdx = 0 (n = 0, 1, \dots)$, $b_n = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)\sin nxdx = \frac{2}{\pi}\int_{0}^{\pi} f(x)\sin nxdx, (n = 0, 1, \dots)$.

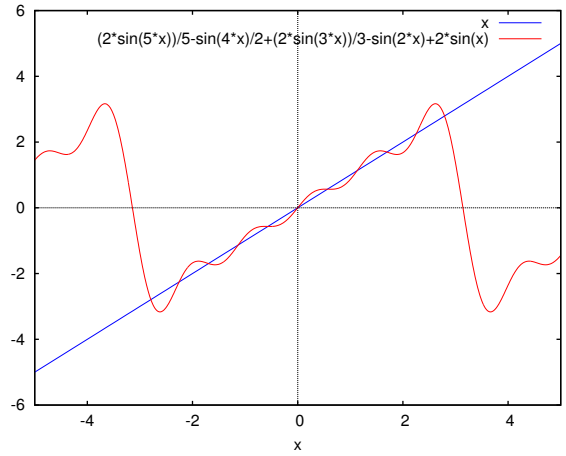


Рис. 3.12. Графік функції $y = f(x)$ і суми перших п'яти членів ряду Фур'є

Отже, ряд Фур'є непарної функції містить тільки синуси, тобто $f(x) = \sum_{n=1}^{\infty} b_n \sin nx$.

Приклад: Розкласти у ряд Фур'є періодичну з періодом 2π функцію, задану на відрізку $[-\pi, \pi]$ рівністю $f(x) = x^2$. Дана функція є парною (рис. 3.13), тому її ряд Фур'є містить тільки косинуси.

Для обчислення коефіцієнтів a_n ряду Фур'є створюємо функцію `fun`, вхідними параметрами якої є назва незалежної змінної (у прикладі це x), число доданих членів ряду (n, надалі функція викликається при $n = 5$) і символьний вираз, що визначає функцію, для якої будується розклад (f, функція `fun` викликається з $f = x^2$).

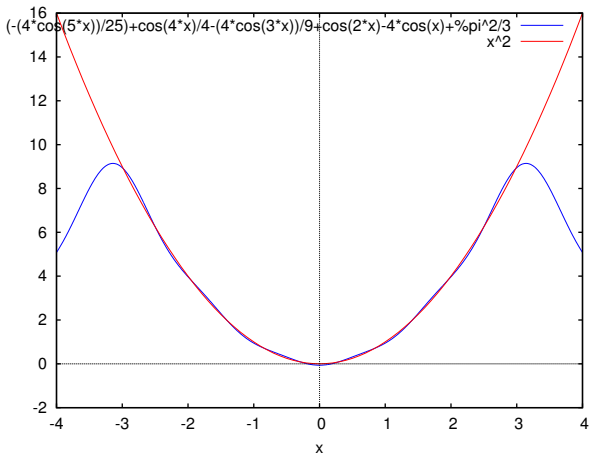


Рис. 3.13. Графік функції $y = x^2$ і суми перших п'яти членів ряду Фур'є

Приклад:

```
(%i1) fun(x,n,f):=(for k:0 thru n do a[k]:1/%pi*integrate(f*cos(k*x),x,-%pi,%pi),
a[0]/2+sum(a[k]*cos(k*x),k,1,n));
(%i2) fun(x,5,x^2);
```

(%o2) $-\frac{4\cdot\cos(5\cdot x)}{25} + \frac{\cos(4\cdot x)}{4} - \frac{4\cdot\cos(3\cdot x)}{9} + \cos(2\cdot x) - 4\cdot\cos(x) + \frac{\pi^2}{3}$

Для аналітичного обчислення коефіцієнтів ряду Фур'є функції $y = |x|$ функцію `fun` необхідно небагато змінити, передбачивши різні вирази для підінтегрального виразу на напівінтервалах $[-\pi, 0)$ і $(0, \pi]$ (вирази `f1` і `f2` у списку параметрів функції). Текст програми на макромові **Maxima**:

```
fun12(x,n,f1,f2):=(for k:0 thru n do
a[k]:1/%pi*(integrate(f1*cos(k*x),x,-%pi,0)+ integrate(f2*cos(k*x),x,0,%pi)),
a[0]/2+sum(a[k]*cos(k*x),k,1,n))$
```

Функція $y = |x|$ також є парною (рис. 3.14), тому її ряд Фур'є містить тільки косинуси.

Результати обчислення коефіцієнтів ряду Фур'є для цієї функції:

```
(%i1) fun12(x,5,-x,x);
```

(%o1) $-\frac{4\cos(5x)}{25\pi} - \frac{4\cos(3x)}{9\pi} - \frac{4\cos(x)}{\pi} + \frac{\pi}{2}$

Для побудови графіка функції $y = |x|$ створюємо функцію `fg(x)`, яку використано для побудови графіка на рис. 3.14.

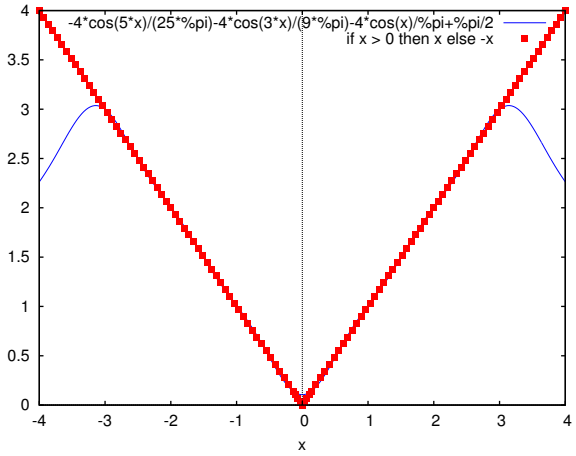


Рис. 3.14. Графік функції $y = |x|$ і суми перших п'яти членів ряду Фур'є

```
(%i3) fg(x):=if x>0 then x else -x$
```

3.9.4 Розкладання функцій у ряд Фур'є на відрізку $[0, \pi]$

Нехай $f(x)$ визначена на відрізку $[0, \pi]$. Для того, щоб функцію $f(x)$ розкласти в ряд Фур'є на цьому відрізку, довизначимо цю функцію довільним чином на інтервалі $[-\pi, 0)$.

Розгляньмо два випадки:

Функцію $f(x)$, задану на $[0, \pi]$, продовжимо на інтервал $[-\pi, 0)$ так, що нова функція $f_1(x)$, була парною:

$$f_1 = \begin{cases} f(-x), & \text{якщо} \quad x \in [-\pi, 0), \\ f(x), & \text{якщо} \quad x \in [0, \pi]. \end{cases}$$

У такому випадку кажуть, що $f(x)$ продовжена на $[-\pi, 0]$ парним чином. Оскільки $f_1(x)$ – парна на $[-\pi, \pi]$ функція, то її ряд Фур'є містить тільки косинуси:

$$f_1(x) = \frac{a_0}{2} + \sum_{n=1} \infty a_n \cos nx.$$

Оскільки на відрізку $[0, \pi]$ має місце рівність $f_1(x) = f(x)$, то ряд Фур'є для функції $f_1(x)$ буде і рядом Фур'є для $f(x)$ на $[0, \pi]$.

Функцію $f(x)$, задану на $[0, \pi]$, продовжимо на інтервал $[-\pi, 0)$ непарним чином:

$$f_2 = \begin{cases} -f(-x), & \text{якщо} \quad x \in [-\pi, 0), \\ f(x), & \text{якщо} \quad x \in [0, \pi]. \end{cases}$$

Оскільки $f_2(x)$ – непарна на $[-\pi, \pi]$ функція, то її ряд Фур'є містить тільки синуси:

$$f_1(x) = \sum_{n=1} \infty b_n \sin nx.$$

Оскільки $f_2(x) = f(x)$ при $\forall x \in [0, \pi]$, то отриманий ряд Фур'є для $f_2(x)$ і буде рядом Фур'є для $f(x)$ на $[0, \pi]$.

Приклад: Функцію $f(x) = 2x+1$, задану на відрізку $[0, \pi]$, розкласти у ряд Фур'є: 1) за косинусами; 2) за синусами.

1) Функцію $f(x)$ продовжимо на $[-\pi, 0)$ парним чином, тобто складемо нову функцію $f_1(x)$ за формулою:

$$f_1 = \begin{cases} -2x + 1, & \text{якщо} \quad x \in [-\pi, 0), \\ 2x + 1, & \text{якщо} \quad x \in [0, \pi]. \end{cases}$$

Обчислюємо коефіцієнти Фур'є для цієї функції за допомогою функції `fun12`:

```
(%i1) fleft:-2*x+1;
(%o1) 1 − 2x
(%I2) fright:2*x+1;
(%o2) 2x + 1
(%I3) fun12(x,7,fleft,fright);
```

$$\text{(\%o3)} \quad -\frac{8\cos(7\,x)}{49\,\pi} - \frac{8\cos(5\,x)}{25\,\pi} - \frac{8\cos(3\,x)}{9\,\pi} - \frac{8\cos(x)}{\pi} + \frac{2\,\pi^2 + 2\,\pi}{2\,\pi}$$

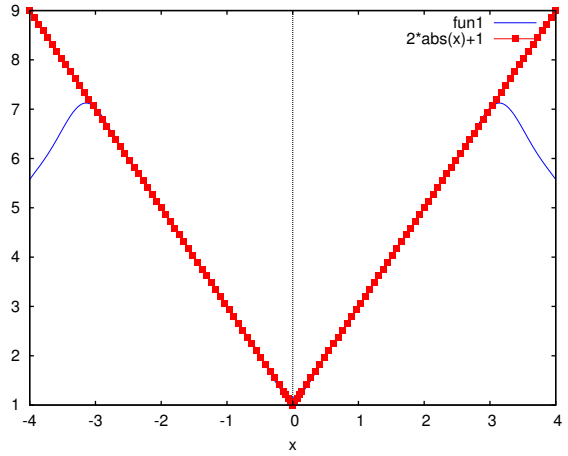


Рис. 3.15. Графік функції $y = 2x + 1$, продовженої парним чином, і суми семи членів відповідного ряду

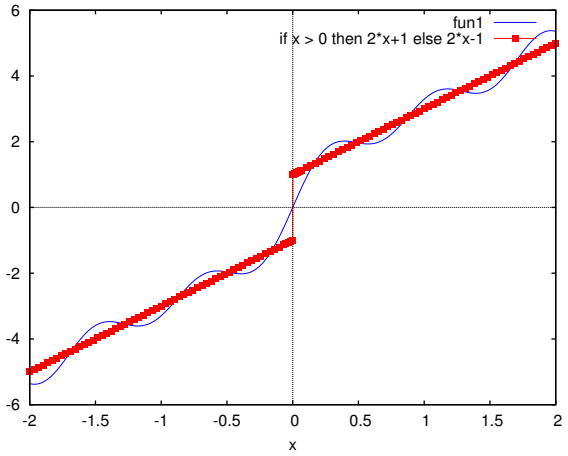


Рис. 3.16. Порівняння графіка функції $y = 2x + 1$ при непарному продовженні і суми семи членів відповідного ряду Фур'є

Графічне зіставлення результатів підсумовування ряду Фур'є і аналітичного виразу заданої функції представлені на рис. 3.15.

2) Функцію $f(x)$ продовжимо на $[-\pi, 0)$ непарним образом. Складемо нову функцію $f_2(x)$ за формулою $f_2 = \begin{cases} 2x-1, & \text{якщо} \quad x \in [-\pi, 0), \\ 2x+1, & \text{якщо} \quad x \in [0, \pi]. \end{cases}$

Обчислимо коефіцієнти Фур'є для цієї функції, використовуючи функцію `fun12sin`, аналогічну наведеній вище.

Приклад:

```
(%i1) fleft:2*x-1$
(%i2) fright:2*x+1$
(%i3) f(x):=(if x>0 then fright else fleft)$
(%i4) fun12sin(x,n,f1,f2):=(for k:1 thru n do b[k]:1/%pi*(integrate(f1*sin(k*x),x,-%pi,0)
+integrate(f2*sin(k*x),x,0,/pi)), sum(b[k]*sin(k*x),k,1,n));
(%i5) fun12sin(x,7,fleft,fright);
```

$$\text{(\%o5)} \quad \frac{\left(\frac{2\,(2\,\pi+1)}{7}+\frac{2}{7}\right)\,\sin\left(7\,x\right)}{\pi}+\frac{\left(\frac{1}{3}-\frac{2\,\pi+1}{3}\right)\,\sin\left(6\,x\right)}{\pi}+\frac{\left(\frac{2\,(2\,\pi+1)}{5}+\frac{2}{5}\right)\,\sin\left(5\,x\right)}{\pi}+\frac{\left(\frac{1}{2}-\frac{2\,\pi+1}{2}\right)\,\sin\left(4\,x\right)}{\pi}+$$
$$\frac{\left(\frac{2\,(2\,\pi+1)}{3}+\frac{2}{3}\right)\,\sin\left(3\,x\right)}{\pi}-2\,\sin\left(2\,x\right)+\frac{\left(4\,\pi+4\right)\,\sin\left(x\right)}{\pi}$$

Графічне зіставлення результатів підсумовування ряду Фур'є і аналітичного виразу заданої функції представлені на рис. 3.16.

3.9.5 Ряд Фур'є для функцій з періодом 2ℓ

Нехай $f(x)$ – періодична з періодом 2ℓ ($\ell \neq \pi$) функція, що на відрізку $[-\ell, \ell]$ задовольняє умовам теореми Діріхле. Розкладемо її на цьому відрізку у ряд Фур'є. Позначимо

$$x = \frac{\ell t}{\pi} \tag{3.1}$$

Тоді

$$f(x) = f\left(\frac{\ell t}{\pi}\right) = \varphi(t)$$

Функція $\varphi(t)$ – уже 2π-періодична функція, оскільки

$$\varphi(t+2\pi) = f\left(\frac{\ell}{\pi}(t+2\pi)\right) = f\left(\frac{\ell t}{\pi} + 2\ell\right) = f\left(\frac{\ell t}{\pi}\right) = \varphi(t).$$

Функцію $\varphi(t)$ розкладемо у ряд Фур'є на відрізку $[-\pi, \pi]$

$$\varphi(t) = f\left(\frac{\ell t}{\pi}\right) = \frac{a_0}{2} + \sum_{n=1}^{\infty}\left(a_n \cos nt + b_n \sin nt\right). \tag{3.2}$$

Коефіцієнти цього ряду обчислюються за формулами:

$$a_n = \frac{1}{\pi}\int_{-\pi}^{\pi} f\left(\frac{\ell t}{\pi}\right)\cos ntdt, \quad n = 0, 1, 2, \ldots \tag{3.3}$$

$$b_n = \frac{1}{\pi}\int_{-\pi}^{\pi} f\left(\frac{\ell t}{\pi}\right)\sin ntdt, \quad n = 1, 2, \ldots \tag{3.4}$$

Вертаючись до попередньої змінної x, з рівності (3.1) маємо $t = \frac{\pi x}{\ell}$. Тоді ряд (3.2) можна представити у вигляді

$$f(x) = \frac{a_0}{2} + \sum_{n=1}^{\infty}\left(a_n \cos\frac{n\pi x}{\ell} + b_n \sin\frac{n\pi x}{\ell}\right).$$

В інтегралах (3.3) і (3.4) зробимо заміну змінної:

$$a_n = \frac{1}{\pi}\int_{-\pi}^{\pi} f\left(\frac{\ell t}{\pi}\right)\cos ntdt = \frac{1}{\ell}\int_{-\ell}^{\ell} f(x)\cos\frac{n\pi x}{\ell}dx, \quad n = 0, 1, 2, \ldots$$

$$b_n = \frac{1}{\pi}\int_{-\pi}^{\pi} f\left(\frac{\ell t}{\pi}\right)\sin ntdt = \frac{1}{\ell}\int_{-\ell}^{\ell} f(x)\sin\frac{n\pi x}{\ell}dx, \quad n = 1, 2, \ldots$$

Якщо $f(x)$ – парна на $[-\ell, \ell]$ функція, то $b_n = 0$ $(n = 1, 2, \ldots)$, а $a_n = \frac{2}{\ell}\int_0^{\ell} f(x)\cos\frac{n\pi x}{\ell}dx$, $(n = 0, 1, \ldots)$, ряд Фур'є такої функції має вигляд:

$$f(x) = \frac{a_0}{2} + \sum_{n=1}^{\infty} a_n \cos\frac{n\pi x}{\ell}.$$

Якщо $f(x)$ – непарна на $[-\ell, \ell]$ функція, то $a_n = 0$ $(n = 0, 1, 2, \ldots)$, а $b_n = \frac{2}{\ell}\int_0^{\ell} f(x)\sin\frac{n\pi x}{\ell}dx$, $(n = 1, 2, \ldots)$, ряд Фур'є такої функції має вигляд:

$$f(x) = \sum_{n=1}^{\infty} b_n \sin\frac{n\pi x}{\ell}.$$

Приклад: Розкласти в ряд Фур'є періодичну з періодом $T = 2$ функцію $f(x)$, задану формулою

$$f(x) = \begin{cases} x, & \text{якщо} \quad 0 < x \leqslant 1; \\ 0, & \text{якщо} \quad -1 < x \leqslant 0. \end{cases}$$

Ця функція на відрізку $[-1, 1]$ задовольняє умовам теореми Діріхле. Ряд Фур'є для даної функції:

$$f(x) = \frac{1}{4} - \frac{2}{\pi^2}\sum_{k=0}^{\infty}\frac{\cos(2k+1)x}{(2k+1)^2} + \frac{1}{\pi}\sum_{k=1}^{\infty}\frac{(-1)^{k+1}}{k}\sin k\pi x.$$

Сума цього ряду у точках $x = \pm 1, \pm 3, \ldots$ дорівнює $\frac{1}{2}$.

Розгляньмо видозміну функції **Maxima**, необхідної для обчислення коефіцієнтів ряду Фур'є для функції з періодом $[-\ell, \ell]$. Розгляньмо текст функції `fun12l`:

```
fun12l(x,n,l,f1,f2):=(for k:0 thru n do
a[k]:1/l*(integrate(f1*cos(%pi*k*x/l),x,-l,0)
+integrate(f2*cos(%pi*k*x/l),x,0,l)), for k:1 thru n do
b[k]:1/l*(integrate(f1*sin(%pi*k*x/l),x,-l,0) + integrate(f2*sin(%pi*k*x/l),x,0,l)),
a[0]/2+sum(a[k]*cos(%pi*k*x/l),k,1,n)+ sum(b[k]*sin(%pi*k*x/l),k,1,n))$
```

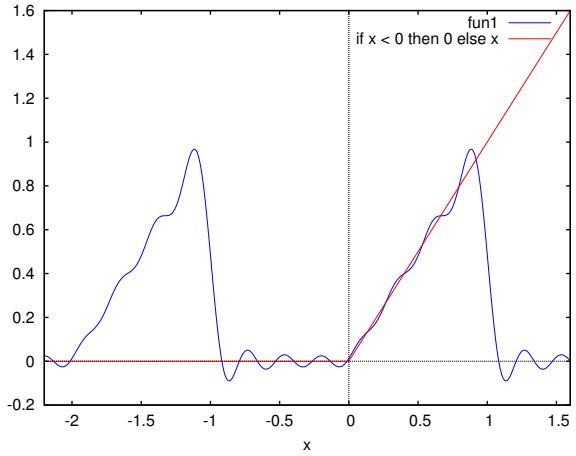


Рис. 3.17. Графік функції $f(x)$ і суми перших семи членів ряду Фур'є

Основна зміна порівняно з варіантами, наведеними вище – використання тригонометричних функцій $\cos\frac{\pi kx}{\ell}$ та $\sin\frac{\pi kx}{\ell}$.

Виведення за допомогою **Maxima** для перших семи членів ряду Фур'є:

```
(%i6) fun12l(x,7,1,0,x);
```
(%o6) $\frac{\sin(7\pi x)}{7\pi}-\frac{2\cos(7\pi x)}{49\pi^2}-\frac{\sin(6\pi x)}{6\pi}+\frac{\sin(5\pi x)}{5\pi}-\frac{2\cos(5\pi x)}{25\pi^2}-\frac{\sin(4\pi x)}{4\pi}+\frac{\sin(3\pi x)}{3\pi}-\frac{2\cos(3\pi x)}{9\pi^2}-\frac{\sin(2\pi x)}{2\pi}+\frac{\sin(\pi x)}{\pi}-\frac{2\cos(\pi x)}{\pi^2}+\frac{1}{4}$

Для побудови графіка власне аналізованої функції (її представляє кусково-неперервна функція $f(x)$) і частинної суми її ряду Фур'є з результатів розкладання формуємо нову функцію $g(x)$, після чого стандартною командою будуємо графіка:

```
(%i7) g(x):=''%$
(%i8) f(x):=(if x<0 then 0 else x)$
(%i9) wxplot2d([g(x),f(x)], [x,-2.2,1.6]);
```

Графічна ілюстрація, що показує зіставлення розглянутої функції і ряду Фур'є на заданому відрізку – на рис. 3.17.

3.9.6 Комплексна форма ряду Фур'є

Нехай функція $f(x)$ на $[-\pi,\pi]$ розкладена в ряд Фур'є

$$f(x)=\frac{a_0}{2}+\sum_{n=1}^{\infty}\left(a_n\cos nx+b_n\sin nx\right). \tag{3.5}$$

Скористаємося формулами Ойлера:

$$\cos nx=\frac{e^{inx}+e^{-inx}}{2};\sin nx=\frac{e^{inx}-e^{-inx}}{2i}$$

Підставимо ці вирази до ряду (3.5), маємо:

$$\begin{aligned}f(x) &= \frac{a_0}{2}+\sum_{n=1}^{\infty}\left(a_n\frac{e^{inx}+e^{-inx}}{2}+b_n\frac{e^{inx}-e^{-inx}}{2i}\right)=\\ &= \frac{a_0}{2}+\sum_{n=1}^{\infty}\left(a_n\frac{e^{inx}+e^{-inx}}{2}-ib_n\frac{e^{inx}-e^{-inx}}{2}\right)=\\ &= \frac{a_0}{2}+\sum_{n=1}^{\infty}\left(\frac{a_n-ib_n}{2}\cdot e^{inx}+\frac{a_n+ib_n}{2}e^{-inx}\right).\end{aligned}$$

Позначимо:

$$\frac{a_0}{2}=c_0,\ \frac{a_n-ib_n}{2}=c_n,\ \frac{a_n+ib_n}{2}=c_{-n}$$

Тоді

$$\begin{aligned}f(x) &= c_0+\sum_{n=1}^{\infty}\left(c_ne^{inx}+c_{-n}e^{-inx}\right)=\\ &= c_0+\sum_{n=1}^{\infty}c_ne^{inx}+\sum_{n=1}^{\infty}c_{-n}e^{-inx}=\\ &= c_0+\sum_{n=1}^{\infty}c_ne^{inx}+\sum_{n=-\infty}^{-1}c_ne^{inx}=\sum_{n=-\infty}^{\infty}c_ne^{inx}.\end{aligned}$$

Отже

$$f(x) = \sum_{n=-\infty}^{\infty} c_n e^{inx} \tag{3.6}$$

Вираження (3.6) називається комплексною формою ряду Фур'є функції $f(x)$ з комплексними коефіцієнтами Фур'є c_n. Коефіцієнти Фур'є c_n обчислюються за формулами ($n = 0, \pm1, \pm2, \dots$):

$$\begin{aligned} c_n &= \frac{1}{2}(a_n - ib_n) = \frac{1}{2\pi}\int_{-\pi}^{\pi} f(x)\left[\cos nx - i\sin nx\right]dx = \\ &= \int_{-\pi}^{\pi} f(x)\left[\cos(-nx) + i\sin(-nx)\right]dx = \int_{-\pi}^{\pi} f(x)e^{-inx}dx. \end{aligned}$$

Якщо $f(x)$ – періодична з періодом 2ℓ функція, то її комплексний ряд Фур'є має вигляд:

$$f(x) = \sum_{n=-\infty}^{\infty} c_n e^{\frac{in\pi x}{\ell}},$$

а коефіцієнти Фур'є визначаються за формулою

$$c_n = \frac{1}{2\ell} = \int_{-\ell}^{\ell} f(x)e^{-\frac{in\pi x}{\ell}}dx.$$

Приклад: Розкласти у ряд Фур'є з комплексними коефіцієнтами періодичну з періодом $\ell = 2$ функцію, задану на відрізку $[-1, 1]$ рівністю $f(x) = x^2$.

```
(%i1) n:5$ f:x^2$ l:1$ c(k):=
1/2/l*integrate(f*exp(-%i*%pi*k*x/l),x,-l,l)$ z:makelist(k-6, k, 1, 2*n+1)$
cr:makelist(c(z[k]),k,1,2*n+1)$ fk:makelist(cr[k]*exp(%i*%pi*z[k]*x/l),k,1,2*n+1)$
g:sum(fk [k],k,1,2*n+1)$ gend:trigreduce(ratsimp(rectform(g)));
(%o9) (−144 cos(5πx)+225 cos(4πx)−400 cos(3πx)+900 cos(2πx)−3600 cos(πx)+300 π²)/(900 π²)
```

У цьому прикладі члени частинної суми ряду Фур'є представляються списком. У представленому обчисленні $z = -5, -4, \dots, 4, 5$. Список `cr` містить коефіцієнти ряду у комплексній формі (при підсумовуванні від $-n$ до n індекс елемента ряду втримується в `z[k]`). Власне члени ряду Фур'є скомпоновані у список `fk`, після підсумовування якого одержуємо суму ряду (вираз `g`). Для побудови графіка `g(x)` слід спростити вираз `g` (див. приклад, результат спрощення – вираз `gend`). Очевидно, що для перегляду проміжних результатів (вони досить об'ємні) термінальні символи $ можна замінити на ;.

3.9.7 Додаткові можливості: пакет `fourie`

Пакет розширення `fourie` призначений для розрахунку коефіцієнтів тригонометричних рядів Фур'є, а також інтеграла Фур'є.

Функції, що входять до складу пакета, дозволяють знаходити точне аналітичний вираз всіх, а не перших декількох коефіцієнтів ряду Фур'є.

Функція fourier надає змогу обчислити коефіцієнти ряду Фур'є (синтаксис виклику: `fourier(f,x,p)`), що повертає список коефіцієнтів Фур'є $f(x)$, визначних на інтервалі $[-p, p]$. Власне ряд Фур'є надає змогу побудувати функція `fourexpand` (синтаксис виклику `fourexpand(l,x,p,limit)`), що конструює і повертає ряд Фур'є, використовуючи список коефіцієнтів Фур'є `l` (`limit` може бути і нескінченним, рівним `%inf`).

Коефіцієнти рядів Фур'є за синусами і за косинусами обчислюються функціями `fourcos(f,x,p)`, `foursin(f,x,p)` (синтаксиси аналогічні функції `fourier`).

Обчислення і підставляння $\cos\pi n$ і $\sin\pi n$ здійснюється спеціальною функцією `foursimp(l)`. Керування підстановкою здійснюється за допомогою прапорів `sinnpiflag` і `cosnpiflag` (якщо вони встановлені в `true`, обчислення і підстановка виконуються, це режим типово).

Для керування процесом розкладання різних функцій у ряд Фур'є передбачені наступні функції:

1. `remfun`. Синтаксис виклику `remfun(f,expr)` або `remfun(f,expr,x)`. Ця функція надає змогу замінити всі входження функції `f(arg)` у виразі `expr` на `arg` (у формі `remfun(f,expr,x)` заміна виконується, тільки якщо `arg` містить `x`);

2. `funp`. Ця функція (синтаксис виклику `funp(f,expr)` або `funp(f,expr,x)`) повертає `true`, якщо вираз `expr` містить функцію `f` або конкретно `f(x)`;

3. `absint`. Дана функція надає змогу обчислити невизначений або визначений інтеграл абсолютних значень функції `f` (її визначення може включати вираз `abs(x)`, `abs(sin(x))`, `abs(a)*exp(-abs(b)*abs(x))`). Синтаксис виклику `absint(f,x,halfplane)` (`halfplane=(pos,neg,both)` – частина числової осі), `absint(f,x)` (невизначений інтеграл за додатною піввіссю), `absint(f,x,a,b)` (визначений інтеграл).

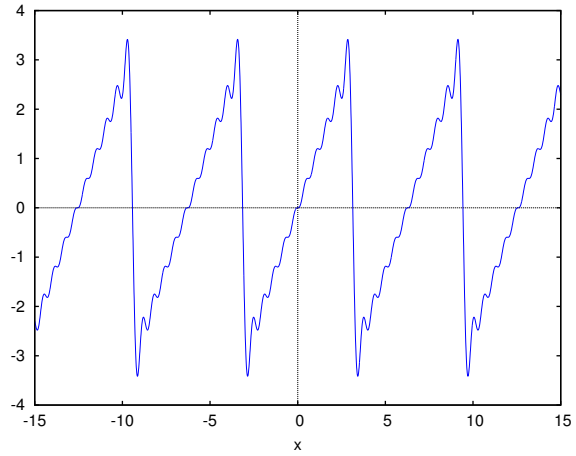


Рис. 3.18. Графік часткової суми ряду Фур'є для функції $f(x) = x$, побудованої за допомогою пакета `fourie`

Загальну форму ряду Фур'є (після підставляння і спрощення) надає змогу побудувати функція `totalfourier(f, x, p)`.

Коефіцієнти інтеграла Фур'є на інтервалі $(-\infty, \infty)$ надає змогу обчислити функція `fourint(f,x)`, інтеграла за косинусами або синусами на інтервалі $(0, \infty)$ – функції `fourintcos(f,x)` і `fourintsin(f,x)` відповідно.

Для використання пакета fourie його необхідно попередньо завантажити командою `load("fourie")`.

Приклади використання пакета fourie (графік отриманої функції наведено на рис. 3.18):

```
(%i1) load("fourie")$ fourier(x,x,%pi);
```
(%t2) $a_0 = 0$
(%t3) $a_n = 0$
(%t4) $b_n = \dfrac{2\left(\frac{\sin(\pi n)}{n^2} - \frac{\pi\cos(\pi n)}{n}\right)}{\pi}$
(%o4) $[\%t2, \%t3, \%t4]$
```
(%i5) foursimp(%);
```
(%t5) $a_0 = 0$
(%t6) $a_n = 0$
(%t7) $b\_n = -\dfrac{2(-1)^n}{n}$
(%o7) $[\%t5,\ \%t6,\ \%t7]$
```
(%i8) fourexpand(%,x,%pi,10);
```
(%o8) $-\dfrac{\sin(10\,x)}{5} + \dfrac{2\sin(9\,x)}{9} - \dfrac{\sin(8\,x)}{4} + \dfrac{2\sin(7\,x)}{7} - \dfrac{\sin(6\,x)}{3} + \dfrac{2\sin(5\,x)}{5} - \dfrac{\sin(4\,x)}{2} + \dfrac{2\sin(3\,x)}{3} - \sin(2\,x) + 2 + sin(x)$

3.9.8 Додаткові можливості: узагальнені ряди Фур'є

Як зазначалося вище, поряд із тригонометричною ортонормованою системою функцій досить широко використовуються і інші (зокрема, поліноми Лежандра, Чебишова, Ерміта тощо). Розгляньмо подання функції узагальненим рядом Фур'є за поліномами Лежандра.

Обчислення значень ортогональних поліномів у **Maxima** здійснюється за допомогою пакета `orthopoly`, що надає змогу оперувати поліномами Чебишова, Лежандра, Ерміта, Якобі тощо, а також рядом сферичних функцій.

Інтегрована на інтервалі $(-1, 1)$ кусочно-неперервна функція може бути представлена узагальненим рядом Фур'є (у цьому випадку – за поліномами Лежандра):

$$f(x) = \sum_{n=0}^{\infty} c_n P_n(x),$$

де $P_n(x)$ – поліном Лежандра степеня n, c_n – коефіцієнти Фур'є для розкладання за поліномами Лежандра. Значення c_n обчислюються за формулою:

$$c_n = \frac{2n+1}{2} \int_{-1}^{1} f(x) P_n(x) dx.$$

Приклад обчислення розкладання функції $y = e^x$ на інтервалі $(-1, 1)$ у ряд за поліномами Лежандра представлений наступними командами:

```
(%i1) load(orthopoly)$ n:5$ f:exp(x)$
```

```
l:1$ c(m):=(2*m+1)/2*integrate(f*legendre_p(m,x),x,-l,l)$ z:makelist(k-1,k,1,n+1)$
cr:makelist(c(z[k]),k,1,n+1)$ fk:makelist(cr[k]*legendre_p(z[k],x),k,1,n+1)$ g: sum(fk[k],k,1,n+1)$
```

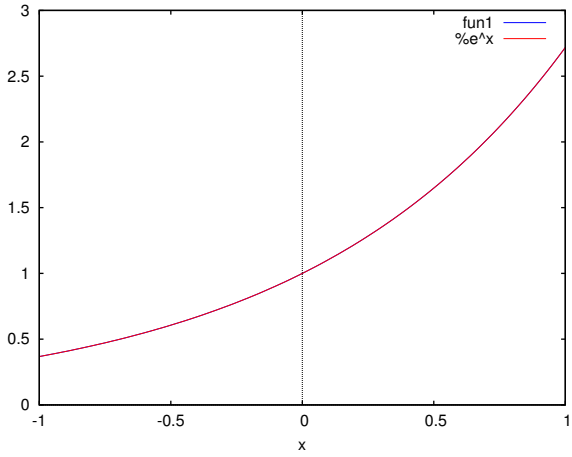


Рис. 3.19. Графік частинної суми узагальненого ряду Фур'є для функції $f(x) = e^x$

Графік отриманого виразу g у порівнянні з функцією e^x показаний на рис. 3.19.

Як видно з рисунка, графіки експоненти і отриманого розкладу збігаються. У збігу результатів можна переконатися, зіставивши вираз g (після спрощення) і розкладання експоненти у ряд Тейлора.

Розділ 4

Обчислювальні методи і програмування з Maxima

4.1 Програмування на вбудованій макромові

4.1.1 Умовні оператори

Основна форма умовного оператора: `if` $cond_1$ `then` $expr_1$ `else` $expr_0$. Якщо умова $cond_1$ виконується, то виконується вираз $expr_1$, інакше – виконується вираз $expr_0$. Пакет **Maxima** надає змогу використати різні форми оператора `if`, наприклад: `if` $cond_1$ `then` $expr_1$ `elseif` $cond_2$ `then` $expr_2$ `elseif ...else` $expr_0$

Якщо виконується умова $cond_1$, то виконується вираз $expr_1$, інакше – перевіряється умова $cond_2$, і якщо вона виконується – виконується вираз $expr_2$, тощо. Якщо жодна з умов не виконується – виконується вираз $expr_0$.

Альтернативні вирази $expr_1$, $expr_2$, ..., $expr_n$ – довільні вирази **Maxima** (зокрема вкладені оператори `if`). Умови – дійсно або потенційно логічні вирази, що зводяться до значень `true` або `false`. Спосіб інтерпретації умов залежить від значення прапорця `prederror`. Якщо `prederror=true`, виводиться помилка, якщо значення якогось із виразів $cond_1$, ..., $cond_n$ відрізняється від `true` або `false`. Якщо `prederror=false` і значення якогось із виразів $cond_1$, ..., $cond_n$ відрізняється від `true` або `false`, результат обчислення `if` – умовний вираз.

4.1.2 Оператори циклу

Для виконання ітерацій використовується оператор `for`. Можуть використовуватися три варіанти його виклику, що відрізняються умовою закінчення циклу:

`for` *variable*`:` *init_value* *step* *increment* `thru` *limit* `do` *body*

`for` *variable*`:` *init_value* *step* *increment* `while` *condition* `do` *body*

`for` *variable*`:` *init_value* *step* *increment* `unless` *condition* `do` *body*

Тут *variable* – змінна циклу; *init_value* – початкове значення; *increment* – крок (типово дорівнює 1); *limit* – кінцеве значення змінної циклу; *body* – оператори тіла циклу.

Ключові слова `thru`, `while`, `unless` вказують на спосіб завершення циклу:

- за досягнення змінною циклу значення `limit`;
- поки виконується умова `condition`;
- поки не буде досягнута умова `condition`.

Параметри *init_value*, *increment*, *limit*, і *body* можуть бути довільними виразами. Контрольна змінна по завершенні циклу має бути додатною (при цьому початкове значення може бути і від'ємним). Вирази *limit*, *increment*, умови завершення (*condition*) обчислюються на кожному кроці циклу, тому їхня складність впливає на час виконання циклу.

При нормальному завершенні циклу величина, що повертається, – атом `done`. Примусовий вихід із циклу здійснюється за допомогою оператора `return`, що може повертати довільне значення.

Контрольна змінна циклу – локальна усередині циклу, тому її зміна у циклі не впливає на контекст (навіть при наявності поза циклом змінної з тією ж назвою).

Приклади:

```
(%i1) for a:-3 thru 26 step 7 do display(a)$
```

$a = -3$
$a = 4$
$a = 11$
$a = 18$
$a = 25$

```
(%i2) s:0$ for i:1 while i <= 10 do s: s+i;
(%o3) done
(%i4) s;
(%o4) 55
(%i5) series: 1$ term: exp(sin(x))$
(%i7) for p:1 unless p > 7 do (term: diff (term, x)/p, series: series + subst (x=0, term)*x^p)$
(%i8) series;
```

(%o8) $\frac{x^7}{90} - \frac{x^6}{240} - \frac{x^5}{15} - \frac{x^4}{8} + \frac{x^2}{2} + x + 1$

```
(%i9) for count: 2 next 3*count thru 20 do display(count)$
```

$count = 2$
$count = 6$
$count = 18$

Умови ініціалізації і завершення циклу можна пропускати. **Приклад** (цикл без явного зазначення змінної циклу):

```
(%i10) x:1000;
(%o10) 1000
(%i11) thru 20 do x: 0.5*(x + 5.0/x)$
(%i12) x;
(%o12) 2.23606797749979
(%i12) float(sqrt(5));
(%o12) 2.23606797749979
```

За 20 ітерацій досягається точне значення $\sqrt{5}$.

Трохи витонченіший приклад – реалізація методу Ньютона для рівняння з одною невідомою (обчислюється та ж величина кореня з п'яти):

```
(%i1) newton(f,x):=([y,df,dfx], df:diff(f('x),'x), do (y: ev(df), x: x-f(x)/y,
if abs ($f(x)$) < 5e-6 then return (x)));
(%i2) f(x):=x^2-5;
(%o2) f(x) := x^2 - 5
(%i3) float(newton(f,1000));
(%o3) 2.236068027062195
```

Ще одна форма оператора циклу характеризується вибором значень змінної циклу із заданого списку. Синтаксис виклику: `for` *variable* `in` *list end_tests* `do` *body*

Перевірки умови завершення *end_tests* до вичерпання списку *list* може і не бути.

Приклад:

```
(%i1) a:[];
(%o1) []
(%i2) for f in [1,4,9,16] do a:cons(sqrt(f),a)$
(%i3) a;
(%o3) [4, 3, 2, 1]
```

4.1.3 Блоки

Як в умовних виразах, так і у циклах замість простих операторів можна писати складені оператори, тобто блоки. Стандартний блок має вигляд: `block([r,s,t],r:1,s:r+1,t:s+1,x:t,t*t);` Спочатку йде список локальних змінних блоку (глобальні змінні з тими ж назвами ніяк не пов'язані із цими локальними змінними). Список локальних змінних може бути порожнім. Далі йде набір операторів. Спрощений блок має вигляд: `(x:1,x:x+2,a:x);` Зазвичай, у циклах і в умовних виразах застосовують саме цю форму блоку. Значенням блоку є значення останнього з його операторів. Усередині даного блоку допускаються оператор переходу на мітку і оператор `return`. Оператор `return` припиняє виконання поточного блоку і повертає як значення блоку свій аргумент `block([],x:2,x:x*x, return(x), x:x*x);`

За відсутності оператора переходу на мітку, оператори в блоці виконуються послідовно. (У цьому випадку слово «мітка» означає аж ніяк не мітку типу «%i5» або «%o7»). Оператор `go` виконує перехід на мітку, розташовану у цьому ж блоці:

```
(%i1) block([a],a:1, mitka, a:a+1, if a=1001 then return(-a),go(mitka));
(%o1) -1001
```

У цьому блоці реалізований цикл, що завершується по досягненні «змінної циклу» значення 1001. Міткою може бути довільний ідентифікатор.

Варто мати на увазі, що цикл сам собою є блоком, так що (на відміну від мови **C**) перервати виконання циклів (особливо вкладених циклів) за допомогою оператора `go` неможливо, тому що оператор `go` і мітка виявляться в різних блоках. Те ж саме стосується і оператора `return`. Якщо цикл, розташований усередині блоку, містить оператор `return`, то при виконанні оператора `return` відбудеться вихід із циклу, але не вихід із блоку:

```
(%i1) block([],x:for i:1 thru 15 do
if i=2 then return(555),display(x),777);
x = 555
(%o1) 777
(%i2) block([],x:for i:1 thru 15 do
if i=52 then return(555),display(x),777);
x = done
(%o2) 777
```

Якщо необхідно вийти з декількох вкладених блоків відразу (або декількох блоків і циклів відразу) і при цьому повернути деяке значення, то варто застосовувати блок `catch`:

```
(%i3) catch(block([],a:1,a:a+1, throw(a),a:a+7),a:a+9);
(%o3) 2
(%i4) a;
(%o4) 2
(%i5) catch(block([],for i:1 thru 15 do if i=2 then throw(555)),777);
(%o5) 555
```

У даному блоці виконання циклу завершується, як тільки значення `i` досягає 2. Значення, що повертається блоком `catch`, дорівнює 555.

```
(%i6) catch(block([],for i:1 thru 15 do if i=52 then throw(555)),777);
(%o6) 777
```

У даному блоці виконання цикл виконується повністю, і значення, що повертається блоком `catch`, дорівнює 777 (умови виходу із циклу за допомогою `throw` не досягаються).

Оператор `throw` – аналог оператора `return`, але він обриває не поточний блок, а всі вкладені блоки аж до першого виявлено блоку `catch`.

Нарешті, блок `errcatch` надає змогу перехоплювати деякі (на жаль, не всі!) з помилок, які в нормальній ситуації привели б до завершення обчислень.

Приклад:

```
(%i1) errcatch(a:1, b:0, log(a/b), c:7);
expt: undefined: 0 to a negative exponent.
(%o1) []
(%i2) c;
(%o2) c
```

Виконання послідовності операцій переривається на першій операції, що призводить до помилки. Інші вирази блоку не виконуються (значення із залишається невизначеним). Повідомлення про виниклу помилку може бути виведено функцією `errormsg()`.

4.1.4 Функції

Поряд з найпростішим способом завдання функції, **Maxima** допускає створення функції у вигляді послідовності операторів: `f(x):=(`$expr_1$`,` $expr_2$`,` $expr_n$`)`; Значення, що повертається функцією – значення останнього виразу $expr_n$.

Щоб використати оператор `return` і змінити значення, що повертається, залежно від логіки роботи функції, слід застосовувати конструкцію `block`, наприклад: `f(x):=block([],` $expr_1$`, ..., if (a>10) then return(a), ...,` $expr_n$`)`.

При $a > 10$ виконується оператор `return` і функція повертає значення a, у протилежному випадку – значення виразу $expr_n$.

Формальні параметри функції або блоку – локальні, і є видимими тільки усередині них. Крім того, при заданні функції можна оголосити локальні змінні (у квадратних дужках на початку оголошення функції або блоку).

Приклад:

`block ([a: a],` $expr_1$`, ..., a: a+3,` $expr_n$`)`

У цьому випадку при оголошенні блоку у локальній змінній a зберігається значення глобальної змінної a, визначеної поза блоком.

Приклад:

```
(%i1) f(x):=([a:a],if a>0 then 1 else (if a<0 then -1 else 0));
(%o1) f(x) := ([a : a], if a > 0 then 1 else if a < 0 then -1 else 0)
(%i2) a:1;
(%o2) 1
(%i3) f(0);
(%o3) 1
```

```
(%i4) a:-4;
(%o4) −4
(%i5) f(0);
(%o5) −1
(%i6) a:0;
(%o6) 0
(%i7) f(0);
(%o7) 0
```

У цьому прикладі значення змінної a задається поза тілом функції, але результат, що повертає нею, залежить від значення a.

Початкові значення локальних змінних функції можуть задаватися двома способами:

- Задання функції `f(x):=(`$expr_1$`, ..., `$expr_n$`);`, виклик функції `f(1);` – початкове значення локальної змінної x дорівнює 1.
- Задання блоку `block([x:1],` $expr_1$`, ...,` $expr_n$`)`, при цьому початкове значення локальної змінної x також дорівнює 1.

Поряд з іменованими функціями, **Maxima** надає змогу використати і безіменні функції (лямбда-функції). Синтаксис використання лямбда-виразів (правда, при використанні з лямбда- виразами все-таки асоціюється назва – див. приклад):

`f1:lambda([`x_1`, ..., `x_m`],` $expr_1$`, ...,` $expr_n$`)`
`f2:lambda([[L]],` $expr_1$`, ...,` $expr_n$`)`
`f3:lambda([`x_1`, ..., `x_m`, [L]],` $expr_1$`, ...,` $expr_n$`)`

Приклад:

```
(%i1) f:lambda([x], x^2);
(%o1) lambda([x], x^2)
(%i2) f(a);
(%o2) a^2
```

Складніший приклад (лямбда-вираз можуть використовуватися у контексті, коли очікується назва функції):

```
(%i3) lambda([x],x^2)(a);
(%o3) a^2
(%i4) apply(lambda([x],x^2),[a]);
(%o4) a^2
(%i5) map(lambda([x],x^2),[a,b,c,d,e]);
(%o5) [a^2, b^2, c^2, d^2, e^2]
```

Аргументи лямбда-виразів – локальні змінні. Інші змінні при обчисленні лямбда-виразів розглядаються як глобальні. Винятки позначаються спеціальним символом – прямими лапками (див. лямбда-функцію $g2$ у прикладі).

```
(%i6) a:%pi$ b:%e$ g:lambda([a],a*b);
(%o8) lambda([a], a b)
(%i9) b:%gamma$ g(1/2);
(%o10) γ/2
(%i11) g2:lambda([a],a*''b);
(%o11) lambda([a], a γ)
(%i12) b:%e$ g2(1/2);
(%o13) γ/2
```

Лямбда-функції можуть бути вкладеними. При цьому локальні змінні зовнішнього виразу доступні як глобальні для внутрішнього (однакові назви змінних маскуються).

Приклад:

```
(%i1) h:lambda([a,b],h2:lambda([a],a*b),h2(1/2));
(%o1) lambda([a, b], h2 : lambda([a], a b), h2(1/2))
(%i2) h(%pi,%gamma);
(%o2) γ/2
```

Подібно звичайним функціям, лямбда-функції можуть мати список параметрів змінної довжини.

Приклад:

```
(%i1) f:lambda([aa,bb,[cc]],aa*cc+bb);
(%o1) lambda([aa, bb, [cc]], aacc + bb)
(%i2) f(3,2,a,b,c);
(%o2) [3a + 2, 3b + 2, 3c + 2]
```

Список `[cc]` при виклику лямбда-функції `f` включає три елементи: `[a,b,c]`. Формула для розрахунку `f` застосовується до кожного елемента списку.

Локальні змінні можуть бути оголошені і за допомогою функції `local` (змінні v_1, v_2, ..., v_n оголошуються локальними викликом `local(`v_1`, `v_2`, ..., `v_n`)` незалежно від контексту).

4.1.5 Транслятор і компілятор у Maxima

Визначивши ту або іншу функцію, можна помітно прискорити її виконання, якщо її відтранслювати або компілювати. Це відбувається тому, що якщо ви не відтранслювали і не компілювали визначену вами функцію, то при кожному черговому її виклику **Maxima** щораз заново виконує ті дії, які входять у визначення функції, тобто фактично розбирає відповідний вираз на рівні синтаксису **Maxima**.

4.1.5.1 Функція `translate`

Функція translate транслює функцію **Maxima** мовою **Lisp**. Наприклад, вираз: $f(x) := 1+x+x^2+x^3+x^4+x^5+x^6+x^7$ транслюється командою: `translate(f);`. Після цього функція, як правило, починає обчислюватися швидше.

Приклад, що ілюструє виграш за часом після трансляції функції:

```
(%i1) f(n):=block([sum,k],sum:0, for k:1 thru n do (sum:sum+k^2),sum)$
```

Функція $f(n)$, організована у вигляді блоку, надає змогу обчислити суму $\sum_{k=1}^{n} k^2$.

Для виконання тестів використовувався той самий комп'ютер (**Maxima** 5.35.1). При безпосередньому виклику функції `f` час обчислення `f(1000000)` склав 27,609 с, після трансляції – 1,25 с. Для оцінки часу обчислення було використано функцію `time`.

```
(%i2) f(1000000);
(%o2) 333333833333500000
(%i3) time(%o2);
(%o3) [60.633]
(%i4) translate(f);
(%o4) [f]
(%i5) f(1000000);
(%o5) 333333833333500000
(%i6) time(%o5);
(%o6) [15.214]
```

Функція `time(%o1, %o2, ...)` повертає список періодів часу в секундах, витрачених для обчислення результатів `%o1`, `%o2`, ... Аргументом функції `time` можуть бути тільки номери рядків виведення, для будь-яких інших змінних функція повертає значення `unknown`.

4.1.5.2 Функція `compile`

Функція `compile` спочатку транслює функцію **Maxima** мовою **Lisp**, а потім компілює цю функцію **Lisp** до двійкових кодів і завантажує їх до пам'яті.

Приклад:

```
(%i9) compile(f);
(%o9) [f]
```

Після цього функція (як правило) починає обчислюватися ще швидше, ніж після трансляції. Наприклад, після компіляції функції `f` з останнього прикладу час обчислення `f(1000000)` склав 12.144 с.

Варто мати на увазі, що як при трансляції, так і при компіляції **Maxima** намагається оптимізувати функцію за швидкістю виконання. Однак **Maxima** працює переважно із цілими числами довільної довжини або текстових виразів. Тому при роботі з більшими за об'ємом функціями можуть виникнути проблеми, пов'язані з перетворенням типів даних. У цьому випадку варто відмовитися від трансляції або компіляції, або переписати функцію, упорядкувавши використання типів.

Приклад: Розгляньмо дві функції, що обчислюють те саме вираз. У функції `f2` явно зазначено, що функція повертає дійсні значення (у форматі із рухомою крапкою).

```
f1(x,n):=block([sum,k], sum:1, for k:1 thru n do (sum:sum+1/x^k),sum)$
f2(x,n):=block([sum,k], mode_declare([function(f2),x], float), sum:1,
for k:1 thru n do (sum:sum+1/x^k),sum)$
```

Час виконання функції `f1` при запуску `f1(5, 10000)` склав 2,967 с. Після компіляції час виконання склав 2,449 з, після трансляції – 2,518 с.

При використанні функції з декларованим типом результату (`f2`) час виконання `f2(5, 10000)` виявився приблизно таким самим, як і `f1` (2,994 із замість 2,967 с). Час виконання тієї ж функції після трансляції (2,135 с) або компіляції (2,091 с) відповідає вказаній вище тенденції. Варто врахувати, що у цьому випадку результат розрахунку – раціональне число. Перетворення його до форми із рухомою крапкою при обчисленні чергового значення суми вимагає додаткових обчислювальних витрат. При звертанні до `f2` з дійсними аргументами `f2(5.0,10000.0)` час обчислень склав усього 2,574 с.

Для функції, що повертає результат, що представляється у вигляді числа із рухомою крапкою, компіляція або трансляція може дати зменшення часу обчислення у кілька разів.

Приклад: Розгляньмо функцію, що обчислює дійсний вираз (у цьому випадку додаються ірраціональні числа)

```
f3(x,n):=block([sum,k],
mode_declare ([function (f3),x], float),
sum:1, for k:1 thru n do (sum:sum+sqrt(x^k)),sum)$
```

Час обчислення виразу `f3(5, 2000)` для некомпільованої і не відтрансльованої функції склав 4,953 с, після трансляції час обчислення `f3(5, 2000)` склав 0,03 с, після компіляції – 0,02 с.

Розгляньмо ще один приклад:

```
f4(x,n):=block([sum,k], sum:1, for k:1 thru n do (sum:sum+k/x),sum)$
```

Час обчислення виразу `f4(5,1000000)` склав 11,973 с, час обчислення виразу `f4(5.0,1000000)` склав 11,384 с. Після трансляції `f4` час обчислення виразу `f4(5,1000000)` склав 3,578 с, а для `f4(5.0,1000000)` – 1,121 с (виграш за часом за рахунок виконання обчислень із рухомою крапкою приблизно у 10 разів).

4.2 Введення-виведення у пакеті Maxima

У цьому розділі розглянуто конструкції, що надають змогу здійснювати обмін даними між **Maxima** і іншими програмами.

4.2.1 Введення-виведення даних у консолі

Основна функція для зчитування даних, що вводяться користувачем: `read(`$expr_1$`, ..., `$expr_n$`)`. Вирази, що вводяться, $expr_1$, $expr_2$, ... при уведенні інтерпретуються. Поля введення розділяються крапками з комами або знаком $. Аргументи функції `read` можуть включати підказку.

Приклад:

```
(%i1) a:42$
(%i2) a:read("Значення a = ",a,", введіть нову величину");
Значення a = 42, введіть нову величину (p+q)^3;
(%o2) (q+p)^3
(%i3) display(a);
```

$a = (q+p)^3$

```
(%o3) done
```

Аналогічна функція `readonly` здійснює тільки введення даних (без їхньої інтерпретації).

Приклад (порівняння використання функцій `read` і `readonly`):

```
(%i1) a:7$
(%i2) readonly("Введіть вираз:");
Введіть вираз: 2^a;
(%o2) 2^a
(%i3) read("Введіть вираз:");
Введіть вираз: 2^a;
(%o3) 128
```

Виведення на екран здійснюється функцією `display`. Синтаксис її виклику: `display(`$expr_1$`, `$expr_2$`, ...)`.

Вирази зі списку аргументів виводяться зліва праворуч (спочатку сам вираз, а потім після знака рівності – його значення).

Аналогічна функція `disp` (синтаксис виклику: `disp(`$expr_1$`, `$expr_2$`, ...)`) виводить на екран тільки значення виразу після його інтерпретації.

Функція `grind` здійснює виведення до консолі **Maxima** аналогічно `disp`, але у формі, зручній для введення із клавіатури.

```
(%i1) a:1$ b:2$ c:3$
(%i4) display(a,b,c);
```

$a = 1$

$b = 2$

$c = 3$
(%o4) *done*
(%i5) disp(a,b,c);
1
2
3
(%o5) *done*
(%i6) grind(a);
1
(%o6) *done*

Керування консольним введенням-виведенням здійснюється за допомогою встановлення прапорців `display2d`, `display_format_internal` тощо.

Виведення на екран довгих виразів частинами (одна частина над іншою) здійснюється функцією `dispterms` (синтаксис виклику: `dispterms(expr)`).

Крім того, для виведення результатів обчислень використовується функція `print`. Синтаксис виклику: `print` (*expr*$_1$`,...,` *expr*$_n$`)`. Вирази *expr*$_1$, ..., *expr*$_n$ інтерпретуються і виводяться послідовно у рядок (на відміну від виведення, виконаного функцією `display`). Функція `print` повертає значення останнього інтерпретованого виразу.

Приклад:

```
(%i1) a:1$ b:2$ c:(a^2+b^2)$
(%i4) rez:print("Приклад:",a,b,c);
Приклад: 1 2 5
(%o4) 5
(%i5) rez;
(%o5) 5
(%i6) display("Приклад:",a,b,c);
Приклад:=Приклад:
```
$a = 1$
$b = 2$
$c = 5$
(%o6) *done*

4.2.2 Файлові операції введення-виведення

4.2.2.1 Введення-виведення текстових даних

Збереження поточного стану робочої області **Maxima** здійснюється за допомогою функції `save`. Ця функція надає змогу зберегти у файлі окремі об'єкти із зазначеними назвами. Варіанти виклику `save`[1]:

`save(filename,` *name*$_1$`,` *name*$_2$`,` *name*$_3$`, ...)` – зберігає поточні значення змінних *name*$_1$, *name*$_2$, *name*$_3$, ... до файла `filename`. Аргументи повинні бути назвами змінних, функцій або інших об'єктів. Якщо назва не асоціюється з якоюсь величиною у пам'яті, вона ігнорується. Функція `save` повертає назву файла, до якого збережено задані об'єкти.

`save(filename, values, functions, labels, ...)` – зберігає всі значення змінних, функцій, міток тощо.

`save(filename, [m, n])` – зберігає всі значення міток введення-виведення у проміжку від `m` до `n` (`m`, `n` – цілі літерали).

`save(filename,` *name*$_1$ $=$ *expr*$_1$`, ...)` – надає змогу зберегти об'єкти **Maxima** із заміною імені *expr*$_1$ на назву *name*$_1$.

`save(filename, all)` – зберігає всі об'єкти, наявні у пам'яті. Глобальний прапорець `file_output_append` управляє режимом запису. Якщо `file_output_append = true`, результати виведення `save` додаються у кінець файла результатів. Інакше файл результату перезаписується. Незалежно від `file_output_append`, якщо файл результатів не існує, то він створюється.

Дані, збережені функцією `save`, можуть бути знову завантажені функцією `load` (див. нижче).

Варіанти запису за допомогою `save` можуть сполучатися один з одним (приклад: `save(filename, aa, bb, cc=42, functions, [11,17]`).

Завантаження попередньо збереженого функцією `save` файла здійснюється функцією `load(filename)`.

Аналогічний синтаксис і у функції `stringout`, що призначена для виведення до файла виразів **Maxima** у форматі, придатному для наступного зчитування **Maxima**.

Синтаксис виклику `stringout`:

`stringout(filename,` *expr*$_1$`,` *expr*$_2$`,` *expr*$_3$`, ...)`
`stringout(filename, [m, n])`
`stringout(filename, input)`
`stringout(filename, functions)`
`stringout(filename, values)`

[1] Не забудьте, що назва файла має бути рядком взятим у прямі лапки, наприклад: `save("foo.l all)$`, або обчислюватися у своє рядкове значення за допомогою двох одинарних лапок: `s:"foo.l"$ save('s, all)$`.

Функція `load(filename)` обчислює вирази у файлі `filename`, створюючи у такий спосіб змінні, функції, і інші об'єкти **Maxima**. Якщо об'єкт із деякою назвою уже є у **Maxima**, при виконанні `load` він буде заміщений зчитуваним. Щоб знайти файл, який завантажується, функція `load` використовує змінні `file_search`, `file_search_maxima` і `file_search_lisp` як довідники пошуку. Якщо файл, який завантажують, не вдалося знайти, виводить повідомлення про помилку.

Завантаження працює однаково добре для коду на **Lisp** і коду на макромові **Maxima**. Файли, створені функціями `save`, `translate_file`, `compile_file` містять код на **Lisp**, а створені за допомогою функції `stringout` містять код **Maxima**. Всі ці файли можуть однаково добре бути обробленими функцією `load`. `Load` використовує функцію `loadfile`, щоб завантажити файли **Lisp**, і `batchload`, щоб завантажити файли **Maxima**.

`Load` не розпізнає конструкції `:lisp` у файлах, що містять код мовою **Maxima**, а також глобальні змінні _, __, %, і %th, поки не будуть створені відповідні об'єкти у пам'яті.

Функція `loadfile(filename)` призначена для завантаження файлів, що містять код на **Lisp**, створених функціями `save`, `translate_file`, `compile_file`. Для задач кінцевого користувача зручніше функція `load`.

Протокол сеансу **Maxima** може записуватися за допомогою функції `writefile` (він записується у форматі виведення до консолі). З тією ж метою використовується функція `appendfile` (запис у кінець наявного файла). Завершення запису і закриття файлу протоколу здійснюється функцією `closefile`. Синтаксис виклику: `writefile(filename)`, `closefile(filename)`.

4.2.2.2 Введення-виведення командних файлів

Основна функція, призначена для введення і інтерпретації командних файлів – функція `batch(filename)`. Функція `batch` читає вирази **Maxima** з файла `filename` і виконує їх. Функція `batch` відшукує `filename` у списку `file_search_maxima`. Файл із назвою `filename` включає послідовність виразів **Maxima**, кожен з яких повинен закінчуватися символом ; або $. Спеціальна змінна % і функція %th звертаються до попередніх результатів у межах файла. Файл може включати конструкції `:lisp`. Пробіли, табуляції, символи кінця рядка у файлі ігноруються. Відповідний вхідний файл може бути створений редактором тексту або функцією `stringout`.

Функція `batch` зчитує усі вирази з файла `filename`, показує введення у консолі, обчислює відповідні вирази і показує виведення також у консолі. Мітки введення призначаються вхідним виразам, мітки виведення – результатам обчислень, функція `batch` інтерпретує кожен вхідний вираз, поки не буде досягнутий кінець файла. Якщо передбачається реакція користувача (введення із клавіатури), виконання `batch` припиняється до завершення введення. Для зупинки виконання `batch`-файла використовується `Ctrl-C`.

Функція `batchload(filename)` зчитує і інтерпретує вираз з командного файла, але не виводить на консоль вхідних і вихідних виразів. Мітки введення і виведення виразам, що зустрічаються у командному файлі, також не призначаються. Спеціальна змінна % і функція %th звертаються до попередніх діалогових міток, не маючи результатів у межах файла. Крім того, файл `filename` не може включати конструкції `:lisp`.

4.3 Вбудовані обчислювальні методи

4.3.1 Числові методи розв'язування рівнянь

4.3.1.1 Розв'язування рівнянь із одним невідомим

Для розв'язування рівняння з одним невідомим у пакеті **Maxima** передбачена функція `find_root`. Синтаксис виклику:

```
find_root(expr, x, a, b)
find_root(f, a, b)
```

Пошук кореня функції `f` або виразу `expr` щодо змінної `x` здійснюється в межах $a \leqslant x \leqslant b$.

Для пошуку коренів використовується метод ділення навпіл або, якщо досліджувана функція досить гладка, метод лінійної інтерполяції.

4.3.2 Розв'язування рівнянь методом Ньютона: пакет `newton1`

Основна функція пакета `newton1` призначена для розв'язання рівнянь методом Ньютона.

Синтаксис виклику: `newton(expr,`x`,`x_0`,eps)`

Ця функція повертає наближений розв'язок рівняння `expr = 0` методом Ньютона, розглядаючи `expr` як функцію однієї змінної x. Пошук починається з $x = x_0$ і виконується, поки не буде досягнута умова $|expr| < eps$. Функція `newton` допускає наявність невизначених змінних у виразі `expr`, при цьому виконання умови $|expr| < eps$, оцінюється як істинне або помилкове. Таким чином, немає потреби оцінювати `expr` тільки як число.

Для використання пакета слід завантажити його командою `load(newton1)`.

Приклади використання функції `newton`:

```
(%i1) load(newton1);
(%o1) /usr/share/maxima/5.35.1/share/numeric/newton1.mac
(%i2) newton(cos(u),u,1,1/100);
```

```
(%o2) 1.570675277161251
(%i3) ev(cos(u),u=%);
(%o3) 1.2104963335033529 10^-4
(%i4) assume(a>0);
(%o4) [a > 0]
(%i5) newton(x^2-a^2,x,a/2,a^2/100);
(%o5) 1.00030487804878 a
(%i6) ev(x^2-a^2,x=%);
(%o6) 6.098490481853958 10^-4 a^2
```

4.3.2.1 Розв'язування рівнянь із декількома невідомими: пакет `mnewton`

Потужна функція для розв'язування систем нелінійних рівнянь методом Ньютона входить до складу пакета `mnewton`. Перед використанням пакет необхідно завантажити:

```
(%i1) load("mnewton");
(%o1) /usr/share/maxima/5.35.1/share/contrib/mnewton.mac
```

Після завантаження пакета `mnewton` стають доступними основна функція – `mnewton` і ряд додаткових змінних для керування нею: `newtonepsilon` (точність пошуку, типова величина – $10^{-\frac{fpprec}{2}}$), `newtonmaxiter` (максимальна кількість ітерацій, типова величина – 50).

Синтаксис виклику: `mnewton(FuncList, VarList, GuessList)`, де `FuncList` – список функцій, що утворюють розв'язувану систему рівнянь, `VarList` – список назв змінних, і `GuessList` – список початкових наближень.

Розв'язок повертається у тому самому форматі, що використовує функція `solve()`. Якщо розв'язку не знайдене, повертається порожній список.

Приклад використання функції `mnewton`:

```
(%i1) load("mnewton")$
(%i2) mnewton([x1+3*log(x1)-x2^2,2*x1^2-x1*x2-5*x1+1], [x1, x2], [5,5]);
(%o2) [[x1 = 3.756834008012769, x2 = 2.779849592817897]]
(%i3) mnewton([2*a^a-5],[a],[1]);
(%o3) [[a = 1.70927556786144]]
```

Як видно із другого прикладу, функція `mnewton` може використовуватися і для розв'язування одиничних рівнянь.

4.3.3 Інтерполяція

Для інтерполяції функцій, заданих таблично, у складі **Maxima** передбачений пакет розширення `interpol`, що надає змогу виконувати лінійну або поліноміальну інтерполяцію.

Пакет включає службову функцію `charfun2(x,a,b)`, що повертає `true`, якщо число `x` належить інтервалу `[a, b)`, і `false` у протилежному випадку.

4.3.3.1 Лінійна інтерполяція

Лінійна інтерполяція виконується функцією `linearinterpol` Синтаксис виклику: `linearinterpol(points)` або `linearinterpol (points, option)`.

Аргумент `points` має бути представлено в одній з таких форм:

- матриця із двома стовпцями, наприклад `p:matrix([2,4], [5,6], [9,3])`, при цьому перше значення пари або перший стовпець матриці – це значення незалежної змінної,
- список пар значень, наприклад `p:[[2,4], [5,6], [9,3]]`,
- список чисел, які розглядаються як ординати інтерпольованої функції, наприклад `p:[4,6,3]`, у цьому випадку абсциси призначаються автоматично (приймають значення 1, 2, 3 тощо).

Як параметр вказується назва незалежної змінної, щодо якої будується інтерполяційна функція.

Приклади виконання лінійної інтерполяції:

```
(%i1) load("interpol")$
(%i2) p:matrix([7,2], [8,2], [1,5], [3,2], [6,7])$
(%i3) linearinterpol(p);
(%o3) (13/2 - 3x/2) charfun2(x, -inf, 3) + 2 charfun2(x, 7, inf) + (37 - 5x) charfun2(x, 6, 7) + (5x/3 - 3) charfun2(x, 3, 6)
(%i4) f(x):=''%;
(%o4) f(x) := (13/2 - 3x/2) charfun2(x, -inf, 3) + 2 charfun2(x, 7, inf) + (37 - 5x) charfun2(x, 6, 7) + (5x/3 - 3) charfun2(x, 3, 6)
(%i5) map(f,[7.3,25/7,%pi]);
(%o5) [2, 62/21, 5%pi/3 - 3]
```

4.3.3.2 Інтерполяція поліномами Лагранжа

Інтерполяція поліномами Лагранжа виконується за допомогою функції `lagrange`.

Синтаксис виклику: `lagrange(points)` або `lagrange(points, option)`. Зміст параметрів `points` і `options` аналогічний зазначеному вище.

Приклад використання інтерполяції поліномами Лагранжа:

```
(%i1) load("interpol")$
(%i2) p:[[7,2], [8,2], [1,5], [3,2], [6,7]]$
(%i3) lagrange(p);
```

(%o3) $\frac{(x-7)(x-6)(x-3)(x-1)}{35} - \frac{(x-8)(x-6)(x-3)(x-1)}{12} + \frac{7(x-8)(x-7)(x-3)(x-1)}{30} - \frac{(x-8)(x-7)(x-6)(x-1)}{60} + \frac{(x-8)(x-7)(x-6)(x-3)}{84}$

```
(%i4) f(x):=''%;
```

(%o4) $f(x) := \frac{(x-7)(x-6)(x-3)(x-1)}{35} - \frac{(x-8)(x-6)(x-3)(x-1)}{12} + \frac{7(x-8)(x-7)(x-3)(x-1)}{30} - \frac{(x-8)(x-7)(x-6)(x-1)}{60} + \frac{(x-8)(x-7)(x-6)(x-3)}{84}$

```
(%i5) map(f,[2.3,5/7,%pi]);
```

(%o5) $[-1.567535, \frac{919062}{84035}, \frac{(\pi-7)(\pi-6)(\pi-3)(\pi-1)}{35} - \frac{(\pi-8)(\pi-6)(\pi-3)(\pi-1)}{12} + \frac{7(\pi-8)(\pi-7)(\pi-3)(\pi-1)}{30} - \frac{(\pi-8)(\pi-7)(\pi-6)(\pi-1)}{60} + \frac{(\pi-8)(\pi-7)(\pi-6)(\pi-3)}{84}]$

```
(%i6) %,numer;
(%o6) [-1.567535, 10.9366573451538, 2.893196551256924]
```

4.3.3.3 Інтерполяція сплайнами

Інтерполяція кубічними сплайнами виконується за допомогою функції `cspline`.

Синтаксис виклику: `cspline(points)` або `cspline(points, option)`. Зміст параметрів `points` і `options` аналогічний зазначеному вище.

Приклад використання інтерполяції кубічними сплайнами:

```
(%i1) load("interpol")$
(%i2) p:[[7,2], [8,2], [1,5], [3,2], [6,7]]$
(%i3) cspline(p);
```

(%o3) $\left(\frac{8283}{1096} - \frac{6091\cdot x}{3288} - \frac{1159\cdot x^2}{1096} + \frac{1159\cdot x^3}{3288}\right)\cdot\text{charfun2}\left(x,-\infty\ ,3\right) + \left(\frac{108928}{137} - \frac{494117\cdot x}{1644} + \frac{5174\cdot x^2}{137} - \frac{2587\cdot x^3}{1644}\right)\cdot\text{charfun2}\left(x,7,\infty\right) + \left(-\frac{199575}{274} + \frac{579277\cdot x}{1644} - \frac{15209\cdot x^2}{274} + \frac{4715\cdot x^3}{1644}\right)\cdot\text{charfun2}\left(x,6,7\right) + \left(\frac{9609}{274} - \frac{48275\cdot x}{1644} + \frac{2223\cdot x^2}{274} - \frac{3287\cdot x^3}{4932}\right)\cdot\text{charfun2}\left(x,3,6\right)$

```
(%i4) f(x):=''%;
```

(%o4) $f(x) := \left(\frac{8283}{1096} - \frac{6091x}{3288} - \frac{1159x^2}{1096} + \frac{1159x^3}{3288}\right)\text{charfun2}\left(x,-\infty\ ,3\right) + \left(\frac{108928}{137} - \frac{494117x}{1644} + \frac{5174x^2}{137} - \frac{2587x^3}{1644}\right)\text{charfun2}\left(x,7,\infty\right) + \left(-\frac{199575}{274} + \frac{579277x}{1644} - \frac{15209x^2}{274} + \frac{4715x^3}{1644}\right)\text{charfun2}\left(x,6,7\right) + \left(\frac{9609}{274} - \frac{48275x}{1644} + \frac{2223x^2}{274} - \frac{3287x^3}{4932}\right)\text{charfun2}\left(x,3,6\right)$

```
(%i5) map(f,[2.3,5/7,%pi]);
```

(%o5) $[1.991460766423356, \frac{273638}{46991}, -\frac{3287\cdot\pi^3}{4932} + \frac{2223\cdot\pi^2}{274} - \frac{48275\cdot\pi}{1644} + \frac{9609}{274}]$

```
(%i6) %,numer;
(%o6) [1.991460766423356, 5.823200187269903, 2.227405312429507]
```

4.3.4 Оптимізація з використанням пакета `lbfgs`

Основна функція пакета (`lbfgs(FOM, X, X0, epsilon, iprint)`) надає змогу знайти наближений розв'язок задачі мінімізації без обмежень цільової функції, обумовленої виразом `FOM`, за списком змінних `X` з початковим наближенням `X0`. Критерій закінчення пошуку визначається градієнтом норми цільової функції (градієнт норми `FOM < epsilonmax(1, norm)`).

Ця функція використовує квазіньютонівський алгоритм із обмеженою пам'яттю (алгоритм `BFGS`). Цей метод називають методом з обмеженим використанням пам'яті, тому що замість повного обернення матриці Гессе (гессіана) використовується наближення з низьким рангом. Кожна ітерація алгоритму – лінійний (одномірний) пошук, тобто, пошук уздовж променя у просторі змінних `X` з напрямком пошуку, обчисленим на базі наближеного обернення матриці Гессе. У результаті успішного лінійного пошуку значення цільової функції (`FOM`) зменшується. Зазвичай (але не завжди) норма градієнта `FOM` також зменшується.

Параметр функції `iprint` надає змогу контролювати виведення повідомлень про поступ пошуку. Величина елемента `iprint[1]` керує частотою виведення (`iprint[1]<0` – повідомлення не виводяться; `iprint[1]=0` – повідомлення на перших і останніх ітераціях; `iprint[1]>0` – виведення повідомлень на кожній `iprint[1]` ітерації). Величина `iprint[2]` керує об'ємом виведеної інформації (якщо `iprint[2]=0`, виводиться лічильник ітерацій, число обчислень цільової функції, її величину, величину норми градієнта `FOM` і довжини кроку). Збільшення `iprint[2]` (цілої змінної, що приймає значення 0, 1, 2, 3) спричиняє збільшення кількості виведеної інформації.

Позначення стовпчиків виведеної інформації:

- `I` – число ітерацій, що збільшується після кожного лінійного пошуку;

- NFN – кількість обчислень цільової функції;
- FUNC – значення цільової функції наприкінці лінійного пошуку;
- GNORM – норма градієнта цільової функції наприкінці чергового лінійного пошуку;
- STEPLENGTH – довжина кроку (внутрішній параметр алгоритму пошуку).

Функція lbfgs реалізована розробниками на **Lisp** шляхом перекодування класичного алгоритму, спочатку написаного на Fortran, тому зберегла деякі архаїчні риси. Однак використовуваний алгоритм має високу надійність і добру швидкодію.

Розгляньмо приклади використання lbfgs.

Найпростіший приклад – мінімізація функції однієї змінної. Необхідно знайти локальний мінімум функції $f(x) = x^3 + 3x^2 - 2x + 1$. Результати розрахунків:

```
(%i1) load(lbfgs);
(%o1) /usr/share/maxima/5.35.1/share/lbfgs/lbfgs.mac
(%i2) FOM:x^3+3*x^2-2*x+1;
(%o2) x^3+3*x^2-2*x+1
(%i3) lbfgs(FOM,[x],[1.1], 1e-4, [1, 0]);
*************************************************
N=    1   NUMBER OF CORRECTIONS=25
INITIAL VALUES
F=  3.761000000000001D+00   GNORM=  8.230000000000001D+00
*************************************************
I  NFN     FUNC                    GNORM                   STEPLENGTH
1    2     8.309999999999997D-01   1.370000000000000D+00   1.215066828675577D-01
2    3     7.056026396574796D-01   3.670279947916664D-01   1.000000000000000D+00
3    4     6.967452517789576D-01   3.053950958095847D-02   1.000000000000000D+00
4    5     6.966851926112383D-01   5.802032710369720D-04   1.000000000000000D+00
5    6     6.966851708806983D-01   8.833119583551152D-07   1.000000000000000D+00
THE MINIMIZATION TERMINATED WITHOUT DETECTING ERRORS.
IFLAG = 0
(%o3) [x = 0.29099433470072]
```

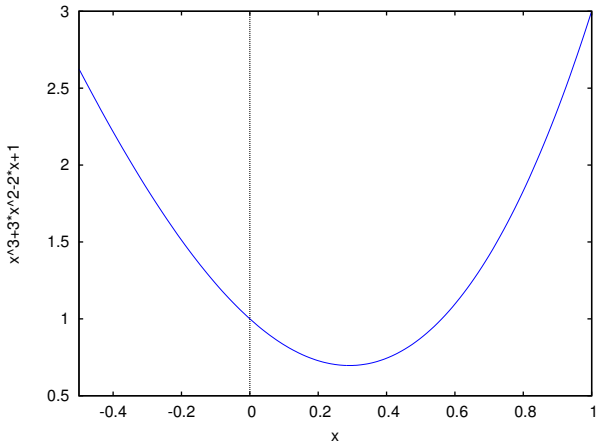


Рис. 4.1. Графік досліджуваної функції в околі мінімуму

Розгляньмо результати мінімізації функції декількох змінних за допомогою lbfgs:

```
(%i1) load (lbfgs)$
(%i2) FOM:2*x*y+8*y*z+12*x*z+1e6/(x*y*z);
```

(%o2) $8 \cdot y \cdot z + 12 \cdot x \cdot z + \dfrac{1000000.0}{x \cdot y \cdot z} + 2 \cdot x \cdot y$

```
(%i3) lbfgs(FOM, [x,y,z], [1,1,1] , 1e-4, [-1,0]);
```

(%o3) $[x = 13.47613086835734,\ \ y = 20.21398622934409,\ \ z = 3.369022781547174]$

4.3.4.1 Оптимізація з обмеженнями методом невизначених множників Лагранжа

Для розв'язання задач мінімізації з обмеженнями в складі **Maxima** передбачений пакет augmented_lagrangian_method, що реалізує метод невизначених множників Лагранжа.

Синтаксис виклику функції:

`augmented_lagrangian_method(FOM, xx, C, yy);`
`augmented_lagrangian_method(FOM, xx, C, yy, optional_args)`.

Розглянута функція повертає наближений розв'язок задачі мінімізації функції декількох змінних з обмеженнями, представленими у вигляді рівностей. Цільова функція задається виразом `FOM`, варійовані змінні – списком `xx`, їхні початкові значення – списком `yy`, обмеження – списком `C` (передбачається, що обмеження прирівнюються до 0). Змінні `optional_args` задаються у формі `символ=значення`.

Розпізнаються наступні символи:

`niter` – число ітерацій методу невизначених множників Лагранжа;

`lbfgs_tolerance` – точність пошуку LBFGS;

`iprint` – той же параметр, що і для `lbfgs`;

`%lambda` – початкове значення невизначеного множника для методу Лагранжа.

Для використання функції `augmented_lagrangian_method` необхідно завантажити її командою `load(augmented_lagrangian)`.

Дана реалізація методу невизначених множників Лагранжа базується на використанні квазіньютонівського методу LBFGS.

4.3.5 Обчислювальне інтегрування: пакет `romberg`

Для обчислення визначених інтегралів обчислювальними методами у **Maxima** є проста у використанні і достатньо потужна функція `romberg` (перед використанням її необхідно завантажити).

Синтаксис виклику:

`romberg(expr, x, a, b)`
`romberg(F, a, b)`

Функція `romberg` обчислює визначені інтеграли методом Ромберґа. У формі `romberg(expr, x, a, b)` повертає оцінку повного інтеграла вираз `expr` за змінною `x` у межах від `a` до `b`. Вираження `expr` повинне повертати дійсне значення (число із рухомою крапкою).

У формі `romberg(F, a, b)` функція повертає оцінку інтеграла функції `F(x)` за змінною `x` у межах від `a` до `b` (`x` являє собою неназваний, єдиний аргумент `F`; фактичний аргумент може бути відмінним від `x`). Функція `F` повинна бути функцією **Maxima** або **Lisp**, що повертає значення із рухомою крапкою.

Точністю обчислень при виконанні `romberg` керують глобальні змінні `rombergabs` і `rombergtol`. Функція `romberg` завершує роботу успішно, коли абсолютне розходження між послідовними наближеннями – менше ніж `rombergabs`, або відносне розходження у послідовних наближеннях – менше ніж `rombergtol`. Таким чином, коли `rombergabs` дорівнює 0.0 (це типове значення), тільки величина відносної помилки впливає на виконання функції `romberg`.

Функція `romberg` зменшує крок інтегрування вдвічі щонайменше `rombergit` раз, тому максимальна кількість обчислень підінтегральної функції становить $2^{\text{rombergit}}$. Якщо критерій точності інтегрування, установлений `rombergabs` і `rombergtol`, не виконано, `romberg` виводить повідомлення про помилку. Функція `romberg` завжди робить принаймні `rombergmin` ітерації; це – евристичне правило, призначене, щоб запобігти передчасному завершенню виконання функції, коли підінтегральний вираз є коливальним.

Обчислення за допомогою `romberg` багатомірних інтегралів можливе, але закладений розробниками спосіб оцінки точності призводить до того, що методи, розроблені спеціально для багатомірних задач, можуть призвести до тієї ж самої точності з істотно меншою кількістю оцінок функції.

Розгляньмо приклади обчислення інтегралів з використанням `romberg`:

```
(%i1) load (romberg);
(%o1) /usr/share/maxima/5.35.1/share/numeric/romberg.lisp
(%i2) g(x,y):=x*y/(x+y);
```

(%o2) $\mathrm{g}\left(x,y\right) := \dfrac{x\cdot y}{y+x}$

```
(%i3) estimate:romberg(romberg(g(x, y), y, 0, x/2), x, 1, 3);
(%o3) 0.8193022864324522
(%i4) assume(x>0);
```

(%o4) $[x > 0]$

```
(%i5) integrate(integrate(g(x, y), y, 0, x/2), x, 1, 3);
```

(%o5) $-9\cdot\log\left(\frac{9}{2}\right)+9\cdot\ \log\left(3\right)+\dfrac{2\cdot\log\left(\dfrac{3}{2}\right)-1}{6}+\dfrac{9}{2}$

```
(%i6) float(%);
(%o6) 0.8193023963959085
```

Як видно з отриманих результатів обчислення подвійного інтеграла, точний і наближений розв'язок збігаються до 7 знака включно.

Розділ 5

Обрамлення Maxima

5.1 Класичні графічні інтерфейси Maxima

5.1.1 Графічний інтерфейс wxMaxima

Для зручності роботи відразу звернемося до графічного інтерфейсу **wxMaxima**, тому що він є найбільш дружнім для користувачів-початківців системи.

Перевагами **wxMaxima** є:

- можливість графічного виведення формул;
- спрощене введення найчастіше використовуваних функцій (через діалогові вікна);
- можливість включення графічних ілюстрацій безпосередньо у текст робочої книги (при використанні формату **wxMaxima**)

5.1.1.1 Робоче вікно wxMaxima

Розгляньмо робоче вікно програми (див. рис. 5.1 і 5.2). Згори вниз розташовуються: текстове меню програми – доступ до основних функцій і налаштувань програми. У текстовому меню **wxMaxima** розташовано пункти функцій для розв'язання великої кількості типових математичних задач, розділені за групами: рівняння, алгебра, аналіз, спростити, графіки, обчислювальні обчислення. Введення команд через діалогові вікна спрощує роботу із програмою для початківців.

Наприклад, пункт меню «Аналіз/Інтегрувати» надає змогу обчислити визначений або невизначений інтеграл. Після введення необхідних параметрів, у робочому вікні ми побачимо команду і результат обчислення:

```
(%i1) integrate(3*x+5/x,x);
```

(%o1) $5\log(x)+\frac{3x^2}{2}$

Приклад використання команд меню для обчислення границі

$$\lim_{x\to 0}\frac{\sin 2x}{x}$$

представлений на рисунках 5.3 і 5.4. Слід зазначити, що оболонка **wxMaxima** при виклику команди і відповідного діалогового вікна генерує текстову команду, інтерпретацію якої здійснює обчислювальне ядро **Maxima**. Переданий ядру **Maxima** рядок виводиться до командного вікно аналогічно команді, введеній вручну. Після генерації і першого виконання команди (або набору команд) можна доповнювати і редагувати автоматично створену команду, розглядаючи її як шаблон.

Нижче розташовується графічне меню основних команд із піктограмами, що відповідають найбільше часто використовуваним функціям для роботи з файлами: відкрити / зберегти / надрукувати дані, а також функціям редагування – копіювати / вилучити / вставити текст та інших.

Центральну частину робочого вікна **wxMaxima** займає командне вікно (псевдотермінал), у яке вводяться команди системи і виводяться результати.

В останніх версіях інтерфейсного пакета **wxMaxima** використовується концепція комірок (cells) у робочій книзі. Комірка включає або набір команд **Maxima**, або результати їхнього виконання (зокрема графіки). Крім того, за аналогією з **Maple** і **Mathematica wxMaxima** підтримує текстові комірки (text cells) для пояснень і коментарів, а також комірки для заголовків і номерів розділів (title cells, section cells, subsection cells). Приклад книги **Maxima** з комірками зазначених типів представлений на рис. 5.5. Допускається вставка зображень у робочу книгу (також у спеціальні комірки).

При збереженні книги (у форматі `wxm`) у файл виводяться тільки вхідні комірки (`input`). Тому при роботі зі збереженим документом не обов'язково інтерпретувати всі комірки, хоча це можливо за допомогою пункту «Обчислити усі комірки» з меню «Комірка»).

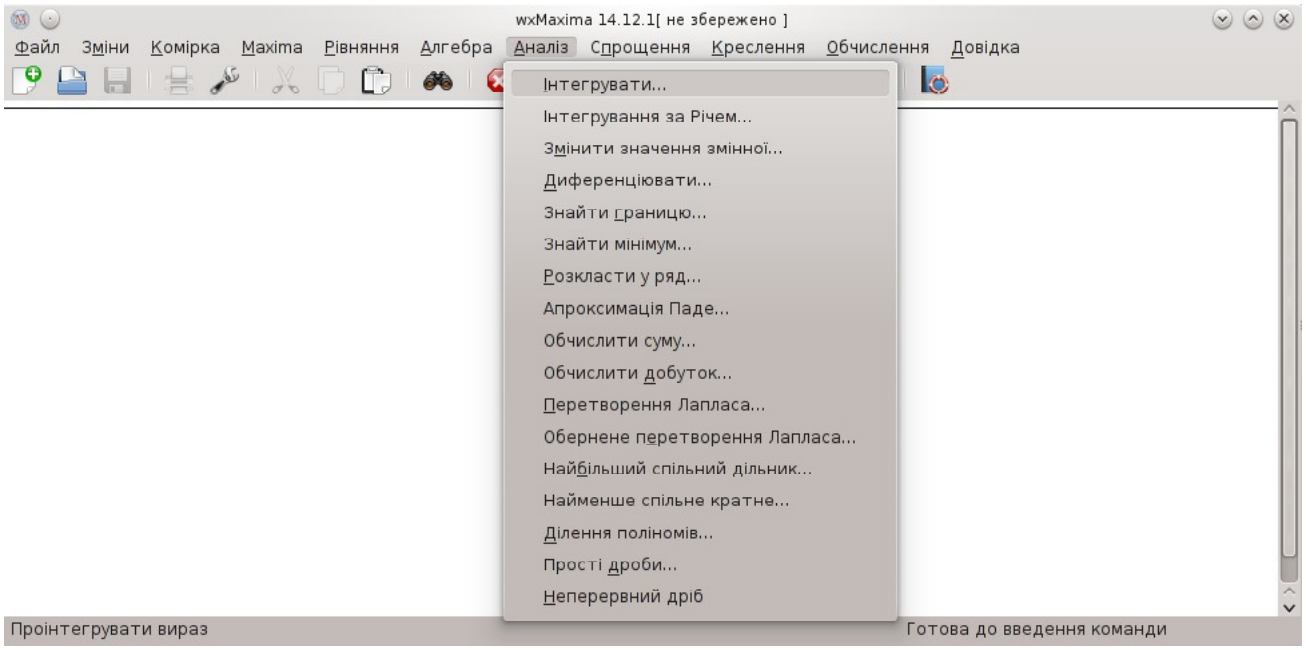


Рис. 5.1. Інтерфейс **wxMaxima**, вибір команди інтегрування.

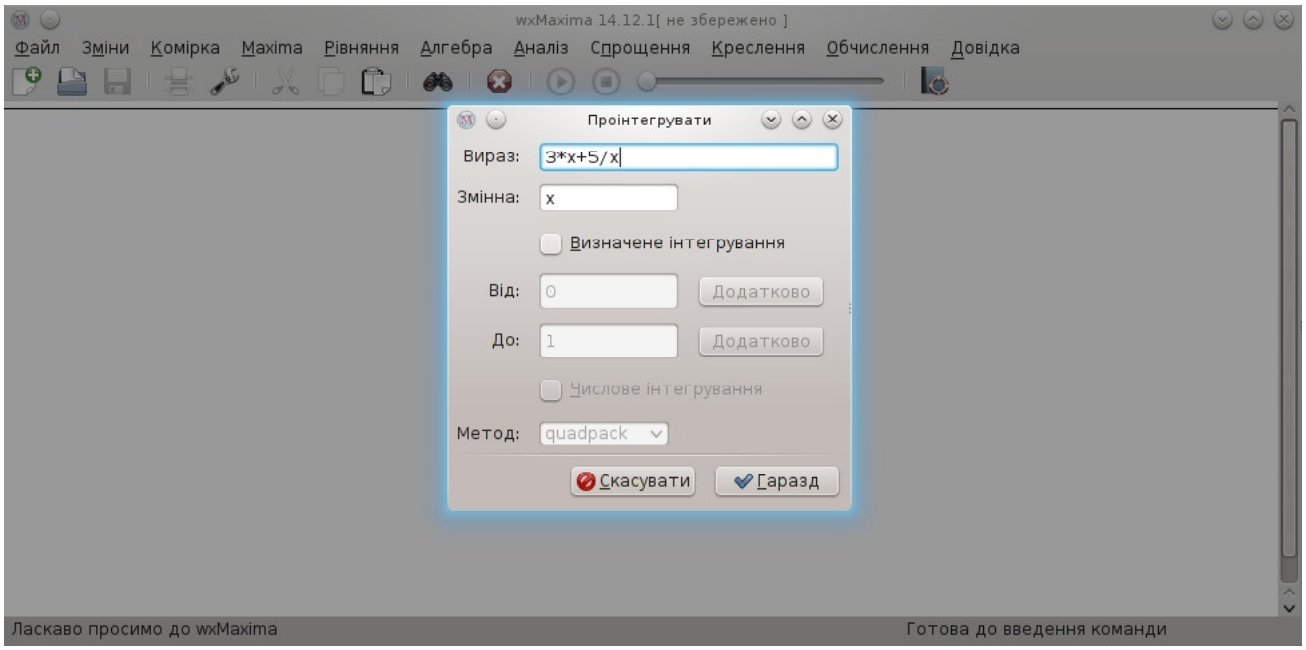


Рис. 5.2. Інтерфейс **wxMaxima**, обчислення інтеграла.

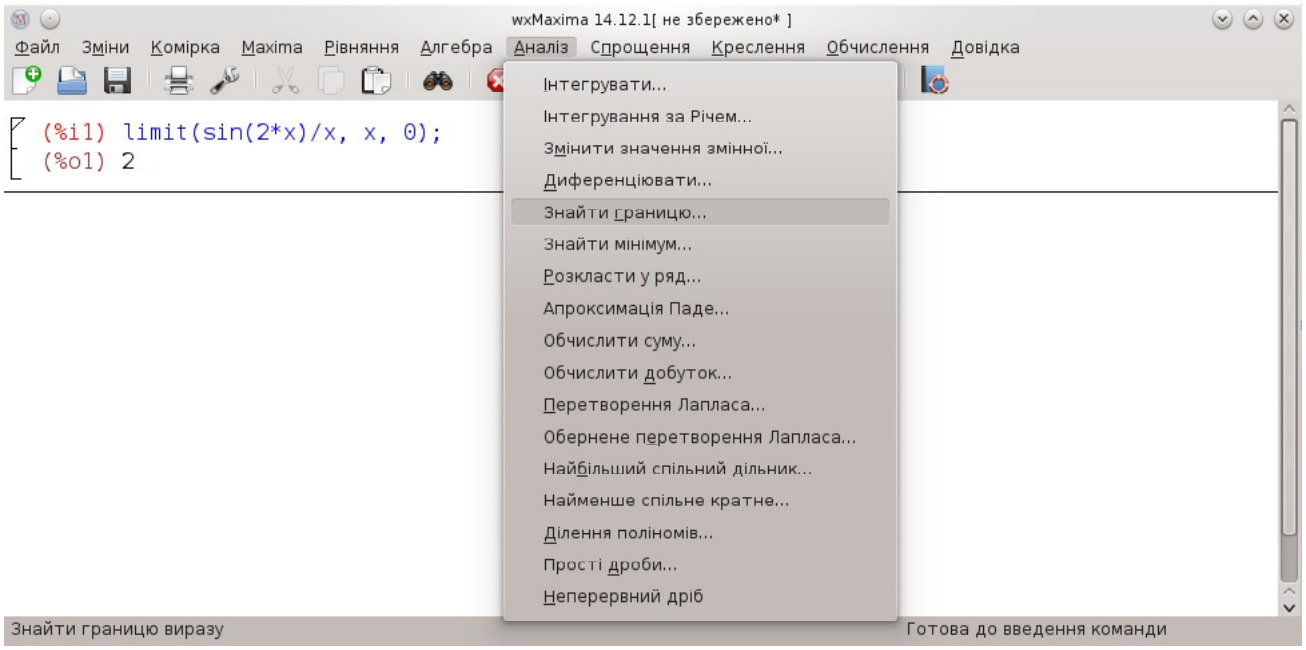


Рис. 5.3. Інтерфейс **wxMaxima**, вибір команди «Знайти границю».

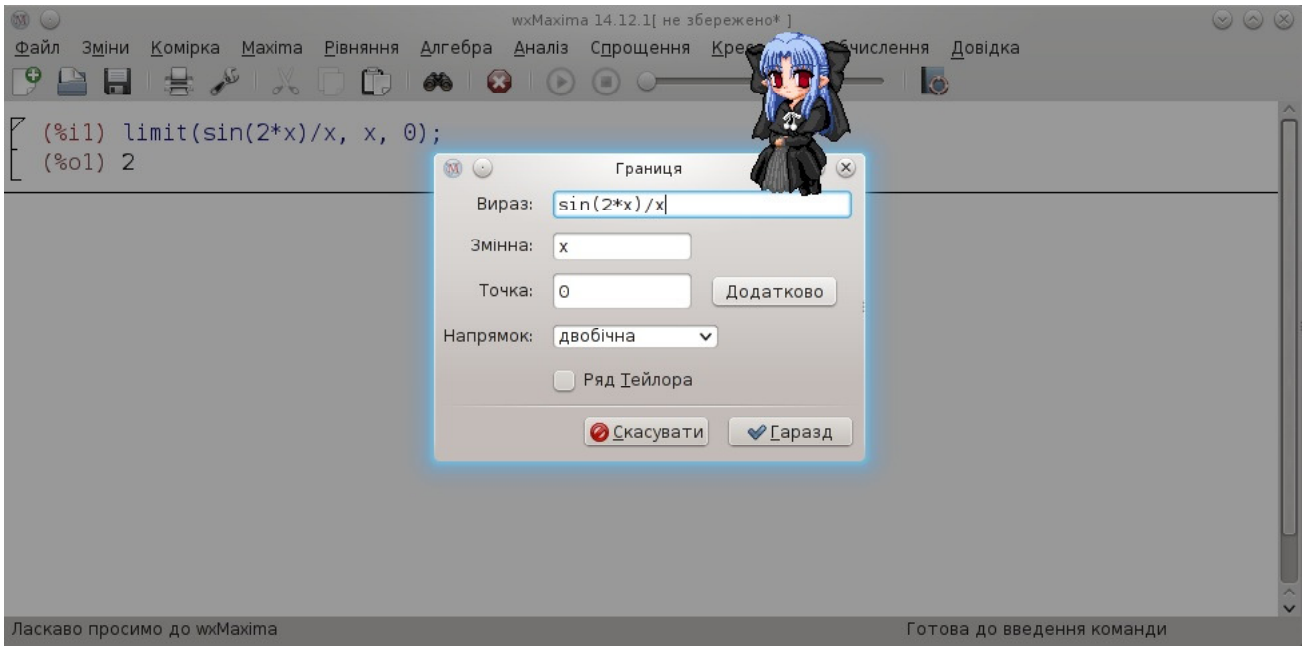


Рис. 5.4. Інтерфейс **wxMaxima**, вікно введення – обчислення границі.

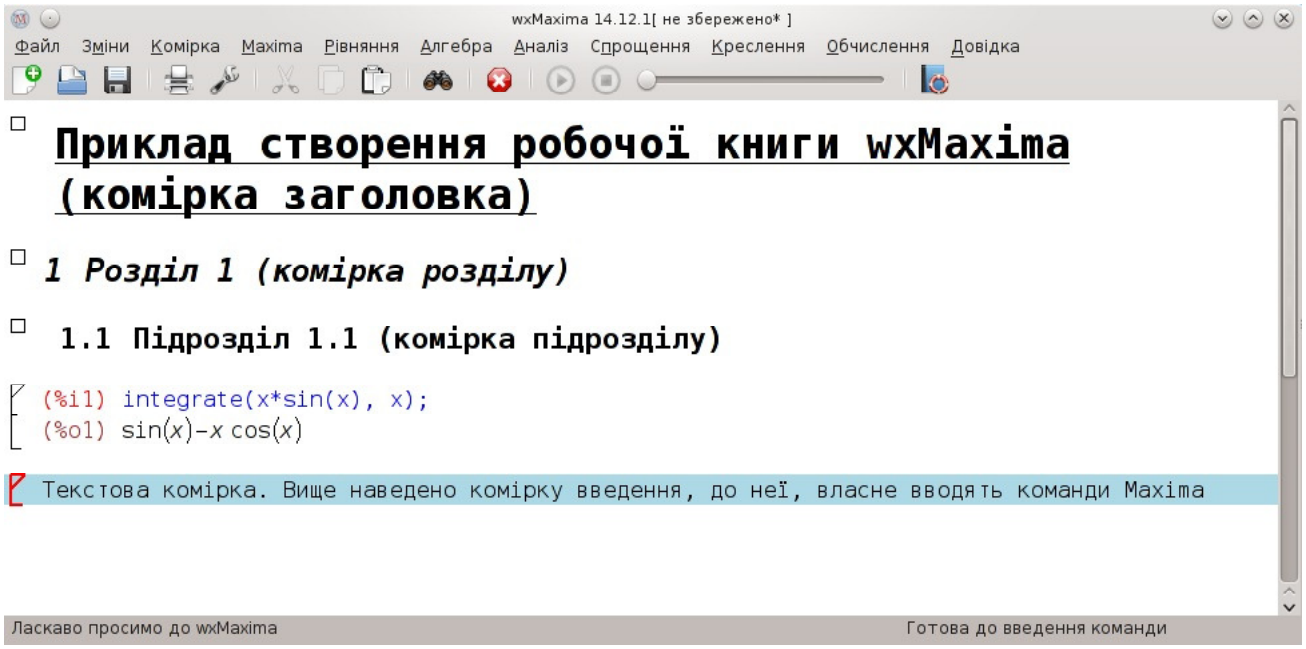


Рис. 5.5. Приклад вставлення комірок різних типів до книги **wxMaxima**.

Робочу книгу **Maxima** можна експортувати у формати `html` або `pdflatex`.

Інтерпретація поточного осередку, у якій може бути кілька команд, здійснюється після натискання комбінації клавіш `Ctrl+Enter`, або командою меню «Комірка». Якщо необхідно запобігти виведенню відгуку команди, варто явно завершити її символом $. Сучасні версії **wxMaxima** автоматично завершують введення, якщо це необхідно, символом ";".

При використанні інтерфейсу **wxMaxima** можна позначити у командному вікні необхідну формулу і викликавши контекстне меню правою кнопкою миші: скопіювати будь-яку формулу у текстовому вигляді, у форматі TeX або у вигляді графічного зображення, для наступного вставлення до якогось документа. Приклад контекстних меню при роботі з **wxMaxima** дивися на рисунках 5.6, 5.7 і 5.8.

Також, у контекстному меню, при виборі результату обчислення, пропонуються ряд операцій з позначеним виразом (наприклад, спрощення, розкриття дужок, інтегрування, диференціювання тощо).

Типово, **wxMaxima** припускає, що команда, що вводиться за допомогою кнопки, застосовується до останнього виведення (тобто аргумент команди – %). Всі кнопки або пункти меню у верхній або нижній частині робочого вікна відповідають тій або іншій команді **Maxima**.

Крім того, **wxMaxima** надає зручний інтерфейс до документації з системи **Maxima**.

Меню «Зміни → Налаштувати» забезпечує досить широкі можливості налаштовування графічного інтерфейсу **wxMaxima**. Передбачено три групи параметрів:

- параметри, що визначають окремі особливості виконання команд;
- параметри виклику обчислювального ядра **Maxima**;
- параметри, що визначають стиль графічного інтерфейсу (мова, шрифти, гама кольорів тощо).

Керування процесом обчислень здійснюється командами пункту головного меню «Maxima». Користувачеві надаються наступні можливості:

- перервати обчислення, запустити знову **Maxima**, очистити пам'ять;
- переглянути вміст пам'яті (змінні, функції, визначення тощо);
- змінити формат перегляду результатів.

5.1.2 Графічний інтерфейс xMaxima

Інтерфейс xMaxima фактично є специфічним типом браузера, оскільки даний інтерфейс передбачає обмін даними з обчислювальним ядром **Maxima** через сокет. Інтерфейс відзначається простотою (точніше, мінімалізмом). В останніх версіях **xMaxima** після запуску відкриваються одночасно вікно браузера системи довідки і консоль команд.

Вважається, що користувач володіє командами **Maxima** і макромовою програмування. Загальний вид командного вікна **xMaxima** представлено на рис. 5.9. Пункти меню File, Edit, Options надають змогу керувати сеансом **Maxima**, зберігати і запускати пакетні файли. У робочу книгу **xMaxima** можна вбудовувати графіки у форматі openmath (залежно від встановлення параметра plot window). Приклад робочого вікна xMaxima із простими графіками представлений на рис. 5.10. Графік у робочій книзі можна обертати, редагувати, зберігати до файла. Як і **wxMaxima**, інтерфейс **xMaxima** надає доступ до `html`-файла довідки з пакета **Maxima**.

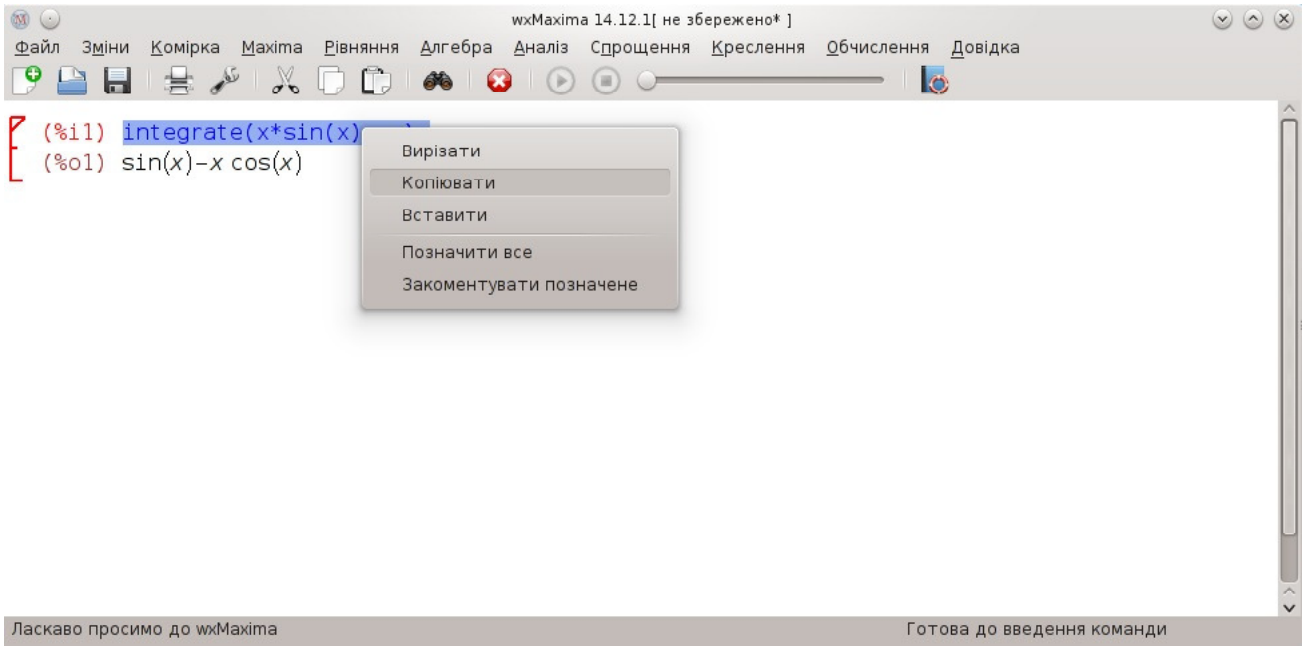


Рис. 5.6. Інтерфейс **wxMaxima**. Контекстне меню рядка введення.

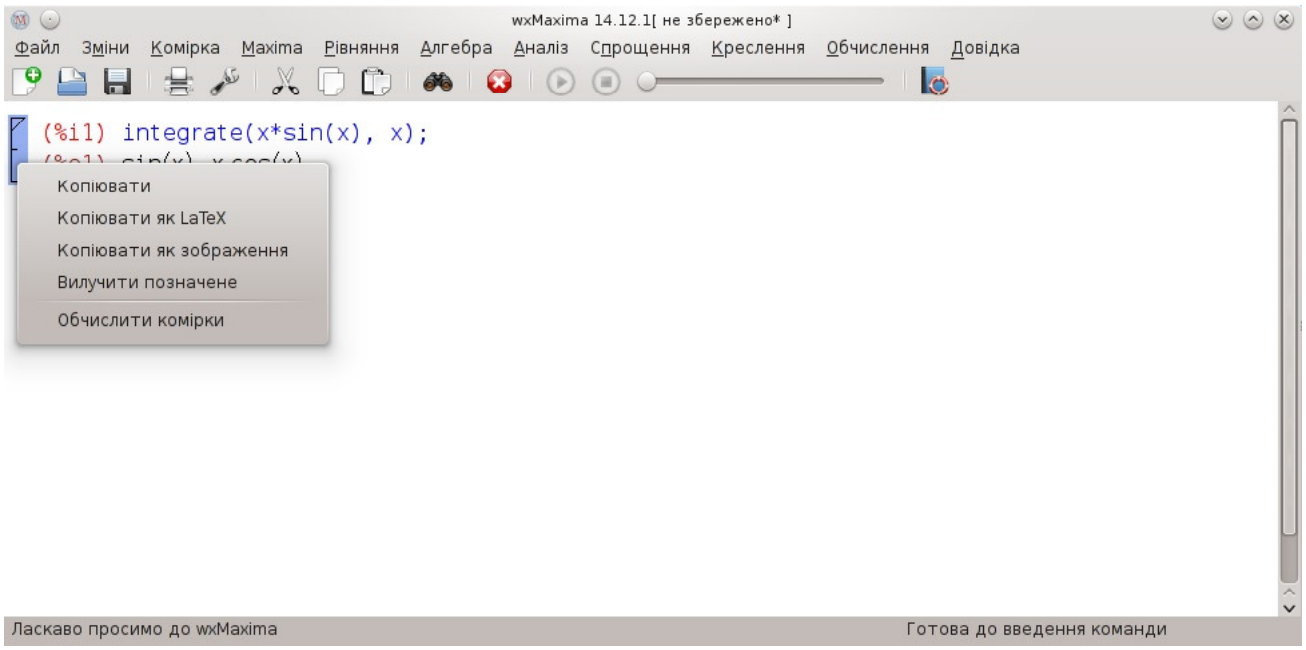


Рис. 5.7. Інтерфейс **wxMaxima**. Контекстне меню комірки.

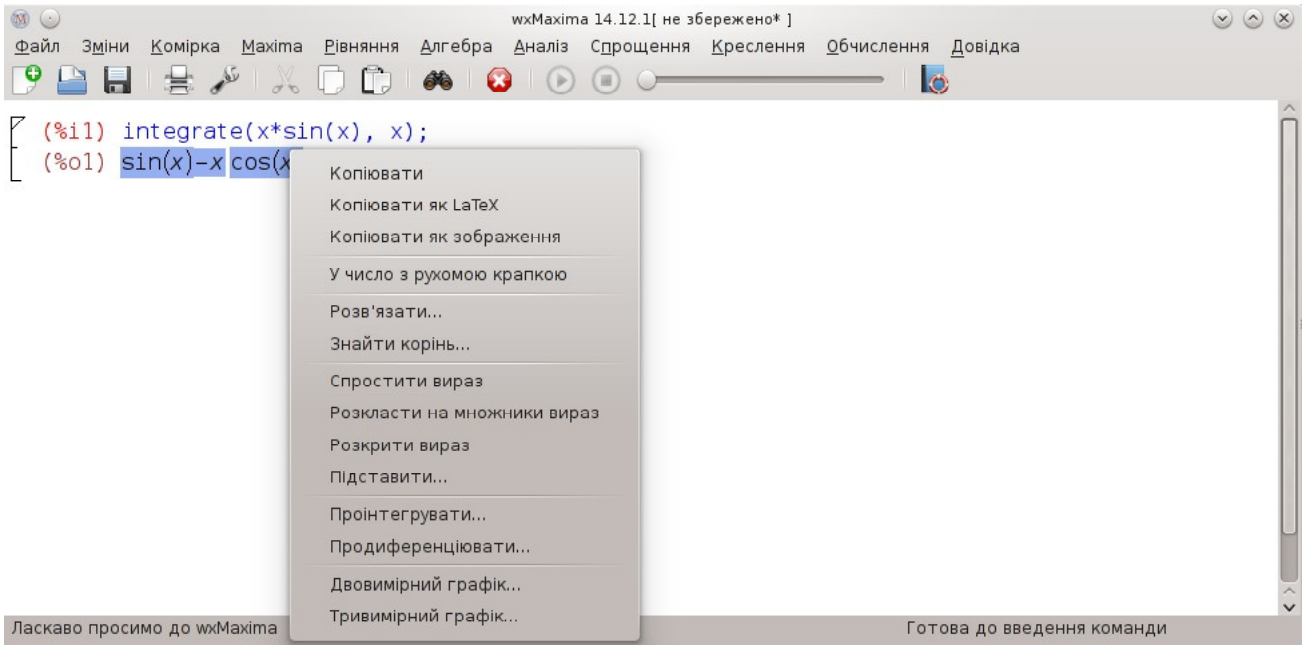


Рис. 5.8. Інтерфейс **wxMaxima**. Контекстне меню рядка виведення.

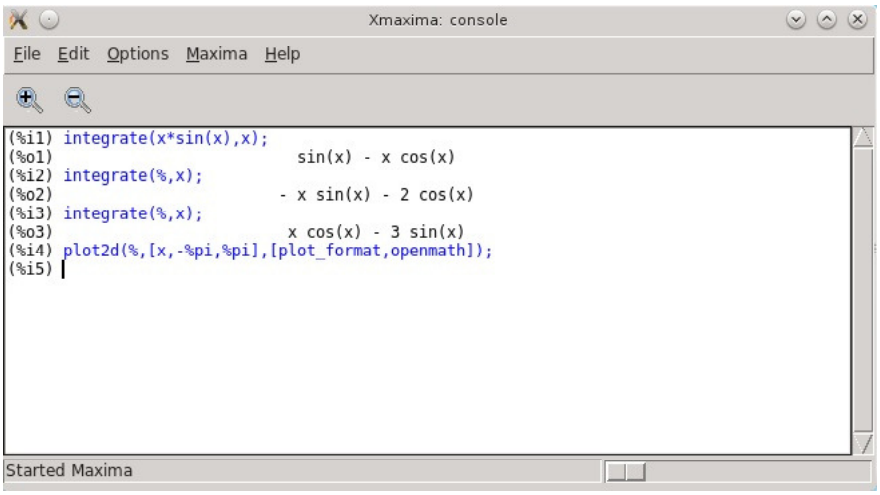


Рис. 5.9. Загальний вигляд робочого вікна xMaxima.

5.1.3 Використання редактора TeXmacs як інтерфейсу Maxima

Широкі можливості роботи у **Maxima** і інших математичних пакетах надає редактор **TeXmacs**. Розробник позиціонує його як LaTeX-редактор, однак це не зовсім так. **TeXmacs** використовує власний внутрішній формат, але надає змогу експортувати документи у LaTeX (при цьому отриманий TeX-файл дуже схожий на результат експортування до `.tex` документа LibreOffice).

У **TexMacs** реалізовано підхід до структури документа, багато в чому ідентичний LaTeX, а також можливості введення і редагування складних математичних формул. Недоліком редактора є невдалий вибір способу локалізації, що утрудняє відкриття документів **TeXmacs** за допомогою інших редакторів (**LibreOffice** тощо).

Важливою особливістю **TeXmacs** є можливість вбудовувати у текст документа сеансу роботи з різними математичними пакетами (зокрема і **Maxima**).

Можливість вбудовувати у текст документа графічні ілюстрації, а також можливість розщеплювати сеанс для введення пояснень і коментарів робить **TeXmacs** досить привабливим засобом для роботи з **Maxima**. У сучасних версіях **TeXmacs** при запуску сеансу **Maxima** у головному меню з'являється пункт **Maxima**, у якому передбачене спанде меню з переліком основних команд **Maxima**.

Остаточну версію TeXmacs-документів доцільно представляти у форматі pdf (цей редактор забезпечує пряме експортування до pdf). При збереженні документів у форматі **TeXmacs** і їхньому наступному редагуванні можливе і редагування полів введення сеансу **Maxima** із переобчисленням результатів.

5.1.4 Робота з Maxima з Emacs

Універсальний редактор **Emacs** також може використовуватися як оболонка до **Maxima**. Для цього передбачено кілька режимів: `maxima-mode`, `EMaxima` і `iMaxima`.

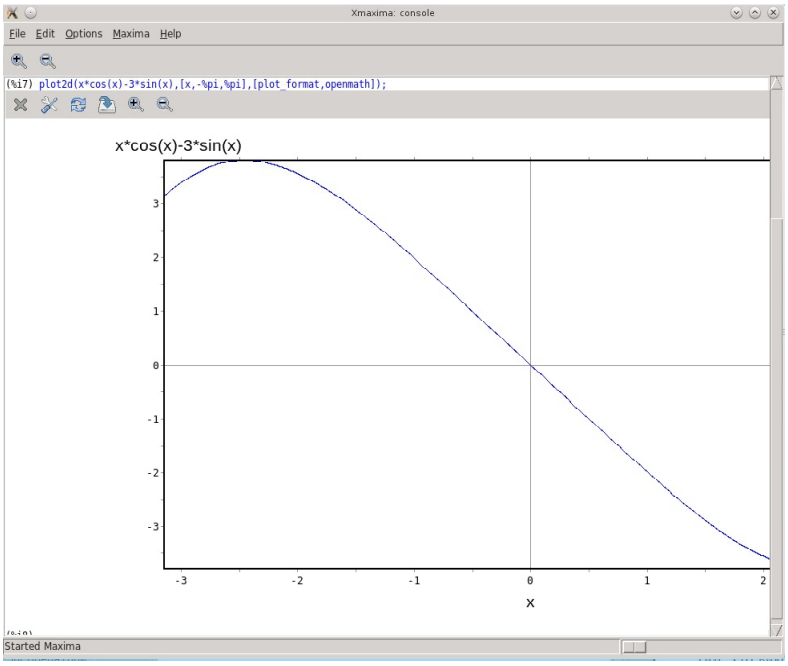


Рис. 5.10. Вбудований графік у робочій книзі xMaxima.

Основний режим роботи з **Maxima** в Emacs – `maxima-mode`. Цей режим запускається клавіатурною комбінацією `M-x-maxima-mode` (зазвичай натисканням `Alt-M-Alt-X` і після появи підказки – набиранням `maxima`). Цей режим трохи аскетичний (схожий на **xMaxima**), але досить зручний. Загальний вигляд робочого вікна для даного режиму представлений на рис. 5.12. На цьому ж рисунку видно меню навігації поточним сеансом, що надає змогу показувати необхідну ділянку сеансу, зберігати частину результатів до протоколу, повторювати введення вже використаних у поточному сеансі команд тощо.

Графіки до робочої книги, відкритої у **Emacs**, не вбудовуються. Збереження копії рисунка слід виконувати засобами gnuplot або openmath.

Інтерфейс `EMaxima` – скоріше не самостійний режим, а надбудова над режимом LaTeX, що напевне сподобається тим, хто використовує **Emacs** для редагування LaTeX-документів. На відміну від режиму **Maxima**, що призначений для звичайного ізольованого запуску повноцінного сеансу **Maxima**, тут йдеться про можливості вставляти окремі команди **Maxima** і, природно, результати їхніх обчислень, безпосередньо до документа LaTeX, що редагується. Запуск режиму здійснюється командою `EMaxima-mode` (`M-x emaxima`).

У найпростішому випадку з використанням `EMaxima` можна створити комірку **Maxima** комбінацією `C-c C-o` («open cell»), увести у ній будь-яку команду або набір команд **Maxima** у текстовій нотації і одержати результат обчислення цієї команди або у звичайному текстовому форматі натисканням `C-c C-u c`, або у LaTeX-форматі за допомогою `C-c C-u C` (тобто `Ctrl-C Ctrl-U Shift-C`). Тут «u e» походить від «update cell»; а суміжні команди, що породжують виведення у простій текстовій формі і у формі LaTeX, завжди прив'язані в `EMaxima` до однакових малої і великої літер відповідно.

Використовувати інтерфейс `EMaxima` зручно при створенні об'ємних документів у LaTeX математичного характеру, які припускають включення результатів символьних обчислень.

Останній **Emacs**-інтерфейс до **Maxima** – `iMaxima` – відрізняється від інших самостійним (а не за допомогою LaTeX-документа, як в `EMaxima`) графічним поданням математичних формул. Властиво, саме для цього він і створений, і його відмінність від `Maxima-mode` полягає саме у можливості графічного показу TeX-коду, створеного **Maxima**.

Цей режим можна налаштувати таким чином, щоб усередині нього запускався режим `Maxima` (тобто `Maxima-Emacs`), і користуватися всіма командами останнього і їхніми клавіатурними прив'язками. Тобто, фактично режим `iMaxima` у такому варіанті можна розглядати як графічний інтерфейс уже над `Maxima-Emacs`; саме це може додати додаткової привабливості останньому. На відміну від всіх розглянутих вище інтерфейсів, `iMaxima` – сторонній проект, що розробляється окремо. Для його встановлення слід додатково встановити пакет `breqn`, відповідальний за перенесення рядків у математичних формулах у форматі LaTeX. Інструкцію зі встановлення самої `iMaxima` і `breqn` можна знайти на сайті проекту.

5.2 Робота з Maxima у KDE: інтерфейс Cantor

Інтерфейс користувача **Cantor** складається із трьох частин:

- Панель вкладок, за допомогою якої можна перемикатися між документами;
- Панель довідки, де буде показаний опис команди, якщо ввести «? команда»;
- Панель поточного документа з меню команд, що нагадують інтерфейс **wxMaxima**.

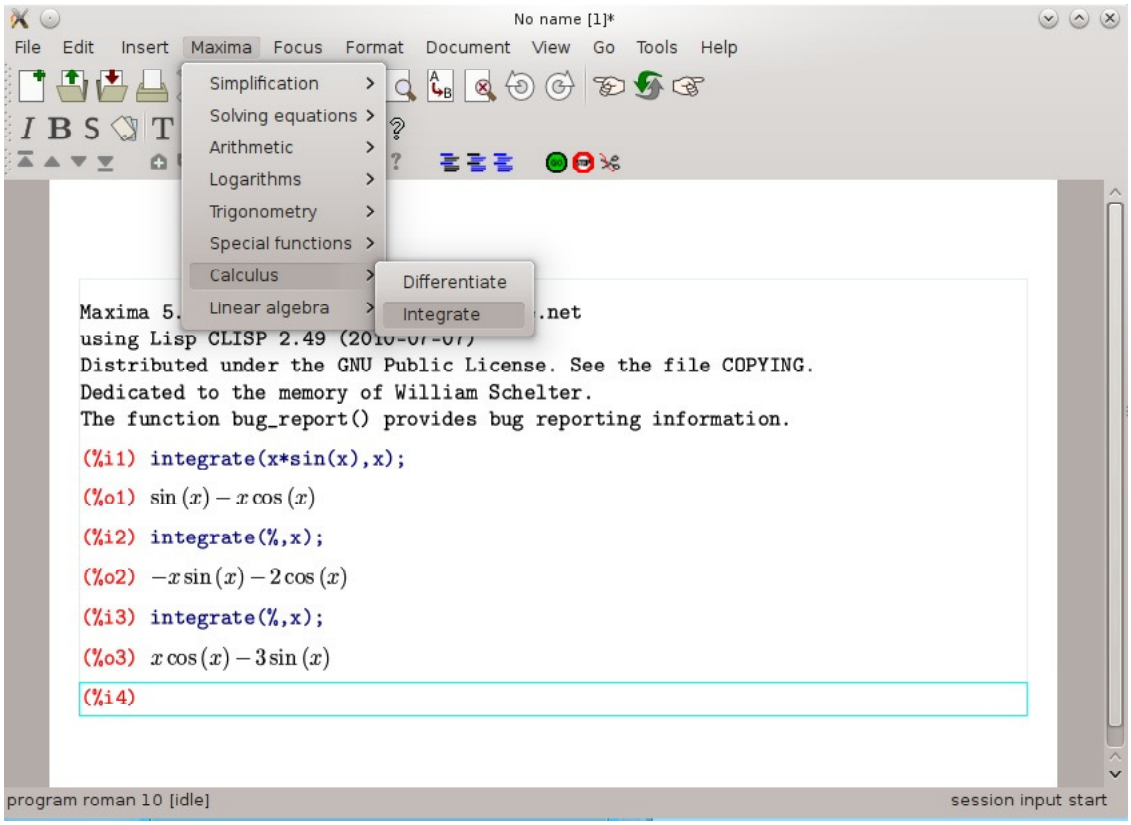


Рис. 5.11. Робоче вікно **TeXmacs** із запущеним сеансом **Maxima**.

Пакет **Cantor** розрахований на стільницю KDE, тому графічний інтерфейс написаний з використанням бібліотек Qt.

5.2.1 Документ Cantor

У **Cantor** ви працюєте з документом. У ньому можна вводити вираз, робити обчислення і бачити результати. Аналогічно **wxMaxima** або **Emacs**, документ **Cantor** включає комірки, що містять команди поточного пакета і результати їхнього виконання. Поряд з командами і результатами, можна вводити і коментарі.

Набір доступних у виразах команд залежить від використовуваної системи комп'ютерної алгебри, тому корисно знати синтаксис конкретної системи. Якщо ви знаєте назву команди, можна подивитися її опис, увівши «? команда». Щоб подивитися приклади документів **Cantor**, виберіть пункт меню «Файл → Отримати приклади робочих аркушів...» і завантажите документи, опубліковані іншими користувачами.

5.2.2 Налаштовування

За допомогою меню «Параметри» можна налаштувати зовнішній вигляд документа. Параметр `Увімкнути верстання LaTeX` впливає на те, у якому форматі буде показано результати обчислень. Якщо цей пункт позначено, результат буде оброблено за допомогою системи LaTeX для створення візуально привабливих формул. Наприклад, `3*x^2*sqrt(2)*x+2/3` перетвориться на $3x^2 \cdot \sqrt{2} \cdot x + \frac{2}{3}$.

Підсвічування синтаксичних конструкцій покращує придатність коду до читання, виділяючи кольорами ключові слова і парні дужки. Якщо позначити пункт «Увімкнути доповнення», **Cantor** буде показувати можливі варіанти продовження команди, яку ви вводите, у відповідь на натискання клавіші `Tab`. Якщо існує тільки одне продовження команди, після натискання клавіші `Tab` назву команди буде автоматично введено повністю.

За допомогою пункту `Увімкнути номери рядків` можна наказати програмі додати нумерацію введених виразів. Нумерацію можна використати для підставляння попередніх результатів у новий вираз.

5.2.3 Інші можливості Cantor

Документи **Cantor** зберігаються у вбудованому форматі `cws`. Кінцевий документ користувача можна зберігати у форматі LaTeX. При роботі з модулем обробки **Maxima** графічні документи можна вбудовувати у документ, за допомогою контекстного меню їх можна зберегти у форматі `eps`.

Позначення з пункту `Інтегрувати графіки на робочому аркуші` можна зняти, при цьому графіки будуть відтворюватися програмою **gnuplot** в окремому вікні. Якщо налаштовано pdf-псевдопринтер, можна роздрукувати документ **Cantor** у форматі `pdf` (зокрема включені у текст графіки і коментарі).

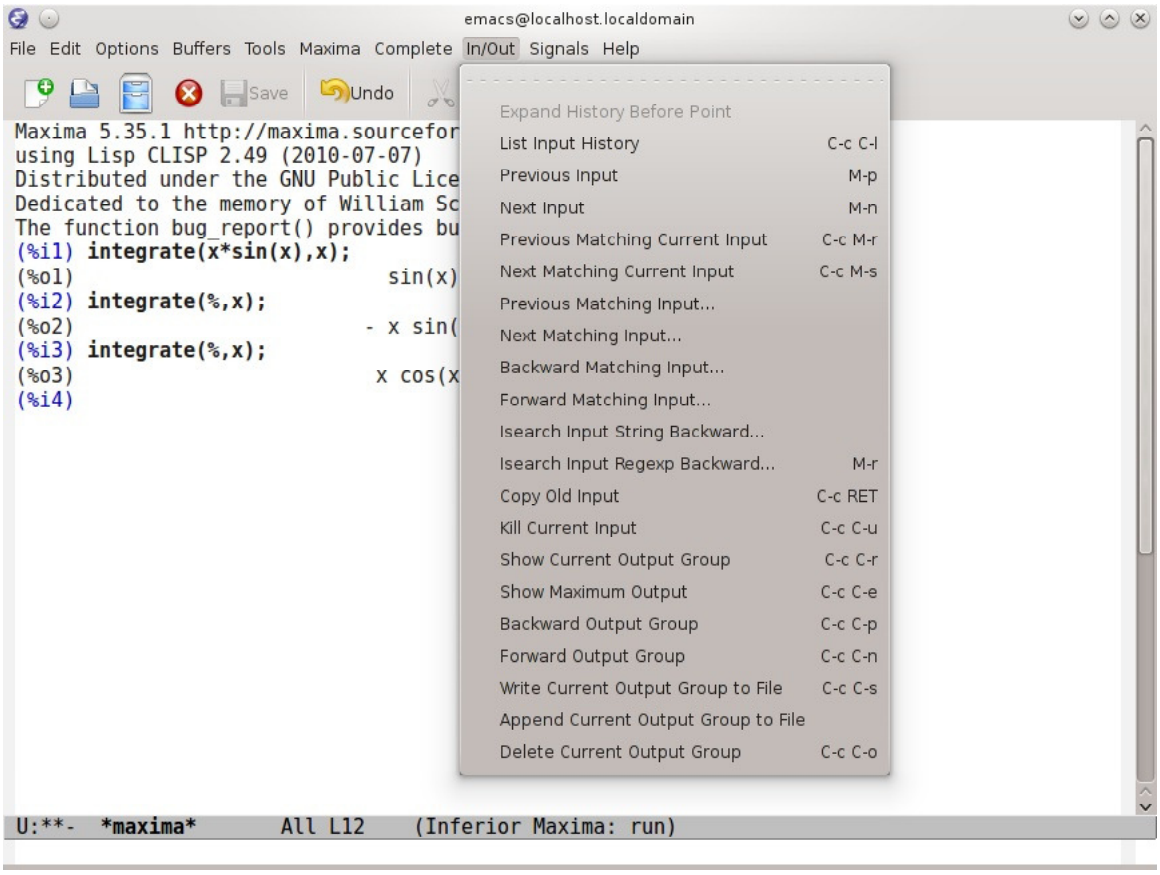


Рис. 5.12. Робоче вікно Emacs із запущеним сеансом Maxima.

5.3 Інтегроване середовище Sage

Sage (англ. «Мудрець») – система комп'ютерної алгебри, використовувати яку можна у багатьох областях математики, зокрема у алгебрі, комбінаториці, обчислювальній математиці і математичному аналізі. Перша версія **Sage** була випущена 24 лютого 2005 року у вигляді вільного програмного забезпечення з ліцензією GNU GPL. Первісною метою проекту було «створення відкритого програмного забезпечення альтернативного системам **Magma**, **Maple**, **Mathematica**, і **MATLAB**». Розробником **Sage** є Вільям Стейн — математик Університету Вашингтона (Офіційний сайт: http://sagemath.org).

Серед численних можливостей **Sage** такі:

- Інтерфейс `notebook` для перегляду і повторного використання введених команд і отриманих результатів, включаючи графіки і текстові анотації, доступний з більшості сучасних браузерів. Доступне захищене з'єднання на основі протоколу HTTPS, коли конфіденційність має значення. Крім того, **Sage** може виконуватися як локально, так і віддалено.
- Інтерфейс введення на основі командного рядка, з використанням мультипарадигмової мови IPython.
- Підтримка паралельних обчислень із використанням як багатоядерних процесорів, так і багатопроцесорних систем і систем розподілених обчислень.
- Внутрішня інфраструктура на python, що підтримує взаємодію з математичними пакетами на python: SymPy, SciPy і NumPy.
- Різні статистичні бібліотеки функцій, які використовують функціональність R і SciPy.
- Можливість побудови плоских і тривимірних графіків для функцій і даних.
- Засоби роботи з матрицями і масивами даних з підтримкою розріджених масивів.
- Набір інструментів для додавання власного інтерфейсу користувача до обчислень і додатків.
- Мережеві інструменти для з'єднання з базами даних SQL, підтримка протоколів мережі, зокрема HTTP, NNTP, IMAP, SSH, IRC, FTP.

Sage – сам собою потужний засіб завдяки численним об'єктно-орієнтованим можливостям і великому об'єму можливостей, реалізованому на python для розв'язання всіляких задач. Однак варто враховувати, що основна ідея **Sage** – інтеграція всіляких математичних пакетів, як відкритих, так і пропрієтарних. Поряд з функцією інтеграції,

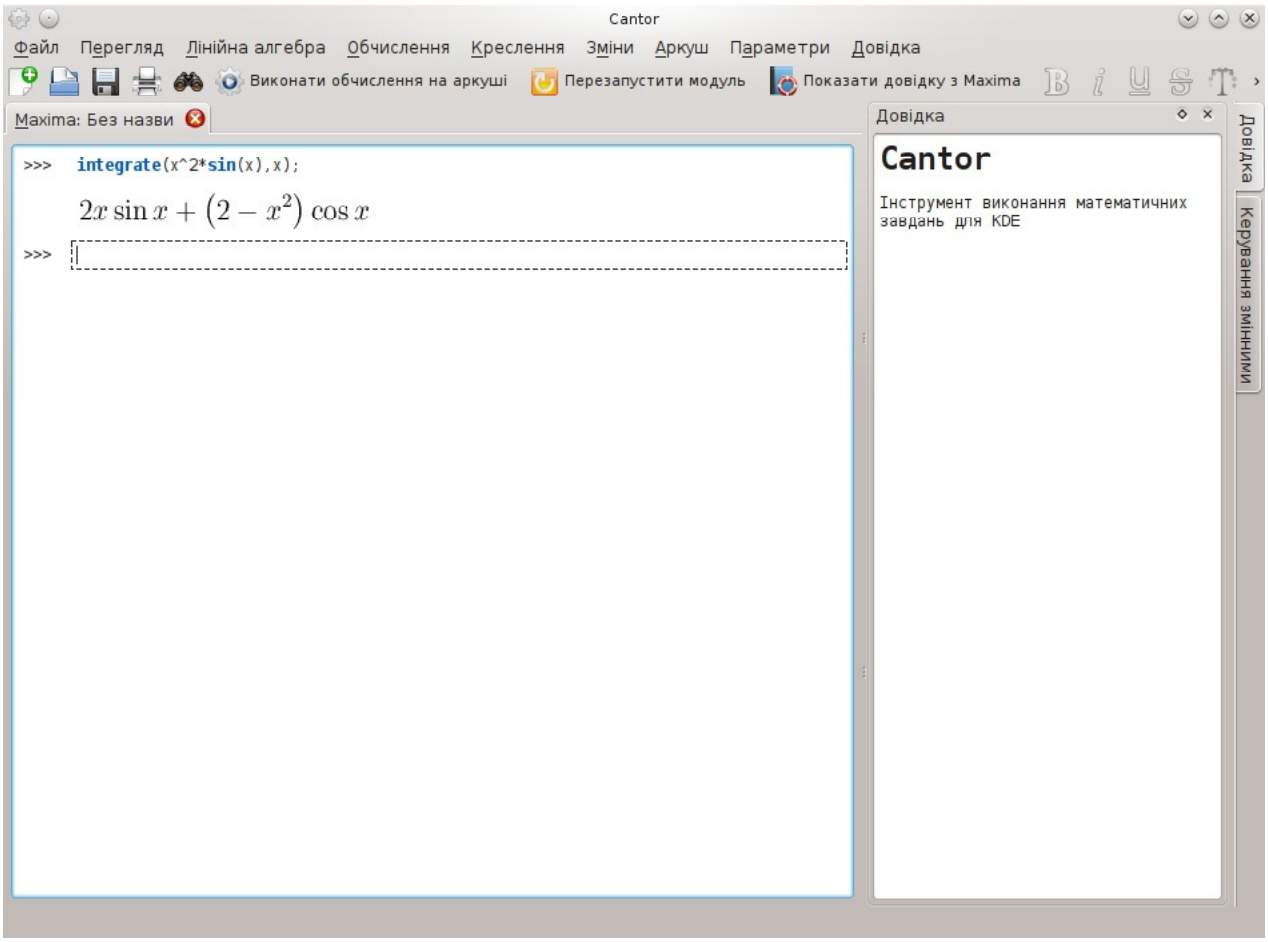


Рис. 5.13. Загальний вигляд робочого вікна Cantor.

Sage включає досить розвинені власні можливості – численні функції і структури даних. Перетворення результатів, отриманих, наприклад, у **Maxima**, до структур **Sage** може виявитися досить складною задачею.

5.4 Побудова графічних ілюстрацій: пакет `draw`

В **Maxima** є кілька альтернативних бібліотек для відтворення графіків функцій, наборів точок, тривимірних тіл, градієнтів тощо. Типово використовується бібліотека `Plot`, але для розв'язування деяких задач може виявитися зручніше бібліотека `Draw`.

Варіанти використання команди `plot2d` розглянуто вище, тому нижче ілюструються лише можливості `draw`. Бібліотека `draw` побудована на інтерфейсі `Maxima-gnuplot`. Бібліотека включає три основні функції, доступні на рівні **Maxima**: `draw2d`, `draw3d`, `draw`. Перед використанням `draw` необхідно завантажити командою `load("draw")`.

Розгляньмо нескладний приклад. На графіку (рис. 5.14) показана крива $y = exp(x)$. Графік побудований з використанням функції `draw2d`. Функції, задані явно, вказуються командою `explicit`. Для кожної функції вказується назва змінної і границі зміни абсциси. Межі ординати вибираються автоматично. Команда побудови графіка:

```
(%i4) draw2d(grid=true,xlabel = "Time", ylabel = "Population", explicit(exp(u),u,-2,2))$
```

На графіку показана сітка (`grid=true`), а також мітки осей (`xlabel` і `ylabel`).

Виведення графіка на друк здійснюється за допомогою зазначення типу термінала. Можливі варіанти – `screen` (типовий екран), `png`, `jpg`, `eps`, `eps_color`, `gif`, `animated_gif`, `wxt`, `aquaterm`.

Команда для виведення графіка на друк має вигляд (зазначено термінал `eps` – `encapsulated postscript`):

```
(%i6) draw2d(terminal=eps,grid=true,xlabel = "Time", ylabel = "Population", explicit(exp(u),u,-2,2))$
```

Типово рисунок зберігається до файла `maxima_out.eps`; вказати назву файла для виведення можна за допомогою параметра `file_name = "назва файла"`.

Побудуємо аналогічний графік (рис. 5.15), але з виведенням кривих $y = exp(x)$ і $y = exp(-x)$ в одних осях зі збереженням графіка до файла `draw_2.eps`. Відповідна команда:

```
(%i7) draw2d(terminal=eps, file_name="draw_2",grid=true, xlabel = "x",ylabel = "y",
explicit(exp(u),u,-2,2), explicit(exp(-u),u,-2,2))$
```

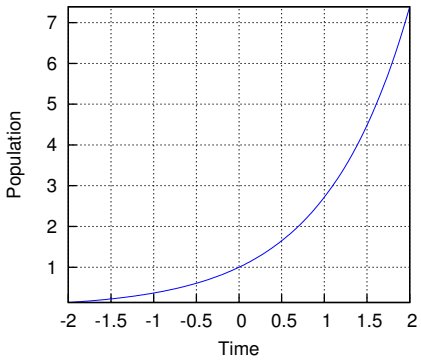


Рис. 5.14. Графік, побудований за допомогою функції `draw2d`.

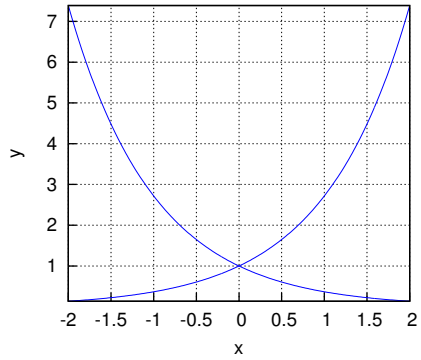


Рис. 5.15. Графік двох функцій (використана функція `draw2d`.

За допомогою пакета `draw` можна будувати і графіки функцій, заданих неявно. У цьому випадку функція задається командою `implicit`, наприклад (результат побудови – на рис. 5.16):

```
(%i1) load(draw)$
(%i2) draw2d(grid = true, title = "Two implicit functions", line_type = solid, key ="y^2=x^3-2*x+1",
implicit(y^2=x^3-2*x+1, x, -4,4, y, -4,4), line_type = dots, key="x^3+y^3=3*x*y^2-x-1",
implicit(x^3+y^3=3*x*y^2-x-1, x,-4,4, y,-4,4))$
```

На графіку добре видно, що криві проведені різними лініями (одна суцільна, інша крапчаста). Для визначення типу лінії використано параметр `line_type=тип лінії` (можливі значення – `solid` і `dots`).

Крім графіків неявних функцій, за допомогою `draw` можуть бути побудовані і графіки параметричних функцій або функцій, заданих у полярних координатах. У цих випадках замість параметрів `explicit` або `implicit` використовуються команди `parametric` і `polar` відповідно. Приклад графіка функції у полярних координатах – на рис. 5.17. Відповідна команда:

```
draw2d(user_preamble = "set grid polar", nticks = 200, xrange = [-5,5], yrange = [-5,5],
color = blue, line_width = 3, title = "Hyperbolic Spiral", polar(10/theta,theta,1,10*%pi))$
```

В останньому прикладі зазначені параметри побудови графіка: інтервали зміни `x` і `y`, рівні `xrange` і `yrange`, товщина лінії `line_width` і її колір `color`. Крім того, важливим параметром є `user_preamble`. Ця параметр вказує команди `gnuplot`, визначені користувачем, які виконуються перед побудовою графіка.

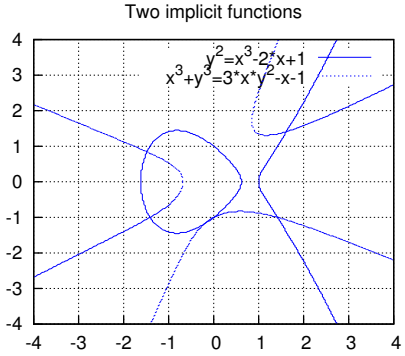


Рис. 5.16. Графік двох функцій, заданих неявно.

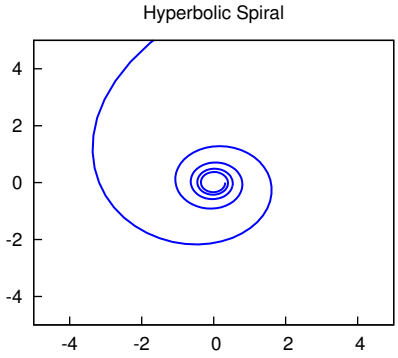


Рис. 5.17. Графік функції у полярних координатах.

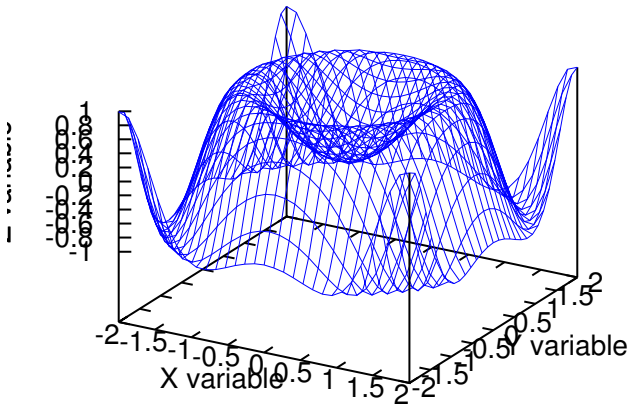


Рис. 5.18. Поверхня, побудована за допомогою функції `draw3d`.

Для використання міток осей і заголовків українською мовою слід у `user_preamble` або у спеціальному файлі `.gnuplot` (цей файл містить команди gnuplot, що виконуються під час запуску програми) указати українське кодування командою `set encoding koi8u`. Крім того, часто виникає потреба вказати і шрифт для виведення заголовка або міток.

Функція draw3d надає змогу будувати тривимірні графіки. **Приклад:**

```
(%i8) draw3d(zlabel = "Z variable",ylabel = "Y variable",
explicit(sin(x^2+y^2),x,-2,2,y,-2,2), xlabel="X variable")$
```

Мітка осі z вказується командою `zlabel=мітка`. Виведення графіка на друк аналогічне зазначеному вище. Приклад (із зазначенням, крім міток осей, і заголовка графіка командою `title=заголовок`) наведений на рис. 5.18.

Очевидно, що за допомогою функції `draw3d`, можна будувати і розфарбовані (або напівтонові) поверхні. Для цього як аргумент функції `draw3d` вказується параметр `enhanced3d` (указує на побудову тривимірної розфарбованої поверхня) і `palette` (`palette=color` – кольорова поверхня, `palette=gray` – відтінки сірого). Приклад поверхні, пофарбованої відтінками сірого – на рис. 5.19. Відповідна команда:

```
(%i12) draw3d(terminal=eps,surface_hide = true, enhanced3d = true, palette=gray,
explicit(20*exp(-x^2-y^2)-10,x,-3,3,y,-3,3))$
```

Пакет `draw` надає змогу будувати декілька графіків на одному рисунку, а також надає ряд інших корисних можливостей, але для їхнього використання необхідно ознайомитися з документацією, що поставляється з пакетом **Maxima**.

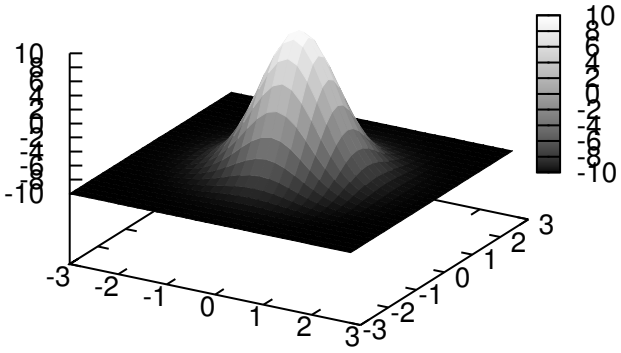


Рис. 5.19. Розфарбована поверхня (використана функція `draw3d`).

Розділ 6

Моделювання з Maxima

6.1 Загальні питання моделювання

На сьогодні існує широкий спектр літератури, присвяченої різним аспектам моделювання систем, природних, технічних і економічних об'єктів.

Одним із широко застосовуваних універсальних програмних засобів для розв'язування задач моделювання є пакет **Matlab** і його розширення для візуальної побудови блокових моделей – пакет **Simulink**. Підхід, пов'язаний з побудовою блокових моделей, розвивався протягом багатьох років, і має велику теоретичну базу.

Розглянутий у цій книзі книзі пакет **Maxima** не розрахований на побудову блокових моделей технічних систем або систем масового обслуговування, але може бути дуже корисним для побудови і аналізу аналітичних або емпіричних ідентифікованих моделей.

Побудова, класифікація та ідентифікація математичних моделей – широка область наукової і практичної діяльності, широко висвітлена у літературі різного призначення.

Визначимо модель як зображення істотних сторін реальної системи (або конструйованої системи), у зручній формі, що віддзеркалює інформацію щодо неї. Моделі бувають концептуальні, фізичні або математичні (інші назви: феноменологічні, емпіричні і аналітичні) залежно від того, який бік явища у цьому випадку є найістотнішим, від методів, які можна використати при побудові моделі, від кількості і якості наявної інформації.

На думку П. Ейкхоффа, при побудові моделі інформацію має бути представлено у зручній формі. Це істотно, оскільки модель повинна створити передумови для наступних рішень. Якщо модель занадто складна, її корисність стає сумнівною. Відносна простота є головною характеристикою моделі. Модель являє собою спрощене віддзеркалення дійсності. У багатьох випадках для того, щоб модель була корисною, її складність має перебувати у певному співвідношенні зі складністю описуваного об'єкта (приклад: біологічні системи).

За способом подання інформації про досліджуваний об'єкт і способом побудови моделі діляться на наступні групи:

- словесні або вербальні моделі (описи об'єкта моделювання природною мовою);
- фізичні моделі, що припускають подання основних властивостей об'єкта моделювання якимось матеріальним об'єктом (моделлю, макетом тощо);
- формальні моделі, що являють собою опис об'єкта моделювання формальною мовою, до цієї групи належать математичні моделі.

Далі, математичні моделі діляться на графічні, табличні, алгоритмічні, аналітичні.

Засобами **Maxima** можна будувати досить широке коло різних математичних моделей.

Процедуру побудови моделі у багатьох джерелах називають ідентифікацією, при цьому даний термін часто ставиться до побудови аналітичних математичних моделей динамічних об'єктів.

Звичайно ідентифікація – багатоетапна процедура. Основні її етапи такі:

1. Структурна ідентифікація полягає у визначенні структури математичної моделі на підставі теоретичних міркувань.
2. Параметрична ідентифікація містить у собі проведення експерименту з ідентифікації і визначення оцінок параметрів моделі за експериментальним даними.
3. Перевірка адекватності – перевірка якості моделі у сенсі обраного критерію близькості результатів застосування моделі і об'єкта.

6.1.1 Аналітичні моделі

Аналітичні моделі являють собою відбиття взаємозв'язків між змінними об'єкта у вигляді математичної формули або групи таких формул. Моделювання засноване на двох основних ознаках:

- на принципі практичної обмеженості кількості фундаментальних законів природи;

- на принципі подібності, що означає, що явища різної фізичної природи можуть описуватися однаковими математичними залежностями.

Для аналітичного моделювання характерно те, що процеси роботи елементів системи записуються у вигляді деяких функціональних співвідношень (алгебраїчних, інтегро-диференціальних, скінченно-різницевих тощо) або логічних умов. Аналітична модель може бути досліджена наступними методами:

- аналітичним, коли прагнуть одержати в загальному вигляді явні залежності для шуканих характеристик;
- обчислювальним, коли, не вміючи розв'язувати рівнянь у загальному вигляді, прагнуть одержати числові результати при конкретних початкових даних;
- якісним, коли, не маючи розв'язку у явному вигляді, можна знайти деякі властивості розв'язку (наприклад, оцінити стійкість розв'язку).

Найбільш повне дослідження процесу роботи системи можна виконати, якщо відомі явні залежності, що пов'язують шукані характеристики з початковими умовами, параметрами і змінними системи. Однак такі залежності вдається одержати тільки для порівняно простих систем. При ускладненні систем дослідження їхнім аналітичним методом натрапляє на значні труднощі, які часто бувають непереборними. Тому, бажаючи використати аналітичний метод, у цьому випадку йдуть на істотне спрощення первісної моделі, щоб мати можливість вивчити хоча б загальні властивості системи. Таке дослідження на спрощеній моделі аналітичним методом допомагає одержати орієнтовні результати для визначення більше точних оцінок іншими методами.

Можливості суто теоретичної побудови математичної моделі зменшуються з ростом складності і новизни досліджуваного об'єкта. Втім, досвід показує, що нерідко навіть для широко використовуваних на практиці й, здавалося б добре вивчених об'єктів і процесів, суто аналітичним шляхом побудувати задовільну модель не вдається і це спонукує дослідника до формування моделі переважно на експериментальній основі, тобто у класі емпіричних (ідентифікованих) моделей.

За степенем відповідності моделі реальному об'єкту моделі можна поділити на такі: на спроможні – такі, що спираються на закони, що характеризують об'єкт моделювання в області їхньої застосовності; апроксимації – побудовані на основі наближених або емпіричних формул, що характеризують об'єкт (їх, на відміну від перших, називають неспроможними).

6.1.2 Ідентифіковані моделі

В основі всіх нині достатньо численних методів ідентифікації або дослідного ототожнення моделі з об'єктом-оригіналом, лежить ідея уявного експерименту з «чорною скринькою» (Н. Вінер). У граничному (теоретичному) випадку «чорна скринька» являє собою якусь систему, про структуру і внутрішні властивості якої невідомо анічогісінько. Зате входи, тобто зовнішні фактори, що впливають на цей об'єкт, і виходи, що представляють собою реакції на вхідні впливи, доступні для спостережень (вимірювання) протягом необмеженого часу. Задача полягає у тому, щоб за спостережуваним даними про входи і виходи виявити внутрішні властивості об'єкта або, іншими словами, побудувати модель.

Модель чорної скриньки є початковим етапом вивчення складних систем.

Дослідження об'єкта моделювання допускає застосування двох стратегій:

1. Здійснюється активний експеримент. На вхід подаються спеціальні сформовані тестові сигнали, характер і послідовність яких визначена заздалегідь розробленим планом. Перевага: за рахунок оптимального планування експерименту необхідна інформація про властивості і характеристики об'єкта виходить при мінімальному об'ємі первинних експериментальних даних і відповідно при мінімальній трудомісткості дослідних робіт. Але ціна за це є досить високою: об'єкт виводиться з його природного стану (або режиму роботи), що не завжди можливо.
2. Здійснюється пасивний експеримент. Об'єкт працює у своєму природному режимі, але при цьому організуються систематичні виміри і реєстрація значень його вхідних і вихідних змінних. Інформацію одержують ту ж, але необхідний об'єм даних звичайно більше, ніж у першому випадку.

На практиці при побудові ідентифікованих (емпіричних) моделей часто доцільна змішана стратегія експерименту. При наявності можливості вільного маніпулювання параметрами об'єкта моделювання виконується активний експеримент. Його результати доповнюють даними пасивного експерименту, що охоплює всі інші значимі змінні. «Чорна скринька» – теоретично граничний випадок. Насправді маємо лише об'єм вихідної інформації. На практиці доводиться мати справа з «сірою» (почасти прозорою) скринькою.

Побудова моделі зводиться до наступних етапів:

1. вибір структури моделі з фізичних міркувань;
2. припасування параметрів до наявних даних (оцінювання);
3. перевірка і підтвердження моделі (діагностична перевірка);
4. використання моделі за її призначенням.

Виходячи з переліку наукових напрямків і розмаїтості додатків, за цими етапами не можна дати якихось загальних рекомендацій. Структура моделі вибирається на основі вихідної (апріорної) інформації про систему і кінцевих цілей. На практиці відшукання відповідної моделі може бути досить важкою задачею навіть для вузької прикладної області.

Розрізняють три основних класи постановки задачі ідентифікації об'єкта:

1. Для складних і слабко вивчених об'єктів системного характеру достовірних вихідних даних про внутрішні властивості і структурні особливості зазвичай немає.

   Тому задача ідентифікації містить у собі як визначення внутрішньої структури об'єкта, так і визначення залежностей, що зв'язують входи і виходи (узагальненого оператора). На початковій стадії моделювання будуються емпіричні ідентифікувальні моделі (на основі статистичної обробки експериментальних результатів).

2. Другий клас задач ідентифікації характеризується тим, що є апріорні дані про структуру модельованого об'єкта, у принципі є. Однак не визначений внесок компонентів об'єкта у його остаточні характеристики. Задачі цього класу, пов'язані з уточненням структури і оцінювання параметрів, часто зустрічаються на практиці і характерні для об'єктів і процесів середньої складності, зокрема технологічних.

3. Третій клас задач пов'язаний з відносно простими і добре вивченими об'єктами, структура яких відома точно і мова йде тільки про те, щоб за експериментальним даними оцінити значення всіх або деяких вхідних у досліджувану структуру параметрів (параметрична ідентифікація). Очевидно, що моделі даного класу тісно змикаються з аналітичними моделями, що потребують додаткового експериментального довизначення, і чіткої границі між ними не існує. Це найбільш масовий клас задач.

Незалежно від характеру розв'язуваної на основі ідентифікації задачі, побудова моделі базується на результатах вимірів відповідних величин змінних.

Реальні властивості переважної більшості складних об'єктів, а також неминучі випадкові погрішності вимірів, що лежать в основі ідентифікації, надають останній статистичний характер, що спричиняє необхідність одержання більших об'ємів первинних експериментальних даних з їхньою наступною обробкою. Тому на практиці побудова моделей шляхом ідентифікації неминуче пов'язана з використанням комп'ютерів, як при одержанні первинних даних (автоматизація експерименту), так і для їхньої обробки і використання.

До **Maxima** включено досить велику кількість засобів для розв'язування задач моделювання, параметричної ідентифікації, дослідження моделей.

6.2 Статистичні методи аналізу даних

6.2.1 Введення-виведення матричних даних

Для читання і запису матричних або потокових даних у складі **Maxima** передбачений пакет `numericalio`.

Функції пакета розраховані на введення-виведення даних, кожне поле яких вважається атомом (у сенсі **Lisp**), тобто цілих чисел, чисел із рухомою крапкою, рядків або символів. Атоми сприймаються `numericalio` так само, як при інтерактивному уведенні у консолі або виконанні пакетного файла. Можливе використання різних символів-роздільників для поділу полів даних (параметр `separator_flag`).

Основні функції пакета numericalio:

- `read_matrix(file_name)` (інші форми виклику – `read_matrix(file_name, separator_flag)`, `read_matrix(S)`, `read_matrix(S, separator_flag)`). Функція read_matrix зчитує матрицю з файла. Тут `file_name` – назва файла, з якого зчитуються дані, `S` – назва потоку. Якщо не зазначено `separator_flag`, дані вважаються розділеними пробілами. Функція повертає зчитаний об'єкт.

- `read_list(file_name)` (інші форми – `read_list(file_name, separator_flag)`, `read_list(S)`, `read_list (S, separator_flag)`) – зчитує список з файла або з потоку.

- `write_data(X, file_name)` (інші форми — `write_data(object, file_name, separator_flag)`, `write_data(X, S)`, `write_data(object, S, separator_flag)`) — здійснює виведення об'єкта `object` (списку, матриці, масиву **Lisp** або **Maxima** тощо) до файла `file_name` (або об'єкта `X` до потоку `S`). Матриці виводяться за стовпцями і рядками з використанням пробілу або іншого символу-роздільника (див. `separator_flag`).

Поряд із зазначеними простими функціями, використовуються більше специфічні: `read_lisp_array`, `read_maxima_array`, `read_hashed_array`, `read_nested_list`, призначені для зчитування масивів у форматі **Lisp** або **Maxima**, особливості застосування яких не розглядаються у цій книзі.

6.2.2 Функції Maxima для розрахунку описової статистики

Система **Maxima** містить ряд функцій для виконання статистичних розрахунків (описової статистики), об'єднані у пакет `descriptive`. Функції, що входять до складу `descriptive`, надають змогу виконати обчислення дисперсії, середньоквадратичного відхилення, медіани, моди тощо. Назви функцій і короткий опис виконуваних ними дій наведені у таблиці 6.1.

Табл. 6.1. Функції пакета `descriptive`

Функція	**Виконувані дії**	**Синтаксис виклику і примітки**
`mean`	Обчислення середнього	`mean(list)` або `mean(matrix)`
`geometric_mean`	Обчислення середнього геометричного	`geometric_mean(list)` або `geometric_mean(matrix)`
`harmonic_mean`	Обчислення середнього гармонійного	`harmonic_mean(list)` або `harmonic_mean(matrix)`
`cor`	Обчислює кореляційну матрицю	`cor(matrix)` або `cor(matrix,logical_value)` `logical_value` дорівнює `true` або `false` (при розрахунку за коваріаційною матрицею)
`cov`, `cov1`	Обчислює коваріаційну матрицю	`cov1(matrix)`, `cov(matrix)`
`median`	Обчислює медіану	`median(list)`, `median(matrix)`
`std`, `std1`	Обчислює середньоквадратичне відхилення (корінь квадратний з `var` або `var1`)	аналогічно `var`
`var`, `var1`	Обчислює дисперсію випадкової величини	`var1(matrix)`, `var(matrix)`, `var1(list)`, `var(list)`
`central_moment`	Обчислює центральний момент порядку k	`central_moment(list,k)`, `central_moment(matrix,k)`
`noncentral_moment`	Обчислює момент порядку k	`noncentral_moment(list,k)`, `noncentral_moment(matrix,k)`
`skewness`	Обчислення асиметрії	`skewness(list)`, `skewness(matrix)`
`kurtosis`	Обчислення ексцесу	`kurtosis(list)`, `kurtosis(matrix)`
`quantile`	Обчислення p-квантиля	`quantile(list,p)`, `quantile(matrix,p)`
`maxi`, `mini`	Вибір найбільшого і найменшого значення у вибірці відповідно	`maxi(list)`, `maxi(matrix)`, `mini(list)`, `mini(matrix)`
`mean_deviation`, `median_deviation`	Сума абсолютних відхилень від середнього або медіани відповідно	Аналогічно до `mean`, `median`
`range`	Розмах варіації вибірки	`range(list)`, `range(matrix)`
`list_correlations`	Повертає список, що включає дві матриці — матрицю, обернену до коваріаційної, і матрицю частинних коефіцієнтів кореляції	`list_correlations(matrix)`, `list_correlations(matrix,logical_value)`, де `logical_value` дорівнює `true` або `false` (при розрахунку за коваріаційною матрицею)
`subsample`	Аналог функції `submatrix`	
`global_variances`	Повертає список, що містить різні типи дисперсії	`global_variances(matrix)`

Побудова графічних ілюстрацій виконується за допомогою функцій `scatterplot` (безпосередня візуалізація даних), `histogram` (будує гістограму), `barsplot` (також будує гістограму, але за дискретними або нечисловими даними), `boxplot` (графік Бокса-Віскера), `piechart` (кругова діаграма). Синтаксис виклику і параметри функцій багато в чому аналогічні компонентам пакета `draw` (див. приклади нижче).

Основні загальні параметри (успадковані від `draw`):

- `terminal` – пристрій, куди виводиться діаграма (можливі значення — `eps` і `png`, типово діаграму буде показано на екрані, інші варіанти — див. стор. 118);
- `file_name` – назва файла, до якого слід вивести гістограму (суфікс встановлюється за параметром `terminal`)
- `title` – основний заголовок;
- `xlabel`, `ylabel` – назви (мітки) осей,

Розгляньмо приклад використання функцій пакета `descriptive` для статистичної обробки масивів даних. Дані беремо з файлів, що входять до складу пакета `descriptive` (файли `biomed.data`, `wind.data` та ін.). Перед початком роботи завантажуємо необхідні пакети `descriptive` і `numericalio`. За допомогою функції `read_matrix` зчитується матриця, що містить 100 рядків і 5 стовпців.

```
(%i1) load(descriptive)$ load(numericalio)$ s:read_matrix (file_search ("wind.data"))$ length(s);
(%o4) 100
(%i5) mean(s); /*розраховуємо середнє значення. При обробці матриці одержуємо список середніх
        за стовпцями.*/
(%o5) [9.948499999999999, 10.1607, 10.8685, 15.7166, 14.8441]
```

```
(%i6) median(s);
(%o6) [10.06, 9.855, 10.73, 15.48, 14.105]
(%i7) var(s);
(%o7) [17.22190675000001, 14.98773651000001, 15.47572875, 32.17651044000001, 24.42307619000001]
(%i8) std(s);
(%o8) [4.149928523480858, 3.871399812729242, 3.933920277534866, 5.672434260526957, 4.941970881136392]
(%i9) mini(s);
(%o9) [0.58, 0.5, 2.67, 5.25, 5.17]
(%i10) maxi(s);
(%o10) [20.25, 21.46, 20.04, 29.63, 27.63]
(%i11) mini(%); /* При обробці списку і пошуку у ньому мінімального елемента одержуємо одне значення!*/
(%o11) 20.04
```

Для побудови діаграм розкиду (xy-діаграм) призначено функцію `scatterplot`. Синтаксис виклику:
`scatterplot(list)`
`scatterplot(list,` $option_1$`,` $option_2$`,...)`
`scatterplot(matrix)`
`scatterplot(matrix,` $option_1$`,` $option_2$`,...)`
Дані для функції `scatterplot` можуть представлятися вектором (списком) або матрицею. Одновимірні масиви розглядаються як тимчасові ряди з рівновіддаленими точками.

Основні параметри, специфічні для даної функції:

- `point_size` – розмір точки на графіку (ціле додатне число);
- `point_type` – вид точки (відсутність точок – `none` (-1), `dot` (0), `plus` (1), `multiply` (2), `asterisk` (3), `square` (4), `filled_square` (5), `circle` (6), `filled_circle` (7), `up_triangle` (8), `filled_up_triangle` (9), `down_triangle` (10), `filled_down_triangle` (11), `diamant` (12), `filled_diamant` (13));
- `color` – кольори точки (той же набір кольорів, що і у пакеті `draw`);
- `grid` – наявність сітки на графіку (`true`/`false`).

Для побудови гістограм використовується функція `histogram` (синтаксис виклику аналогічний `scatterplot` і основні параметри ідентичні параметрам `scatterplot`). Розгляньмо додаткові параметри, специфічні для `histogram`:

- `nclasses` (типово 10) – число класів гістограми, або список із зазначенням границь класів і їхньої кількості, або тільки границі;
- `frequency` – указує масштаб шкали ординат, можливі значення: абсолютний, відносний і процентний (`default`, `absolute`, `percent`);
- `htics(default, auto)` – формат проміжних ділень на гістограмі, можливі значення – `auto`, `endpoints`, `intervals`, або список міток.

Під час побудови гістограми доступні також локальні і глобальні параметри пакета `draw`.

Для графічного подання описової статистики слугує діаграма Бокса-Віскера, що є зручним способом наочно представити статистичні дані п'ятьма параметрами: найменше і найбільше значення вибірки, нижній, середній і верхній квартилі. На даній діаграмі можуть бути показані і викиди (якщо вони є).

Для побудови діаграм Бокса-Віскера використовується функція `boxplot`. Синтаксис виклику: `boxplot(data)` або `boxplot(data,` $option_1$`,` $option_2$`,...)`.

Параметр `data` – список або матриця з декількома стовпцями. Параметри функції `boxplot` ідентичні параметрам `scatterplot`.

Стовпчасті діаграми (звичайно частотні) будуються для даних, розбитих на категорії, за допомогою функції `barsplot`. Ці діаграми надають змогу графічно представити розходження між даними категорій.

Досить розповсюдженим способом графічного зображення структури статистичних сукупностей є секторна діаграма, тому що ідея цілого дуже наочно виражається кругом, що представляє всю сукупність. Відносна величина кожного значення зображається у вигляді сектора круга, площа якого відповідає внеску цього значення у суму значень. Цей тип графіків будується у **Maxima** функцією `piechart`.

Розгляньмо приклади використання графічних утиліт пакета `descriptive`.

Для подальшого використання зчитуємо дані з файла `wind.data` (це тестовий файл, що є частиною пакета `descriptive`, містить матрицю 100×5).

```
(%i1) load(descriptive)$ load(numericalio)$ s:read_matrix(file_search("wind.data"))$
(%i4) x:makelist(s[k][1],k,1,length(s))$
(%i5) y:makelist(s[k][2],k,1,length(s))$
(%i6) m:makelist([x[k],y[k]],k,1,100)$
(%i7) xy:apply('matrix,m)$
```

Будуємо графік (точковий) залежності `y` від `x` (див. рис. 6.1). Результати зберігаються у файлі `maxima_out.eps` (назва файла – типова, він створюється у домашньому каталозі користувача). Відповідна команда:

```
(%i8) scatterplot(xy,terminal=eps);
```

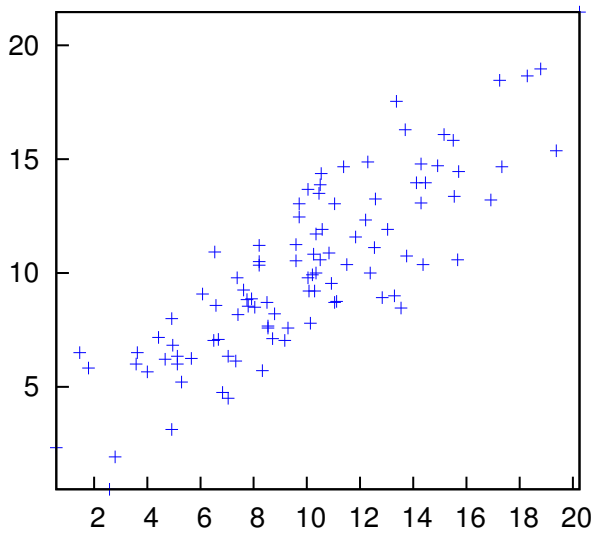


Рис. 6.1. Точковий графік.

Зчитуючи дані з файла `pidigits.data`, будуємо гістограму частотного розподілу десяткових знаків числа π (див. рис. 6.2). Результати зберігаються у файлі `histogram.eps` (назва файла задається параметром `file_name="histogram"`, він створюється в домашньому каталозі користувача). Відповідні команди:

```
load (descriptive)$ s1:read_list(file_search("pidigits.data"))$
histogram(s1, nclasses=8, title="pi digits", xlabel="digits", ylabel="Absolute frequency",
fill_color=grey, fill_density=0.6, terminal=eps, file_name="histogram")$
```

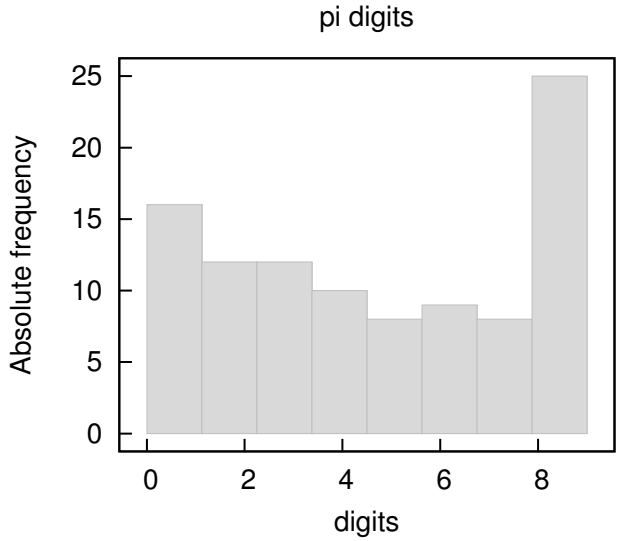


Рис. 6.2. Гістограма.

Наступний приклад – графік Бокса-Віскера з анотаціями за осями (див. рис. 6.3). Відповідні команди (результат зберігається у файлі `boxwisker.eps`):

```
(%i10) boxplot(s,title="Test plot", xlabel="Seasons", terminal=eps, file_name="boxwisker")$
```

Приклад побудови стовпчастої діаграми з використанням функції `barsplot` – на рис. 6.4. Графік побудовано такою командою (результат зберігається у файл `barsplot.eps`):

```
load (descriptive)$ l1:makelist(random(8),k,1,50)$ l2:makelist(random(8),k,1,100)$
barsplot(l1, l2, box_width=1/2, fill_density=3/4, sample_keys=["A","B"],
bars_colors=[grey10,grey50], terminal=eps, file_name="barsplot")$
```

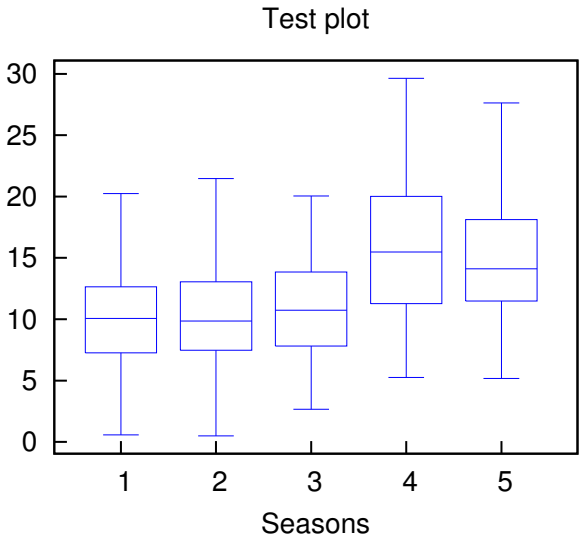


Рис. 6.3. Графік Бокса-Віскера.

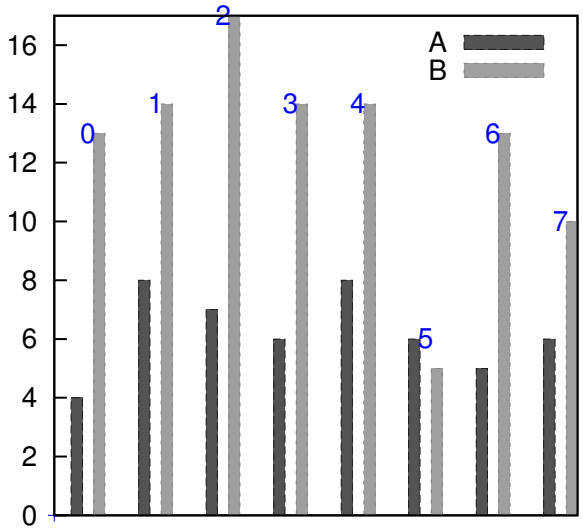


Рис. 6.4. Гістограма розподілу у групах.

Основні параметри команди `barsplot`:

- `box_width` – відносна ширина прямокутників (типово 3/4, величина в межах [0,1]);
- `grouping` – індикатор представлення складених значень (можливі значення `clustered` і `stacked`);
- `groups_gap` – натуральне число, що відповідає розриву між двома сусідніми групами (відносна величина, типово 1);
- `bars_colors` – список кольорів для складених зразків (типово [ ]);
- `start_at` – указує початок графіка за віссю x (типово 0).

З функцією `barsplot` можна використати і параметри пакета `draw`.
Для побудови кругових (секторних) діаграм використовується функція `piechart`.
Приклад використання `piechart` (рис. 6.5):

```
load(descriptive)$
s1:read_list(file_search("pidigits.data"))$
piechart(s1, xrange=[-1.1, 1.3], yrange=[-1.1, 1.1], title="Digit frequencies in pi")$
```

Кольори секторів і радіус діаграми описуються параметрами `sector_colors` і `pie_radius`.
На жаль, базова програма виведення графіки **Maxima** – gnuplot – написана дуже давно, і сприймає кириличні символи тільки у кодуванні KOI8-U (в останніх версіях і utf8). Можливим розв'язанням (прийнятим для побудови графіків у цій книзі) є створення файла `.gnuplot`, що містить наступні команди: `set encoding koi8u` або `set encoding utf8`. Іншим варіантом виходу з ситуації є використання термінала `pdfcairo` (параметр `terminal=pdfcairo`) для створення рисунків у форматі PDF.

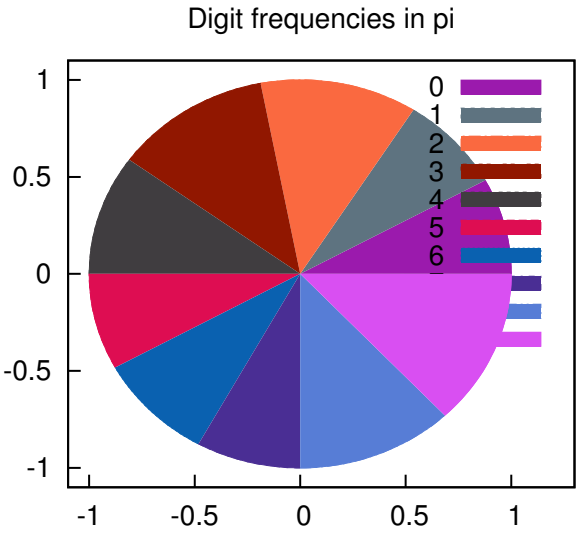


Рис. 6.5. Кругова діаграма.

6.2.3 Перевірка статистичних гіпотез

Для перевірки статистичних гіпотез до **Maxima** включений пакет `stats`. Він надає змогу, зокрема, проводити зіставлення середніх або дисперсій двох вибірок. Передбачено і перевірку нормальності розподілу, а також ряд інших стандартних тестів. для використання `stats` пакет слід завантажити командою `load("stats")`; потрібні для роботи пакети `descriptive` і `distrib` завантажуються автоматично.

Функції пакета `stats` повертають дані типу `inference_result`. Об'єкти цього типу містять потрібні для аналізу статистичних розподілів і перевірки гіпотез результати.

Функція `test_mean` надає змогу оцінити середнє значення і довірчий інтервал за вибіркою. Синтаксис виклику: `test_mean(x)` або `test_mean (x,` $option_1$`,` $option_2$`,...)`.

Функція `test_mean` використовує перевірку за критерієм Стьюдента. Аргумент `x` – список або одновимірна матриця із тестованою вибіркою. Можливе також використання центральної граничної теореми (параметр `asymptotic`). Параметри test_mean:

- `'mean`, типово 0, очікуване середнє значення;
- `'alternative`, типово `'twosided`, тип гіпотези, яку перевіряють (можливі значення `'twosided`, `'greater` і `'less`);
- `'dev`, типово `'unknown`, величина середньоквадратичного відхилення, якщо воно відоме (`'unknown` або додатне вираз) ;
- `'conflevel`, типово 95/100, рівень значущості для довірчого інтервалу (величина у межах від 0 до 1);
- `'asymptotic`, типово `false`, указує, який критерій слід використати (t-критерій або центральну граничну теорему).

Результати, які повертає функція:

- `'mean_estimate` – середнє за вибіркою;
- `'conf_level` – рівень значущості, вибраний користувачем;
- `'conf_interval` – оцінка довірчого інтервалу;
- `'method` – використана процедура;
- `'hypotheses` – статистичні гіпотези, які перевіряють (нульова H_0 і альтернативна H_1);
- `'statistic` – кількість степенів свободи для перевірки нульової гіпотези;
- `'distribution` – оцінка розподілу вибірки;
- `'p_value` – імовірність помилкового вибору гіпотези H_1, якщо виконується H_0.

Приклади використання `test_mean`:

Виконується t-тест із невідомою дисперсією. Нульова гіпотеза H_0: середнє дорівнює 50 проти альтернативної гіпотези H_1: середнє менше 50; відповідно до результатів розрахунку, величина ймовірності p занадто велика, щоб відкинути H_0.

```
(%i1) load("stats")$
(%i2) data: [78,64,35,45,45,75,43,74,42,42]$
(%i3) test_mean(data,'conflevel=0.9,'alternative='less,'mean=50);
```

(%o3)
$$\left(\begin{array}{c} MEAN\ TEST \\ mean\_estimate = 54.3 \\ conf\_level = 0.9 \\ conf\_interval = [-\infty\ , 61.51314273502714] \\ method = Exact\ t-test.\ Unknown\ variance. \\ hypotheses = H0:\ mean\ =\ 50\ ,\ H1:\ mean\ <\ 50 \\ statistic = 0.8244705235071678 \\ distribution = [student\_t, 9] \\ p\_value = 0.7845100411786887 \end{array}\right)$$

Наступний тест – перевірка гіпотези H_0 (середнє дорівнює 50) проти альтернативної гіпотези H_1 середнє за вибіркою відмінне від 50. Відповідно до величини $p \ll 1$ приймається нульова гіпотеза. Даний тест застосовується для великих вибірок.

```
(%i1) load("stats")$
(%i2) test_mean([36,118,52,87,35,256,56,178,57,57,89,34,25,98,35, 98,41,45,198,54,79,63,35,45,44,
75,42,75,45,45,45,51,123,54,151],'asymptotic=true,'mean=50);
```

(%o2)
$$\left(\begin{array}{c} MEAN\ TEST \\ mean\_estimate = 74.88571428571429 \\ conf\_level = 0.95 \\ conf\_interval = [57.72848600856193, 92.04294256286664] \\ method = Large\ sample\ z-test.\ Unknown\ variance. \\ hypotheses = H0:\ mean\ =\ 50\ ,\ H1:\ mean\ \#\ 50 \\ statistic = 2.842831192874313 \\ distribution = [normal, 0, 1] \\ p\_value = 0.004471474652002261 \end{array}\right)$$

Функція `test_means_difference` надає змогу перевірити, чи належать вибірки x_1 і x_2 до однієї генеральної сукупності.

Синтаксис виклику: `test_means_difference(`x_1`, `x_2`)` або `test_means_difference (`x_1`,`x_2`,`$option_1$`,`$option_2$`,...)`.

Ця функція виконує t-тест для порівняння середніх за вибірками x_1 і x_2 (x_1, x_2 – списки або одновимірні матриці). Порівняння вибірок може виконуватися також на підставі центральної граничної теореми (для великих вибірок). Параметри функції `test_means_difference` такі ж, як і для `test_mean`, крім оцінок середньоквадратичних відхилень вибірок (якщо вони відомі) Список параметрів:

- `'alternative`, типово `'twosided`, тип гіпотези, яку перевіряють (можливі значення: `'twosided`, `'greater` і `'less`);
- `'dev1`, `'dev2`, типово `'unknown`, величини середньоквадратичних відхилень для вибірок x_1 і x_2, якщо вони відомі (`'unknown` або додатній вираз);
- `'conflevel`, типово 95/100, рівень значущості для довірчого інтервалу (величина у межах від 0 до 1);
- `'asymptotic`, типово `false`, указує, який критерій слід використати (t-критерій або центральну граничну теорему).

Виведення результатів `test_means_difference` не відрізняється від виведення результатів `test_mean`.

Приклади використання `test_means_difference`: Для двох малих вибірок перевіряється гіпотеза H_0 щодо рівності середніх проти альтернативної гіпотези H_1: розходження математичних очікувань статистично значиме, тобто вибірки належать до різних генеральних сукупностей.

```
(%i1) load("stats")$
(%i2) x:[20.4,62.5,61.3,44.2,11.1,23.7]$
(%i3) y:[1.2,6.9,38.7,20.4,17.2]$
(%i4) test_means_difference(x,y,'alternative='greater);
```

(%o4)
$$\left(\begin{array}{c} DIFFERENCE\ OF\ MEANS\ TEST \\ diff\_estimate = 20.31999999999999 \\ conf\_level = 0.95 \\ conf\_interval = [-0.04597417812881588, \infty\ ] \\ method = Exact\ t-test.\ Welch\ approx. \\ hypotheses = H0:\ mean1\ =\ mean2\ ,\ H1:\ mean1\ >\ mean2 \\ statistic = 1.838004300728477 \\ distribution = [student\_t, 8.62758740184604] \\ p\_value = 0.05032746527991905 \end{array}\right)$$

Оцінка довірчого інтервалу для дисперсії вибірки виконується за допомогою функції `test_variance`.

Синтаксис виклику: `test_variance(x)` або `test_variance(x,` $option_1$`,` $option_2$`,...)`

Ця функція використовує тест χ^2. Вважається, що розподіл вибірки x нормальний. Параметри функції `test_variance`:

- `'mean`, типово `'unknown`, оцінка математичного очікування (середнє за вибіркою), якщо воно відомо;
- `'alternative`, типово `'twosided`, тип гіпотези, яку перевіряють (можливі значення `'twosided`, `'greater` і `'less`);
- `'variance`, типово 1, це оцінка дисперсії вибірки для порівняння з фактичною дисперсією;
- `'conflevel`, типово 95/100, рівень значущості для довірчого інтервалу (величина у межах від 0 до 1).

Основний результат, який повертає функція – оцінка дисперсії вибірки `var_estimate` і довірчий інтервал для неї.

Приклад: Перевірка, чи відрізняється дисперсія вибірки з невідомим математичним очікуванням від значення 200.

```
(%i1) load("stats")$
(%i2) x:[203,229,215,220,223,233,208,228,209]$
(%i3) test_variance(x,'alternative='greater,'variance=200);
```

(%o3)
$$\left(\begin{array}{c} VARIANCE\ \ TEST \\ var\_estimate = 110.75 \\ conf\_level = 0.95 \\ conf\_interval = [57.13433376937479, \infty\ ] \\ method = Variance\ \ Chi - square\ \ test.\ \ Unknown\ \ mean. \\ hypotheses = H0:\ \ var\ \ =\ \ 200\ \ ,\ \ H1:\ \ var\ \ >\ \ 200 \\ statistic = 4.43 \\ distribution = [chi2, 8] \\ p\_value = 0.8163948512777688 \end{array}\right)$$

Порівняння дисперсій двох вибірок виконується за допомогою функції `test_variance` (синтаксис виклику `test_variance_ratio(`x_1`,` x_2`)` або `test_variance_ratio(`x_1`,` x_2`,` $option_1$`,` $option_2$`,...)`).

Дана функція призначена для зіставлення дисперсій двох вибірок з нормальним розподілом за критерієм Фішера (F-тест). Аргументи x_1 і x_2 – списки або одновимірні матриці, що містять незалежні вибірки.

Параметри функції `test_variance_ratio`:

- `'mean1`, `'mean2`, типово `'unknown`, оцінки математичних очікувань вибірок x_1 і x_2, якщо вони відомі;
- `'alternative`, типово `'twosided`, тип гіпотези, яку перевіряють (можливі значення `'twosided`, `'greater` і `'less`);
- `'conflevel`, типово 95/100, рівень значущості для довірчого інтервалу (величина у межах від 0 до 1).

Основний результат, який повертає функція `test_variance_ratio` – відношення дисперсій вибірок `ratio_estimate`.

Приклад: перевіряється гіпотеза про рівність дисперсій двох вибірок у порівнянні з альтернативною гіпотезою про те, що дисперсія першої більше, ніж дисперсія другої.

```
(%i1) load("stats")$
(%i2) x:[20.4,62.5,61.3,44.2,11.1,23.7]$
(%i3) y:[1.2,6.9,38.7,20.4,17.2]$
(%i4) test_variance_ratio(x,y,'alternative='greater);
```

(%o4)
$$\left(\begin{array}{c} VARIANCE\ \ RATIO\ \ TEST \\ ratio\_estimate = 2.316933391522034 \\ conf\_level = 0.95 \\ conf\_interval = [0.3703504689507263, \infty\ ] \\ method = Variance\ \ ratio\ \ F - test.\ \ Unknown\ \ means. \\ hypotheses = H0:\ \ var1\ \ =\ \ var2\ \ ,\ \ H1:\ \ var1\ \ >\ \ var2 \\ statistic = 2.316933391522034 \\ distribution = [f, 5, 4] \\ p\_value = 0.2179269692254463 \end{array}\right)$$

За відсутності уявлень щодо розподілу вибірки може використовуватися непараметричний тест для порівняння середніх. Оцінка медіани неперервної вибірки виконується за допомогою функції `test_sign`. Синтаксис виклику: `test_sign(x)` або `test_sign(x,` $option_1$`,` $option_2$`,...)`.

Функція `test_sign` допускає дві параметри: `alternative` (аналогічно `test_mean`) і `median` (типово 0, або оцінка значення медіани для перевірки статистичної значущості).

Результати, які повертає функція:

- `'med_estimate`: медіана вибірки;
- `'method`: використана процедура;

- 'hypotheses: статистичні гіпотези, які перевіряють (нульова H_0 і альтернативна H_1);
- 'statistic: кількість степенів свободи для перевірки нульової гіпотези;
- 'distribution: оцінка розподілу вибірки;
- 'p_value: імовірність помилкового вибору гіпотези h_1, якщо виконується H_0.

Приклад: перевірка гіпотези H_0 про рівність медіани вибірки 6, проти альтернативної гіпотези H_1: медіана більше 6.

```
(%i1) load("stats")$
(%i2) x:[2,0.1,7,1.8,4,2.3,5.6,7.4,5.1,6.1,6]$
(%i3) test_sign(x,'median=6,'alternative='greater);
```

$$
\text{(\%o3)}\quad \left(\begin{array}{c} SIGN\ \ TEST \\ med\_estimate = 5.1 \\ method = Non\ \ parametric\ \ sign\ \ test. \\ hypotheses = H0:\ \ median\ \ =\ \ 6\ \ ,\ \ H1:\ \ median\ \ >\ \ 6 \\ statistic = 7 \\ distribution = [binomial, 10, 0.5] \\ p\_value = 0.05468749999999989 \end{array}\right)
$$

Аналогічна функція – `test_signed_rank(x)` (або із зазначенням параметрів `test_signed_rank(x, option`$_1$`, option`$_2$`, ...)`), яка використовує тест правила знаків Вілкоксона для оцінки гіпотези про медіану неперервної вибірки. Параметри і результати функції `test_signed_rank` такі ж, як і для функції `test_sign`.

Приклад: перевірка гіпотези h_0: медіана дорівнює 15 проти альтернативної гіпотези H_1: медіана більше 15.

```
(%i1) load("stats")$
(%i2) x:[17.1,15.9,13.7,13.4,15.5,17.6]$
(%i3) test_signed_rank(x,median=15,alternative=greater);
```

$$
\text{(\%o3)}\quad \left(\begin{array}{c} SIGNED\ \ RANK\ \ TEST \\ med\_estimate = 15.7 \\ method = Exact\ \ test \\ hypotheses = H0:\ \ med\ \ =\ \ 15\ \ ,\ \ H1:\ \ med\ \ >\ \ 15 \\ statistic = 14 \\ distribution = [signed\_rank, 6] \\ p\_value = 0.28125 \end{array}\right)
$$

Непараметричне порівняння медіан двох вибірок реалізовано в одній функції – `test_rank_sum`. У цій функції використовується тест Вілкоксона-Мана-Уїтні. U-критерій Мана-Уїтні – непараметричний метод перевірки гіпотез, що часто використовується як альтернатива t-тесту Стьюдента. Зазвичай, цей тест використовується для порівняння медіан двох розподілів x_1 і x_2, що не є нормальними (відсутність нормальності не надає змогу застосувати t-тест).

Синтаксис виклику: `test_rank_sum(`x_1`,`x_2`)` або `test_rank_sum(`x_1`, `x_2`, `$option_1$`)`.

Функція допускає лише один параметр: `alternative` (аналогічний до параметра `test_means_difference`).

Результати, які повертає функція:

- 'method: використана процедура;
- 'hypotheses: статистичні гіпотези, які перевіряють (нульова H_0 і альтернативна H_1);
- 'statistic: число степенів свободи для перевірки нульової гіпотези;
- 'distribution: оцінка розподілу розподілу вибірки;
- 'p_value: імовірність помилкового вибору гіпотези H_1, якщо виконується H_0.

Приклад: перевірка, чи однакові медіани вибірок x_1 і x_2.

```
(%i1) load("stats")$
(%i2) x:[12,15,17,38,42,10,23,35,28]$
(%i3) y:[21,18,25,14,52,65,40,43]$
(%i4) test_rank_sum(x,y);
```

$$
\text{(\%o4)}\quad \left(\begin{array}{c} RANK\ \ SUM\ \ TEST \\ method = Exact\ \ test \\ hypotheses = H0:\ \ med1\ \ =\ \ med2\ \ ,\ \ H1:\ \ med1\ \ \#\ \ med2 \\ statistic = 22 \\ distribution = [rank\_sum, 9, 8] \\ p\_value = 0.1995886466474702 \end{array}\right)
$$

Для вибірок більшого об'єму розподіл вибірок є приблизно нормальним. Порівнюємо гіпотези H_0: медіана 1 = медіана 2 і H_1: медіана 1 < медіана 2.

```
(%i1) load("stats")$
(%i2) x:[39,42,35,13,10,23,15,20,17,27]$
(%i3) y:[20,52,66,19,41,32,44,25,14,39,43,35,19,56,27,15]$
(%i4) test_rank_sum(x,y,'alternative='less);
```

$$(\%o4)\quad \left(\begin{array}{c} RANK\ \ SUM\ \ TEST \\ method = Asymptotic\ \ test.\ \ Ties \\ hypotheses = H0:\ \ med1\ \ =\ \ med2\ \ ,\ \ H1:\ \ med1\ \ <\ \ med2 \\ statistic = 48.5 \\ distribution = [normal, 79.5, 18.95419580097078] \\ p\_value = 0.05096985666598441 \end{array}\right)$$

Перевірка нормальності розподілу здійснюється функцією `test_normality(x)`. У цій функції реалізований тест Шапіро-Уїлка. Вибірка x (список або одномірна матриця) повинна бути розміром не менш 2, але не більше 5000 елементів (інакше видається повідомлення про помилку). Функція повертає два значення: `statistic` – величина W-статистики і величина ймовірності p (якщо p більше прийнятого рівня значущості, нульова гіпотеза про нормальність розподілу вибірки x не відкидається). Статистика W характеризує близькість вибіркового розподілу до нормального (чим ближче W до 1, тим менше ймовірність помилково прийняти гіпотезу про нормальність розподілу).

Приклад: перевірка гіпотези щодо нормальності розподілу генеральної сукупності за заданою вибіркою.

```
(%i1) load("stats")$
(%i2) x:[12,15,17,38,42,10,23,35,28]$
(%i3) test_normality(x);
```

$$(\%o3)\quad \left(\begin{array}{c} SHAPIRO\ \ -\ \ WILK\ \ TEST \\ statistic = 0.9251055695162439 \\ p\_value = 0.4361763918860427 \end{array}\right)$$

6.2.4 Обчислення коефіцієнтів лінійної регресії

Коефіцієнти і оцінка статистичної значущості для найпростішої лінійної регресії можуть визначатися за допомогою функції `simple_linear_regression` з пакета `stats`. Функція обчислює коефіцієнти і параметри лінійної регресії $y = a_0 + a_1 \cdot x$ (тобто тільки найпростішої).

Синтаксис виклику: `simple_linear_regression(x)` або `simple_linear_regression(x,` $option_1$`)`.

Параметри функції `simple_linear_regression`: `conflevel` (рівень значущості, звичайно 0.95, див. вище) і `regressor` (типово x, назва незалежної змінної). Розглянута функція виводить велику кількість статистичних параметрів:

1. `'model`: отримане рівняння регресії;
2. `'means`: середнє;
3. `'variances`: дисперсії обох змінних;
4. `'correlation`: коефіцієнт кореляції;
5. `'adc`: коефіцієнт детермінації;
6. `'a_estimation`: оцінка параметра a;
7. `'a_conf_int`: довірчий інтервал для a;
8. `'b_estimation`: оцінка параметра b;
9. `'b_conf_int`: довірчий інтервал для b;
10. `'hypotheses`: нульова і альтернативна гіпотеза щодо параметра b;
11. `'statistic`: статистичні характеристики вибірки, використані для перевірки нульової гіпотези;
12. `'distribution`: розподіл вибірки;
13. `'p_value`: величина ймовірності для перевірки гіпотези про статистичну значущість b;
14. `'v_estimation`: оцінка залишкової дисперсії;
15. `'v_conf_int`: довірчий інтервал для залишкової дисперсії;
16. `'cond_mean_conf_int`: довірчий інтервал для середнього;
17. `'new_pred_conf_int`: довірчий інтервал для нового прогнозу;

18. `'residuals`: список, що містить залишки.

Типово на консоль виводяться тільки параметри 1, 4, 14, 10, 11, 12, і 13 у цьому списку. Інші параметри приховані, але доступ до них забезпечується за допомогою функцій `items_inference` або `take_inference`.

Задаємося вихідними даними

```
(%i9) s:[[125,140.7], [130,155.1], [135,160.3], [140,167.2], [145,169.8]]$
```

Обчислюємо коефіцієнти та інші параметри регресії

```
(%i10) z:simple_linear_regression(s,conflevel=0.99);
```

(%o10)
$$\left(\begin{array}{c} SIMPLE\ \ LINEAR\ \ REGRESSION \\ model = 1.405999999999985\,x - 31.18999999999804 \\ correlation = 0.9611685255255155 \\ v\_estimation = 13.57966666666665 \\ b\_conf\_int = [0.04469633662525307, 2.767303663374718] \\ hypotheses = H0:\ \ b\ \ =\ \ 0\ \ , H1:\ \ b\ \ \#\ \ 0 \\ statistic = 6.032686683658114 \\ distribution = [student\_t, 3] \\ p\_value = 0.009131954735741799 \end{array}\right)$$

```
(%i11) z:simple_linear_regression(s,conflevel=0.95);
```

(%o11)
$$\left(\begin{array}{c} SIMPLE\ \ LINEAR\ \ REGRESSION \\ model = 1.405999999999985\,x - 31.18999999999804 \\ correlation = 0.9611685255255155 \\ v\_estimation = 13.57966666666665 \\ b\_conf\_int = [0.6642874364502123, 2.147712563549759] \\ hypotheses = H0:\ \ b\ \ =\ \ 0\ \ , H1:\ \ b\ \ \#\ \ 0 \\ statistic = 6.032686683658114 \\ distribution = [student\_t, 3] \\ p\_value = 0.009131954735741799 \end{array}\right)$$

Деякі додаткові параметри:

```
(%i5) take_inference(model,z), x=133;
(%o5) 155.808
(%i6) take_inference(means,z);
(%o6) [135.0, 158.62]
(%i7) take_inference(new_pred_conf_int,z), x=133;
(%o7) [142.8757995613282, 168.7402004386718]
```

Графічна ілюстрація побудованої лінійної залежності наведена на рис. 6.6. Використана команда:

```
(%i11) plot2d([[discrete,s], take_inference(model,z)],
[x, 120,150], [style, [points], [lines]], [gnuplot_term,ps] , [gnuplot_out_file, "regress.eps"]);
```

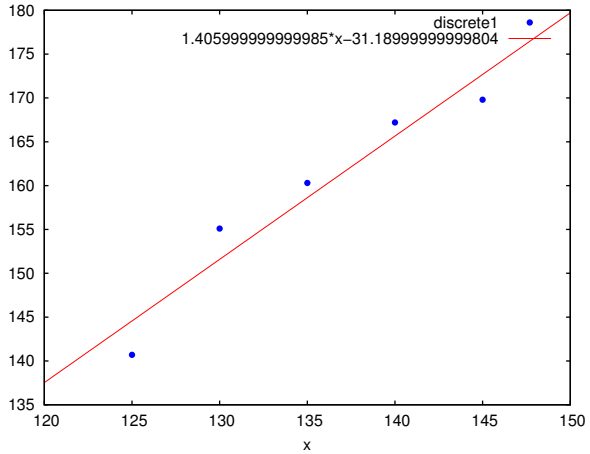


Рис. 6.6. Проста лінійна регресія.

6.2.5 Використання методу найменших квадратів

Система комп'ютерної алгебри **Maxima** містить потужний модуль для лінійного і нелінійного оцінювання параметрів різних моделей з використанням методу найменших квадратів – пакет `lsqares`.

Основна функція пакета `lsqares` – це функція `lsquares_estimates`.

Синтаксис виклику: `lsquares_estimates(`D`,`x`,`e`,`a`)` або `lsquares_estimates(`D`,`x`,`e`,`a`,initial=L,tol=t)`

Функцію призначено для оцінки параметрів, що найкраще відповідають рівнянню e у змінних a і a за набором даних D, які визначаються методом найменших квадратів. Функція `lsquares_estimates` спочатку намагається відшукати точний розв'язок, і якщо це не вдається, шукає приблизний розв'язок. Значення, яке повертає функція – список типу `[a=..., b=..., c=...]`. Елементи списку забезпечують мінімум середньоквадратичної похибки. Дані D повинні бути матрицею. Кожний рядок – один запис або один випадок, кожний стовпець відповідає значенням деякої змінної.

Список змінних x дає назву для кожного стовпця D (навіть для стовпців, які не входять в аналіз). Список параметрів містить назви параметрів, для яких відшукуються оцінки. Рівняння e є виразом або рівнянням у змінних x і a; якщо e записано не у формі рівняння, його буде розглянуто як рівняння $e = 0$. Якщо деякий точний розв'язок може бути знайдено за допомогою `solve`, дані D можуть містити і нечислові значення.

Додаткові аргументи `lsquares_estimates` визначені як рівняння і передаються «дослівно» функції `lbfgs`, що використовується, щоб знайти оцінки обчислювальним методом, коли точний результат знайти неможливо. Однак, якщо ніякого точного розв'язку не знайдено, у кожного елемента D повинне бути числове значення, у тому числі константи (зокрема `%pi` і `%e`) або числові літерали (цілі числа, раціональні, із рухомою крапкою, і із рухомою крапкою підвищеної точності). Обчислення виконуються зі звичайною арифметикою із рухомою крапкою, у такий спосіб всі інші види чисел перетворяться до значень із рухомою крапкою. Для роботи з `lsquares_estimates` слід завантажити цю функцію командою `load(lsquares)`.

Приклад (точний розв'язок):

```
(%i1) load(lsquares)$
(%i2) M:matrix([1,1,1],[3/2,1,2],[9/4,2,1],[3,2,2],[2,2,1]);
```

$$(\%o2)\quad \begin{pmatrix} 1 & 1 & 1 \\ \frac{3}{2} & 1 & 2 \\ \frac{9}{4} & 2 & 1 \\ 3 & 2 & 2 \\ 2 & 2 & 1 \end{pmatrix}$$

```
(%i3) lsquares_estimates(M,[z,x,y],(z+D)^2=A*x+B*y+C,[A,B,C,D]);
```

(%o3) $[[A = -\frac{59}{16}, B = -\frac{27}{16}, C = \frac{10921}{1024}, D = -\frac{107}{32}]]$

Інший приклад (точного розв'язку немає, відшукується наближений):

```
(%i1) load(lsquares)$ M:matrix([1, 1], [2, 7/4], [3, 11/4], [4, 13/4]);
```

$$(\%o2)\quad \begin{pmatrix} 1 & 1 \\ 2 & \frac{7}{4} \\ 3 & \frac{11}{4} \\ 4 & \frac{13}{4} \end{pmatrix}$$

```
(%i3) lsquares_estimates(M,[x,y],y=a*x^b+c,[a,b,c],initial=[3,3,3], iprint=[-1,0]);
```

(%o3) $[[a = 1.375751433061394, b = 0.7148891534417651, c = -0.4020908910062951]]$

Для обчислення нев'язок для рівняння e при підставлянні до нього даних, що зберігаються у матриці D, можна використати функцію `lsquares_residuals(`D`,`x`,`e`,`a`)` (зміст параметрів той же, що і для функції `lsquares_estimates`).

Приклад використання функції `lsquares_estimates` і `lsquares_residuals` (ті ж дані, що використано для розрахунку параметрів простої лінійної регресії):

```
(%i1) load(lsquares)$
(%i2) s:[[125,140.7], [130,155.1],[135,160.3], [140,167.2], [145,169.8]];
```

(%o2) $[[125, 140.7],\ [130, 155.1],\ [135, 160.3],\ [140, 167.2],\ [145, 169.8]]$

```
(%i3) D:apply(matrix,s);
```

$$(\%o3)\quad \begin{pmatrix} 125 & 140.7 \\ 130 & 155.1 \\ 135 & 160.3 \\ 140 & 167.2 \\ 145 & 169.8 \end{pmatrix}$$

```
(%i4) a:lsquares_estimates(D,[y,x],y = A+B*x, [A,B]);
```

(%o4) $[[A = \frac{8231525}{267474}, B = \frac{87875}{133737}]]$

```
(%i5) float(%);
```

(%o5) $[[A = 30.77504729431646, B = 0.6570732108541391]]$

```
(%i6) lsquares_residuals(D, [y,x], y=A+B*x, first(a));
```

(%o6) $[1.774751938506171,\ -2.687102297793416,\ -1.103882994234965,\ 0.63768814912851,\ 2.653921502650718]$

Інші функції, що входять до складу пакета `lsquares`, за синтаксисом використання та ідеї реалізації аналогічні наведеним (див. документацію для розробників).

6.3 Моделювання динамічних систем

Багато моделей, засновані на нелінійних диференціальних рівняннях, демонструють абсолютно дивовижні властивості, причому розв'язки більшості з цих рівнянь можна одержати лише у обчислювальний спосіб.

Моделі, засновані на задачах Коші для ЗДР, часто називають динамічними системами, підкреслюючи, що, як правило, вони містять похідні за часом t і описують динаміку деяких параметрів. Проблеми, пов'язані з динамічними системами, насправді достатньо різноманітні і найчастіше не зводяться до простого інтегрування ЗДР.

6.3.1 Моделювання системи хімічних реакцій

Розгляньмо приклад. Досліджуємо систему із трьох диференціальних рівнянь, що описують модель хімічної кінетики: $A \to B$

$B \to C$

Система відповідних диференціальних рівнянь $\frac{dc_A}{dt} = -k_1 c_A$

$\frac{dc_B}{dt} = k_1 c_A - k_2 c_B$

$\frac{dc_C}{dt} = k_2 c_B$

Початкові умови:

$c_A = 1, c_B = 0, c_C = 0.$

Результати розв'язання наведені на рис. 6.7.

```
(%i1) eq1:'diff(ca(t),t)=-k1*ca(t); eq2:'diff(cb(t),t)=k1*ca(t)-k2*cb(t); eq3:'diff(cc(t),t)=k2*cb(t);
```
(%o1) $\frac{d}{dt}\,\mathrm{ca}\,(t) = -k1\,\mathrm{ca}\,(t)$

(%o2) $\frac{d}{dt}\,\mathrm{cb}\,(t) = k1\,\mathrm{ca}\,(t) \;\; - k2\,\mathrm{cb}\,(t)$

(%o3) $\frac{d}{dt}\,\mathrm{cc}\,(t) \;\; = k2\,\mathrm{cb}\,(t)$

```
(%i4) atvalue(ca(t),t=0,1);
```
(%o4) 1

```
(%i5) atvalue(cb(t),t=0,0); atvalue(cc(t),t=0,0);
```
(%o5) 0

(%o6) 0

```
(%i7) sol:desolve([eq1,eq2,eq3],[ca(t),cb(t),cc(t)]);
```
(%o7) $[\mathrm{ca}\,(t) = e^{-k1\,t}, \mathrm{cb}\,(t) = \frac{k1\,e^{-k1\,t}}{k2-k1} - \frac{k1\,e^{-k2\,t}}{k2-k1}, \mathrm{cc}\,(t) = \frac{k1\,e^{-k2\,t}}{k2-k1} - \frac{k2\,e^{-k1\,t}}{k2-k1} + 1]$

```
(%i8) ratsimp(sol);
```
(%o8) $[\mathrm{ca}\,(t) = e^{-k1\,t}, \mathrm{cb}\,(t) = \frac{\left(k1\,e^{k2\,t} - k1\,e^{k1\,t}\right)\;\; e^{-k2\,t-k1\,t}}{k2-k1}, \mathrm{cc}\,(t) = \frac{\left(\left((k2-k1)\;\; e^{k1\,t} - k2\right)\;\; e^{k2\,t} + k1\,e^{k1\,t}\right)\;\; e^{-k2\,t-k1\,t}}{k2-k1}]$

```
(%i9) k1:0.1; k2:0.5; ev((sol));
```
(%o9) 0.1

(%o10) 0.5

(%o11) $[\mathrm{ca}\,(t) = e^{-0.1\,t}, \mathrm{cb}\,(t) = 0.25\,e^{-0.1\,t} - 0.25\,e^{-0.5\,t}, \mathrm{cc}\,(t) = -1.25\,e^{-0.1\,t} + 0.25\,e^{-0.5\,t} + 1]$

```
(%i12) plot2d([%e^(-0.1*t),0.25*%e^(-0.1*t)-0.25*%e^(-0.5*t), -1.25*%e^(-0.1*t)+0.25*%e^(-0.5*t)+1],
[t,0,50], [gnuplot_preamble,"set grid"], [gnuplot_term,"png size 500,500"],
[gnuplot_out_file,"chem.png"]);
```

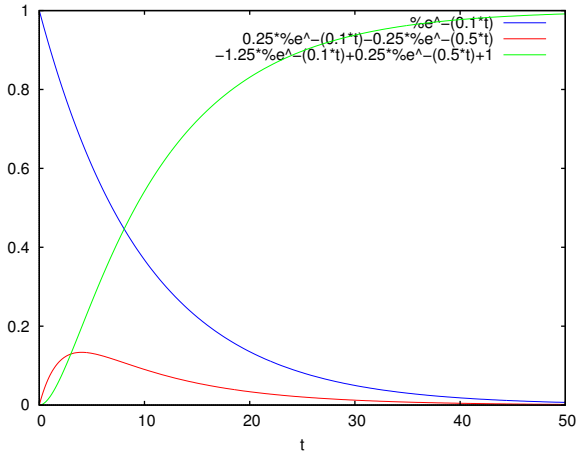


Рис. 6.7. Кінетика хімічних реакцій.

Трохи складніша задача – моделювання кінетики паралельно-послідовних реакцій, що протікають за такою схемою:

$A \to B$

$B + C \to A$

$B \to C$

Залежно від сталих швидкості хімічних реакцій ця система може бути досить жорсткою.

Приклад командного файла для розв'язання жорсткої системи ЗДР у **Maxima** (ця система нелінійна, тому використовуємо метод Рунґе-Кутти, однак розрахунки утрудняються жорсткістю системи):

```
load("dynamics");
load("draw");
k1:0.1; k2:100; k3:10;
eq1:-k1*ca+k3*cb*cc;
eq2:k1*ca-k3*cb*cc-k2*cb;
eq3:k2*cb;
t_range: [t,0,100,0.01];
sol: rk([eq1,eq2,eq3],[ca,cb,cc],[1,0,0],t_range)$
len:length(sol);
t:makelist(sol[k][1],k,1,len)$
ca:makelist(sol[k][2],k,1,len)$
cb:makelist(sol[k][3],k,1,len)$
cc:makelist(sol[k][4],k,1,len)$
draw2d(title="Chemical system",xlabel="ca",ylabel="cb",
    grid=true, points_joined=true, points(t,ca), points(t,cb), points(t,cc), terminal=eps);
```

Ця система досить важко розв'язується за допомогою функції `rk`. Збільшення констант до `k2=1000` і `k3=100` робить задачу практично нерозв'язної засобами пакета `dynamics`.

Найпростішим класичним прикладом існування автоколивань у системі хімічних реакцій є тримолекулярна модель «Брюсселятор», запропонована у Брюсселі Пригожиним і Лефевром (1965). Основною метою при вивченні цієї моделі було встановлення якісних типів поводження, сумісних з фундаментальними законами хімічної і біологічної кінетики. У цьому сенсі брюсселятор відіграє роль базової моделі, таку ж як гармонійний осцилятор у фізиці, або моделі Вольтерра в динаміці популяцій.

У межах цієї книги брюсселятор розглядається як приклад автоколивальної системи.

Опис моделі брюсселятора у **Maxima** наведено у наступному командному файлі:

```
load("dynamics");
load("draw");
B:0.5;
eq1:-(B+1)*y0+y0^2*y1+1;
eq2:B*y0-y0^2+1;
t_range:[t,0,10,0.1];
sol:rk([eq1,eq2], [y0,y1], [1,1], t_range)$
len:length(sol);
t:makelist(sol[k][1],k,1,len)$
y0:makelist(sol[k][2],k,1,len)$
y1:makelist(sol[k][3],k,1,len)$
draw2d(title="Brusselator",xlabel="t",ylabel="y0,y1",grid=true,points_joined = true,
   points(t,y0),points(t,y1),terminal=eps);
```

Графічну ілюстрацію (автоколивальний режим у системі) наведено на рис. 6.8, а фазові портрети – на рис. 6.9 і рис. 6.10.

При виконанні розрахунків варто звернути увагу на жорсткість системи ЗДР, що описує брюсселятор, зокрема, при `B=2.5` для побудови наведеної ілюстрації довелося зменшити крок за часом до 0.002. При черговому запуску командного файла, що містить команди завантаження пакетів, рекомендуємо перезапустити **Maxima**.

6.3.2 Фазові портрети динамічних систем

Для вивчення динамічних систем центральним моментом є аналіз фазових портретів, тобто розв'язків, які отримуються при виборі різноманітних початкових умов.

Розв'язування ЗДР часто зручніше зображувати не у вигляді графіка `y0(t)`, `y1(t)`, ..., а у фазовому просторі, за кожною з осей якого відкладаються значення кожної зі знайдених функцій. При такій побудові графіка аргумент `t` буде присутній на ньому лише параметрично.

Як правило, розв'язання задач Коші для ЗДР та систем ЗДР – задача добре розроблена і з обчислювальної точки зору досить проста. На практиці частіше зустрічаються інші, складніші задачі, зокрема, дослідження поведінки динамічної системи залежно від початкових умов. При цьому у більшості випадків буває необхідним вивчити тільки асимптотичний розв'язок ЗДР, тобто $y(t \to \infty)$, який називають атрактором. Дуже наочним чином можна візуалізувати таку інформацію на фазовій площині, багато в чому завдяки тому, що існує всього кілька типів атракторів, і для них можна побудувати чітку класифікацію.

З одного боку, кожен розв'язок буде виходити із точки, координати якої є початковими умовами, але, виявляється, для більшості ЗДР цілі сімейства траєкторій будуть закінчуватися у тих самих атракторах (стаціонарних точках або

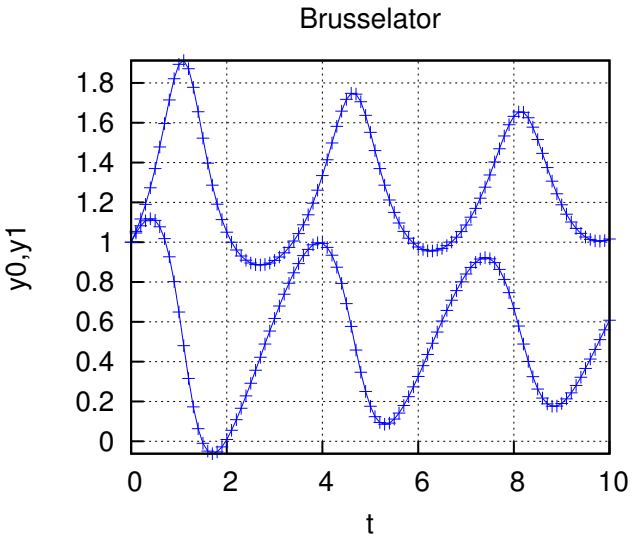


Рис. 6.8. Зміна концентрацій при моделюванні автоколивальної хімічної реакції (брюсселятора).

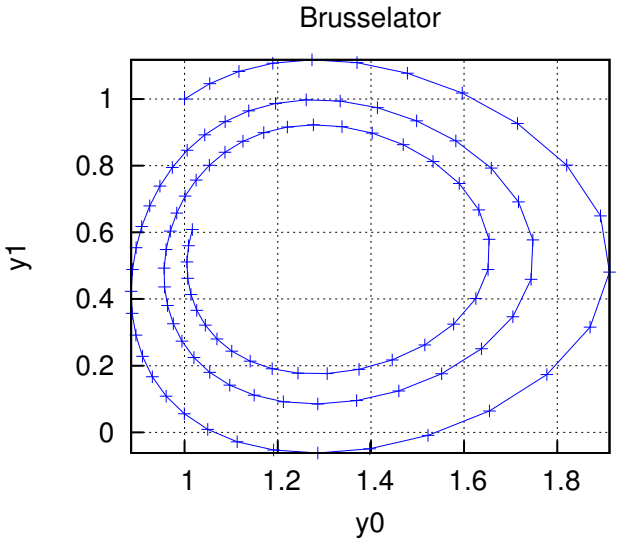


Рис. 6.9. Фазовий портрет для брюсселятора (`B=0.5`).

граничних циклах). Множина розв'язків, обчислена для всіляких початкових умов, утворить фазовий портрет динамічної системи. З обчислювальної точки зору задача дослідження фазового портрета часто зводиться до звичайного сканування сімейств розв'язків ЗДР при різних початкових умовах.

Подальше ускладнення задач аналізу фазових портретів пов'язане з їхньою залежністю від параметрів, що входять до системи ЗДР. Зокрема, при плавній зміні параметра моделі може мінятися розташування атракторів на фазовій площині, а також можуть виникати нові атрактори і припиняти своє існування старі. У першому випадку, за відсутності особливостей, буде відбуватися просте пересування атракторів фазовою площиною (без зміни їхніх типів і кількості), а у другому — фазовий портрет динамічної системи буде докорінно перебудовуватися. Критичне сполучення параметрів, при яких фазовий портрет системи якісно міняється, називається у теорії динамічних систем точкою біфуркації.

Розгляньмо декілька найбільш відомих класичних прикладів динамічних систем. Це моделі динаміки популяцій (Лотка-Вольтерра), генератора автоколивань (Ван дер Поля), турбулентної конвекції (Лоренца) і хімічної реакції з дифузією (Пригожина). Для вивчення динамічних систем розроблено спеціальну теорію, центральним моментом якої є аналіз фазових портретів, тобто розв'язків, які отримують при виборі усіх можливих початкових умов.

6.3.3 Модель динаміки популяцій

Модель взаємодії «хижак-жертва» незалежно запропонували у 1925–1927 рр. Лотка і Вольтерра. Два диференціальних рівняння моделюють тимчасову динаміку чисельності двох біологічних популяцій жертв x і хижаків y. Вважається, що жертви розмножуються зі сталою швидкістю, а їхня чисельність спадає внаслідок поїдання хижаками. Хижаки ж розмножуються зі швидкістю, пропорційною кількості їжі, і вмирають у природний спосіб.

Модель була створена для біологічних систем, але із певними виправленнями застосовна до конкуренції фірм,

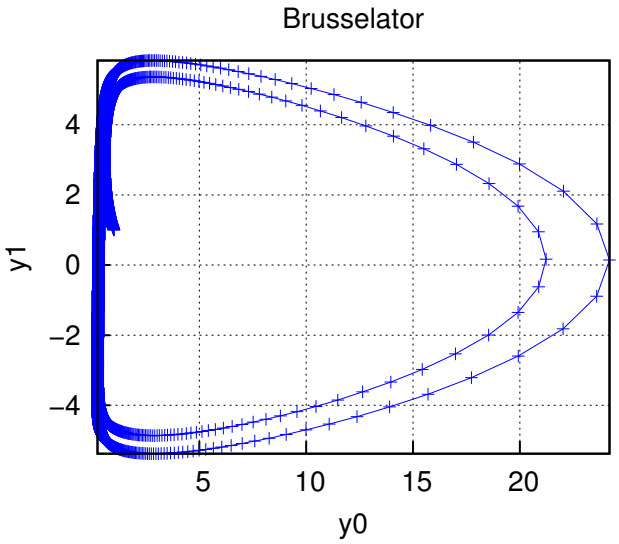


Рис. 6.10. Фазовий портрет для брюсселятора (`B=2.5`).

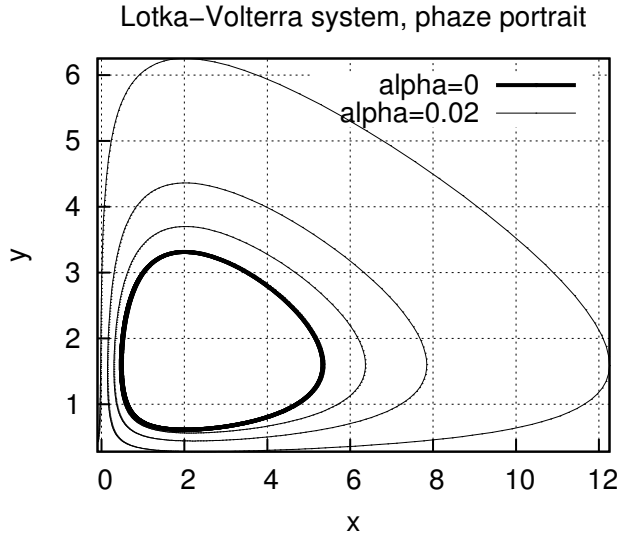


Рис. 6.11. Фазовий портрет для системи Лотка-Вольтерра.

будівництва фінансових пірамід, росту народонаселення, екологічної проблематики тощо.

Ця модель Вольтерра-Лотка з логістичним виправленням описується системою рівнянь

$$\frac{dy(t)}{dt} = y(t)\,(a - bz(t)) - \alpha y^2(t);$$
$$\frac{dz(t)}{dt} = (-c + dy(t)) - \alpha z^2(t);$$

з умовами заданої чисельності «жертв» і «хижаків» у початковий момент $t = 0$.

Розв'язуючи цю задачу при різних початкових значеннях, одержуємо різні фазові портрети (звичайний коливальний процес і поступова загибель популяцій). Результати наведено на рисунках 6.11, 6.12 і 6.13.

```
(%i1) a:4$ b:2.5$ c:2$ d:1$ alpha=0$
eq1:'diff(y(t),t)=(a-b*z(t))*y(t)-alpha*y(t)^2;
eq2:'diff(z(t),t)=(-c+d*y(t))*z(t)-alpha*y(t)^2;
atvalue(y(t),t=0,3); atvalue(z(t),t=0,1);
```

(%o6) $\frac{d}{dt}\,\mathrm{y}\,(t) = \mathrm{y}\,(t)\,(4 - 2.5\,\mathrm{z}\,(t)) - \alpha\,\mathrm{y}\,(t)^{\,2}$

(%o7) $\frac{d}{dt}\,\mathrm{z}\,(t) = (\mathrm{y}\,(t) - 2)\,\mathrm{z}\,(t) - \alpha\,\mathrm{y}\,(t)^{\,2}$

(%o8) 3

(%o9) 1

```
(%i10) desolve( [eq1,eq2] ,[y(t),z(t)]);
rat: replaced-2.5 by-5/2=-2.5
```

```
rat: replaced-2.5 by-5/2=-2.5
rat: replaced-2.5 by-5/2=-2.5
rat: replaced 2.5 by 5/2= 2.5
(%o10)
```

$$[\mathrm{y}(t) = \mathrm{ilt}\left(-\frac{5\,\mathrm{laplace}\left(\mathrm{y}(t)\ \mathrm{z}(t), t, g17456\right) + 2\,\alpha\,\mathrm{laplace}\left(\mathrm{y}(t)^2, t, g17456\right) - 6}{2\,g17456 - 8}, g17456, t\right),$$

$$\mathrm{z}(t) = \mathrm{ilt}\left(\frac{\mathrm{laplace}\left(\mathrm{y}(t)\ \mathrm{z}(t), t, g17456\right) - \alpha\,\mathrm{laplace}\left(\mathrm{y}(t)^2, t, g17456\right) + 1}{g17456 + 2}, g17456, t\right)]$$

Очевидна проблема – нерозв'язність даної системи у явному вигляді методом перетворення Лапласа, оскільки вона нелінійна.

Використаємо обчислювальний метод Рунґе-Кутти з пакета `dynamics`. Результати розв'язання для значень $\alpha = 0$ і $\alpha = 0.02$ представлені на рис. 6.12 і 6.13.

Розгляньмо командний файл для задачі моделювання системи Лотка-Вольтерра у **Maxima**:

```
a:4; b:2.5; c:2; d:1; alpha1:0;
ode1:(a-b*x)*y-alpha1*x^2$ ode2:(-c+d*y)*x-alpha1*y^2$
alpha2:0.02;
ode3:(a-b*x)*y-alpha2*x^2$ ode4:(-c+d*y)*x-alpha2*y^2$
load("dynamics");
t1:[]$ xg1:[]$ yg1:[]$ t2:[]$ xg2:[]$ yg2:[]$
l1:rk([ode1,ode2],[y,x],[1,3],[t,0,9,0.01])$
l2:rk([ode3,ode4],[y,x],[1,3],[t,0,9,0.01])$
for i:1 thru length(l1) do(t1:append(t1,[l1[i][1]]),
    xg1:append(xg1, [l1[i][2]]),yg1:append(yg1, [l1[i][3]]));
for i:1 thru length(l2) do(t2:append(t2,[l2[i][1]]),
    xg2:append(xg2, [l2[i][2]]),yg2:append(yg2, [l2[i][3]]));
load("draw");
draw2d(terminal='eps, file_name="lotka1",grid=true,xlabel = "x",
    ylabel = "y", title="Lotka-Volterra system, phaze portrait",
    key= "alpha=0",points_joined = true, point_type = none,
    line_width = 4,color = black, points(xg1,yg1),
    points_joined = true, color = black,point_type = none,
    line_width = 1,key="alpha=0.02", points(xg2,yg2))$
draw2d(terminal='eps, file_name="lotka2",grid=true,xlabel = "t",
    ylabel = "x,y", title="Lotka-Volterra system, alpha=0",
    key= "x(t)",points_joined = true, line_width = 1,
    color = black,point_type = none, points(t1,xg1),
    points_joined = true, line_width = 4, point_type = none,
    color = black, key= "y(t)", points(t1,yg1))$
draw2d(terminal='eps, file_name="lotka3",grid=true,xlabel = "t",
    ylabel = "x,y", title="Lotka-Volterra system, alpha=0.02",
    key= "x(t)",points_joined = true, point_type = none,
    line_width = 1, color = black, points(t2,xg2),
    points_joined = true, line_width = 4, point_type = none,
    color = black, key= "y(t)", points(t2,yg2))$
```

Диференціальні рівняння формуються символьними виразами, що визначають праві частини. Порядок слідування виразів для розрахунку правих частин ЗДР у першому списку функції `rk` має бути відповідним. Варто зауважити, що результат виконання функції `rk` — дворівневий список (кожен елемент списків `11` і `12` — також список, що містить значення незалежної змінної і відповідні значення шуканих функцій), тому його перетворюють на вектори, які використовуються для побудови графічних ілюстрацій.

6.3.4 Рух твердого тіла

Розгляньмо приклад побудови тривимірного фазового портрета. Знаходимо розв'язок задачі Ойлера щодо вільного руху твердого тіла:

$$\frac{dx_1}{dt} = x_2 x_3;$$
$$\frac{dx_2}{dt} = -x_1 x_3;$$
$$\frac{dx_3}{dt} = -0.51 x_1 x_3.$$

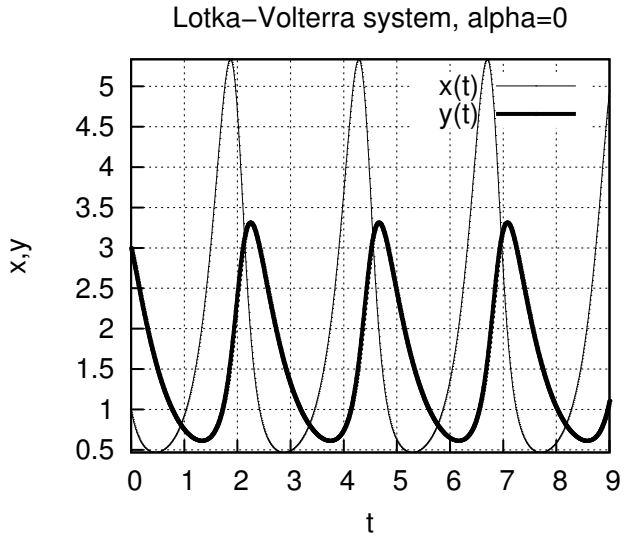


Рис. 6.12. Розв'язування системи Лотка-Вольтерра залежно від часу ($\alpha = 0$).

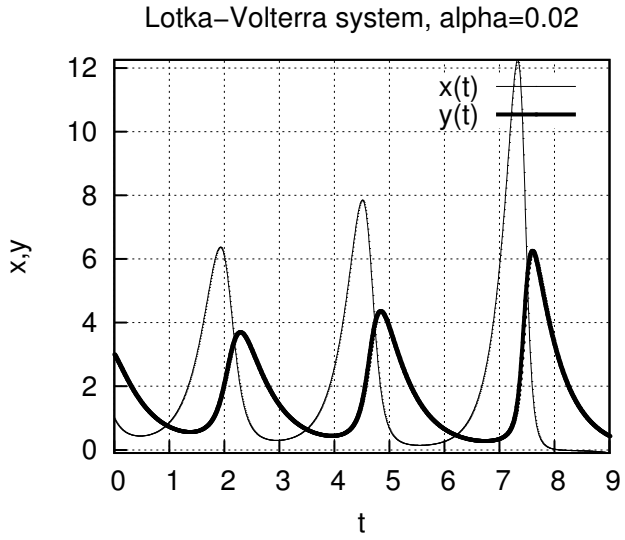


Рис. 6.13. Розв'язування системи Лотка-Вольтерра залежно від часу ($\alpha = 0,02$).

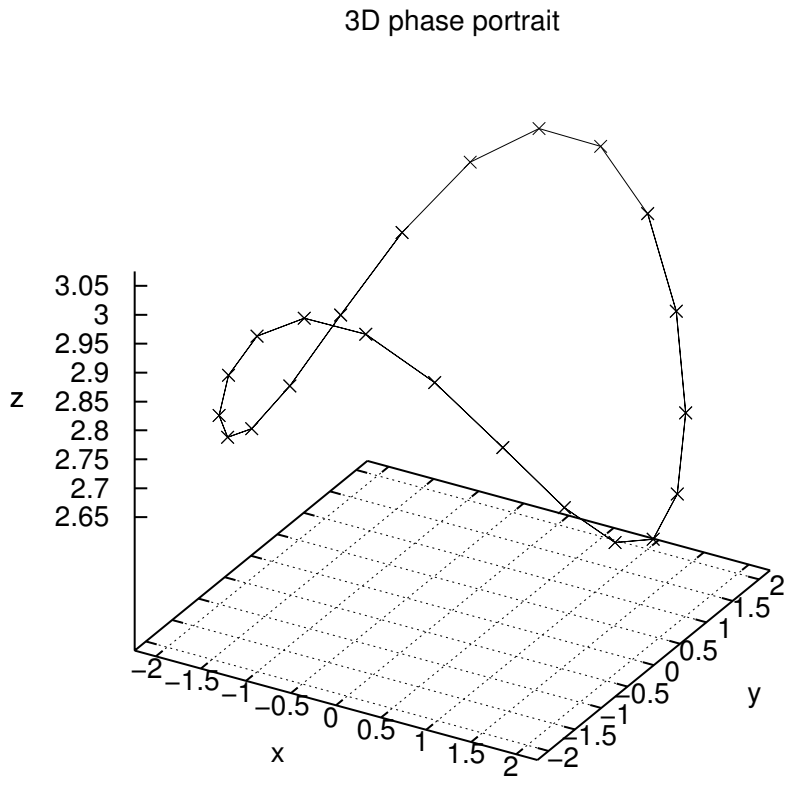


Рис. 6.14. Фазовий портрет тривимірної динамічної системи.

```
(%i1) eq1:'diff(x1(t),t)=x2(t)*x3(t);
eq2:'diff(x2(t),t)=-x1(t)*x3(t);
eq3:'diff(x3(t),t)=-0.51*x1(t)*x2(t);
```
(%o1) $\frac{d}{dt}\,\mathrm{x1}\,(t) = \mathrm{x2}\,(t)\;\mathrm{x3}\,(t)$
(%o2) $\frac{d}{dt}\,\mathrm{x2}\,(t) = -\mathrm{x1}\,(t)\;\mathrm{x3}\,(t)$
(%o3) $\frac{d}{dt}\,\mathrm{x3}\,(t) = -0.51\,\mathrm{x1}\,(t)\;\mathrm{x2}\,(\ t)$
```
(%i4) load("dynamics")$ l:rk([y*z, -x*z, -0.51*x*y], [x,y,z], [1,2,3], [t,0,4,0.1])$
```

Фазовий портрет для цієї динамічної системи (тривимірна крива) представлений на рис. 6.14.

6.3.5 Атрактор Лоренца

Одну із самих знаменитих динамічних систем було запропоновано у 1963 р. Лоренцем як спрощену модель конвективних турбулентних рухів рідини у посудині тороїдальної форми, яку нагрівають. Система складається із трьох ЗДР і має три параметри моделі. Задаємо праві частини рівнянь моделі Лоренца:

```
(%i1) eq:[s*(y-x), x*(r-z)-y, x*y-b*z];
```
(%o1) $[s\ (\ y-x)\ \ ,x\ (\ r-z)\ \ -y, x\,y-b\,z]$

Задаємо тимчасові параметри розв'язку:

```
(%i2) t_range:[t,0,50,0.05];
```
(%o2) $[t, 0, 50, 0.05]$

Задаємо параметри системи:

```
(%i3) s:10.0; r:28.0; b:2.6667;
```
(%o3) 10.0
(%o4) 28.0
(%o5) 2.6667

Задаємо початкові значення x, y, z:

```
(%i6) init: [1.0,0,0];
```
(%o6) $[1.0, 0, 0]$

Виконуємо розв'язання:

```
(%i7) sol:rk(eq,[x,y,z],init,t_range)$
(%i8) len:length(sol);
(%o8) 1001
```

Виокремлюємо компоненти розв'язку і будуємо графічні ілюстрації

```
(%i10) t:makelist(sol[k][1],k,1,len)$
x:makelist(sol[k][2],k,1,len)$
y:makelist(sol[k][3],k,1,len)$
z:makelist(sol[k][4],k,1,len)$
plot2d([discrete,t,x])$ plot2d([discrete,t,y])$
```

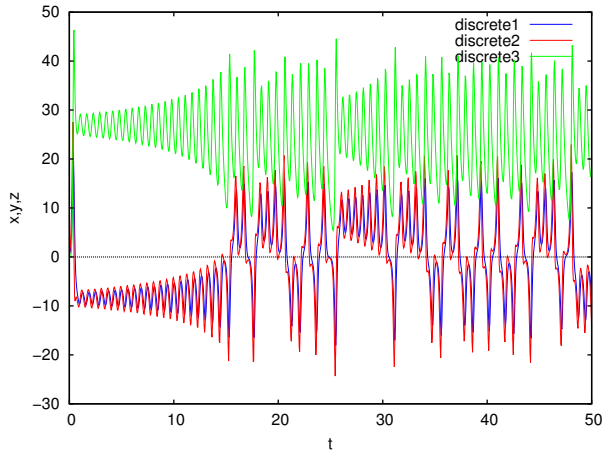


Рис. 6.15. Приклад формування динамічного хаосу (атрактор Лоренца).

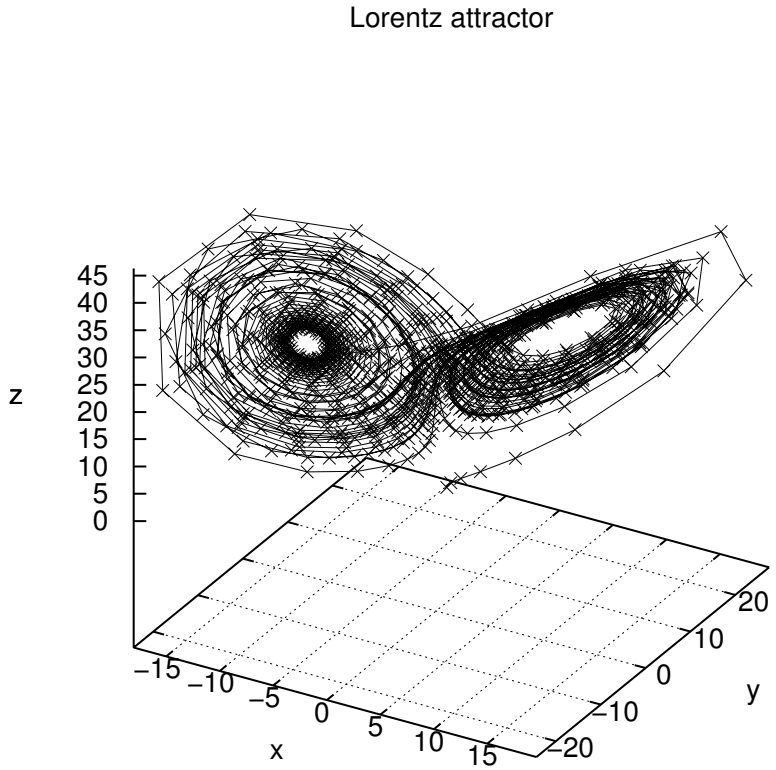


Рис. 6.16. Тривимірний фазовий портрет (атрактор Лоренца).

Результати розв'язання (хаотичні коливання x, y, z) представлені на рис. 6.15 і 6.16 (фазовий портрет системи). На рисунках об'єднані в одних осях криві $x(t)$, $y(t)$, $z(t)$.

Розв'язком системи Лоренца при певному сполученні параметрів є дивний атрактор (або атрактор Лоренца) – множина траєкторій на фазовому просторі, що притягують точку системи, за виглядом ідентична випадковому процесу. У деякому сенсі атрактор Лоренца є стохастичними автоколиваннями, які підтримуються в динамічній системі за рахунок зовнішнього джерела.

Розв'язок у вигляді дивного атрактора з'являється тільки при деяких сполученнях параметрів. Перебудова типу фазового портрета відбувається в області проміжних значень параметра r. Критичне сполучення параметрів, при яких фазовий портрет системи якісно міняється, називається у теорії динамічних систем точкою біфуркації. Фізичний зміст біфуркації у моделі Лоренца, відповідно до сучасних уявлень, описує перехід ламінарного руху рідини до турбулентного.

6.3.6 Модель автоколивальної системи: рівняння Ван дер Поля

Розгляньмо розв'язання рівняння Ван дер Поля, що описує електричні коливання е замкнутому контурі, що складається із з'єднаних послідовно конденсатора, індуктивності, нелінійного опору і елементів, що забезпечують підкачування енергії ззовні. Невідома функція часу $y(t)$ відповідає електричному струму, а у параметрі μ закладені кількісні співвідношення між складовими електричного ланцюга, зокрема і нелінійного компонента опору:

$$\frac{d^2y(t)}{dx^2} - \mu(1 - y^2(t))\frac{dy(t)}{dt} + y(t) = 0.$$

Розв'язком рівняння Ван дер Поля є коливання, вигляд яких для $\mu = 1$ показаний на рис. 6.17. Вони називаються автоколиваннями і принципово відрізняються від розглянутих раніше (наприклад, чисельності популяцій у моделі Вольтерра) тим, що їхні характеристики (амплітуда, частота, спектр) не залежать від початкових умов, а визначаються виключно властивостями самої динамічної системи. Через якийсь час розрахунків після виходу з початкової точки розв'язок виходить на той самий цикл коливань, називаний граничним циклом. Атрактор типу граничного циклу є замкнутою кривою на фазовій площині. До нього асимптотично притягаються усі навколишні траєкторії, що виходять із різних початкових точок, як зсередини (рис. 6.18), так і зовні граничного циклу.

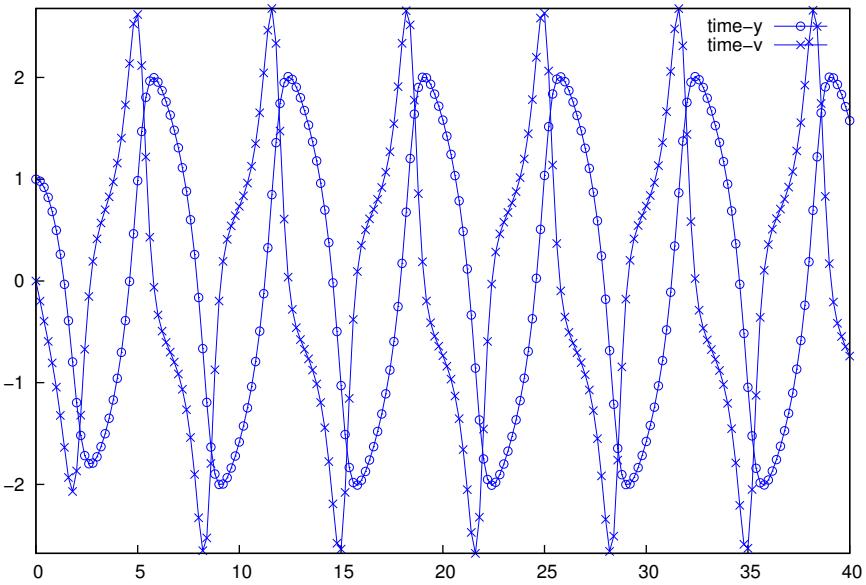


Рис. 6.17. Розв'язок рівняння Ван дер Поля.

Використаний командний файл **Maxima** (для побудови графічної ілюстрації використаний пакет `draw`):

```
load("dynamics")$ load("draw")$
mu:1$ s:rk([v,mu*(1-y^2)*v-y],[y,v],[1,0],[t,0,40,0.2])$
time:makelist(s[k][1],k,1,length(s))$
y:makelist(s[k][2],k,1,length(s))$
v:makelist(s[k][3],k,1,length(s))$
draw2d(points_joined=true, point_type=6, key="y-v",
    xlabel="y",ylabel="v",points(y,v),terminal=eps)$
```

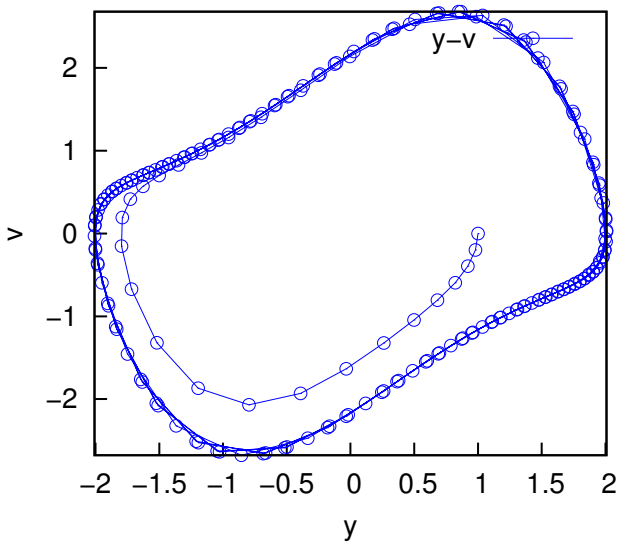


Рис. 6.18. Фазовий портрет рівняння Ван дер Поля.

Розділ 7

Розв'язування фізичних і математичних задач із Maxima

Доступна література і мережа Інтернет у формі «електронного помічника» студентів і школярів звичайно пропонує пакет **MathCad**, зрідка – **Maple** або **Mathematica**. Матеріал цього розділу містить ряд різнорідних задач, які вирішувалися різними авторами вручну або за допомогою **MathCad**.

7.1 Операції з поліномами і раціональними функціями

Розгляньмо розв'язання за допомогою **Maxima** декількох задач із класичного збірника за редакцією М.І. Сканаві. В **Maxima** «покрокове» спрощення виразів із послідовним використанням стандартного набору примітивів (формул суми або різниці кубів, формул піднесення суми або різниці до степеня тощо) виконати складно, тому результат є фактично довідковим, на який варто орієнтуватися при рішенні вручну, за допомогою ручки і паперу.

7.1.1 Спрощення алгебраїчних виразів

Приклад:
Спростити вираз і обчислити його, якщо дані числові значення параметрів:

```
(%i1) g:(1/a-1/(b+c))/(1/a+1/(b+c))*(1+(b^2+c^2-a^2)/2/b/c)/((a-b-c)/a/b/c);
```

$$(\%o1)\quad \frac{a\,b\,c\left(\frac{c^2+b^2-a^2}{2\,b\,c}+1\right)\left(\frac{1}{a}-\frac{1}{c+b}\right)}{(-c-b+a)\left(\frac{1}{c+b}+\frac{1}{a}\right)}$$

```
(%i2) ratsimp(%);
```

$$(\%o2)\quad -\frac{a\,c+a\,b-a^2}{2}$$

```
(%i3) %,a=0.02,b=-11.05,c=1.07;
(%o3) 0.1
```

Приклад: Спростити вираз і обчислити його, якщо дані числові значення параметрів:

```
(%i1) (sqrt(x)+1)/(x*sqrt(x)+x+sqrt(x))/(1/(x^2-sqrt(x)));
```

$$(\%o1)\quad \frac{(\sqrt{x}+1)\left(x^2-\sqrt{x}\right)}{x^{\frac{3}{2}}+x+\sqrt{x}}$$

```
(%i2) ratsimp(%) ;
(%o2) x-1
```

Приклад: Зробити зазначену підставляння і результат спростити:

```
(%i3) expr:(x^3-a^(-2/3)*b^(-1)*(a^2+b^2)*x+b^(1/2))/(b^(3/2)*x^2);
```

$$(\%o3)\quad \frac{x^3-\frac{\left(b^2+a^2\right)\,x}{a^{\frac{2}{3}}\,b}+\sqrt{b}}{b^{\frac{3}{2}}\,x^2}$$

```
(%i4) ratsimp(%);
```

$$(\%o4)\quad \frac{a^{\frac{2}{3}}\,b\,x^3+\left(-b^2-a^2\right)\,x+a^{\frac{2}{3}}\,b^{\frac{3}{2}}}{a^{\frac{2}{3}}\,b^{\frac{5}{2}}\,x^2}$$

```
(%i5) radcan(%);
```

$$(\%o5)\quad \frac{a^{\frac{2}{3}}\,b\,x^3+\left(-b^2-a^2\right)\,x+a^{\frac{2}{3}}\,b^{\frac{3}{2}}}{a^{\frac{2}{3}}\,b^{\frac{5}{2}}\,x^2}$$

Без зазначеного підставляння спрощення за допомогою комбінації функцій ratsimp і `radcan` не вдається.

```
(%i6) %,x=a^(2/3)*b^(-1/2);
```

$$(\%o6)\ \frac{\frac{a^{\frac{2}{3}}\left(-b^2-a^2\right)}{\sqrt{b}}+a^{\frac{2}{3}}\,b^{\frac{3}{2}}+\frac{a^{\frac{8}{3}}}{\sqrt{b}}}{a^2\,b^{\frac{3}{2}}}$$

```
(%i7) ratsimp(%) ;
```

Кінцевий результат виявляється простим `(%o7)` 0.

7.1.2 Розкладання поліномів і раціональних виразів на множники

Приклад: Розкласти на множники:

```
(%i1) expr:a^4+4*b^4;
```

$$(\%o1)\ 4\,b^4+a^4$$

```
(%i2) factor(%);
```

$$(\%o2)\ \left(2\,b^2-2\,a\,b+a^2\right)\,\left(2\,b^2+2\,a\,b+a^2\right)$$

7.1.3 Розв'язування алгебраїчних рівнянь

Maxima (як і будь-який інший пакет символьної математики) не завжди здатна одержати остаточний розв'язок. Однак отриманий результат може виявитися все-таки простіше, ніж початкова задача.

Приклад (також зі збірника за ред. М.І. Сканаві): Розв'язати рівняння $\sqrt{x-2}=x-4$:

```
(%i1) solve([sqrt(x-2)=x-4], [x]);
```

$$(\%o1)\ [x=\sqrt{x-2}+4]$$

Рівняння має один розв'язок: $x=6$, однак для відшукання його за допомогою **Maxima** доведеться вдатися до заміни початкового рівняння його наслідком:

```
(%i3) solve([(x-2)=(x-4)^2],[x]);
```

$$(\%o3)\ [x=6,x=3]$$

Розв'язки для подальшого використання можна отримати зі списку за допомогою функції `ev`:

```
(%i1) sol:solve([(x-2)=(x-4)^2],[x]);
```

$$(\%o3)\ [x=6,x=3]$$

```
(%i2) ev(x,sol[1]);
```

$$(\%o2)\ 6$$

```
(%i3) ev(x,sol[2]);
```

$$(\%o3)\ 3$$

Ще два приклади розв'язання алгебраїчних рівнянь:

```
(%i1) eq:7*(x+1/x)-2*(x^2+1/x^2)=9;
```

$$(\%o1)\ 7\,\left(x+\frac{1}{x}\right)-2\,\left(x^2+\frac{1}{x^2}\right)=9$$

```
(%i2) sol:solve([eq],[x]);
```

$$(\%o2)\ [x=2,x=\frac{1}{2},x=-\frac{\sqrt{3}\,i-1}{2},x=\frac{\sqrt{3}\,i+1}{2}]$$

```
(%i3) x1:ev(x,sol[1]); x2:ev(x,sol[2]);
  /*комплексні корені не розглядаємо*/
```

$$(\%o4)\ 2$$

$$(\%o5)\ \frac{1}{2}$$

Рівняння з радикалами перед розв'язуванням у **Maxima** доводиться перетворювати до степеневої форми (для виділення лівої і правої частини вираз використовують функції `lhs` і `rhs` відповідно):

```
(%i1) eq:sqrt(x+1)+sqrt(4*x+13)=sqrt(3*x+12);
```

$$(\%o1)\ \sqrt{4\,x+13}+\sqrt{x+1}=\sqrt{3\,x+12}$$

```
(%i2) eq1:lhs(eq)^2=rhs(eq)^2;
```

$$(\%o2)\ \left(\sqrt{4\,x+13}+\sqrt{x+1}\right)^2=3\,x+12$$

```
(%i3) solve([eq1],[x]);
```

$$(\%o3)\ [x=-\sqrt{x+1}\,\sqrt{4\,x+13}-1]$$

```
(%i4) eq2:x+1=rhs(%[1])+1;
```

$$(\%o4)\ x+1=-\sqrt{x+1}\,\sqrt{4\,x+13}$$

```
(%i5) eq3:lhs(eq2)^2=rhs(eq2)^2;
```

$$(\%o5)\ (x+1)^2=(x+1)\,(4\,x+13)$$

Остання команда дозволила одержати степеневе рівняння, розв'язне аналітично у **Maxima** (для цього і треба було двічі піднести до квадрата початкове рівняння).

```
(%i6) solve([eq3],[x]);
```
(%o6) $[x=-4, x=-1]$

Перевірку розв'язання виконуємо за допомогою функції `ev`.

Розв'язок $x=-4$ не задовольняє початковому рівнянню.

```
(%i7) ev(eq,%[1]);
```
(%o7) $2\sqrt{3}\,i=0$

Розв'язок $x=-1$ перетворює початкове рівняння на тотожність:

```
(%i8) ev(eq,%o6[2]);
```
(%o8) $3=3$

Розгляньмо ще один приклад, що ілюструє заміну і підстановку при розв'язанні алгебраїчних рівнянь:

```
(%i1) eq:sqrt(x+3-4*sqrt(x-1))+sqrt(x+8-6*sqrt(x-1))=1;
```

Вихідне рівняння:

(%o1) $\sqrt{x-4\sqrt{x-1}+3}+\sqrt{x-6\sqrt{x-1}+8}=1$

Виконаємо заміну $\sqrt{x+1}=z$, $z=x^2+1$:

```
(%i2) eq1:subst(z,sqrt(x-1),eq);
```
(%o2) $\sqrt{-4\,z+x+3}+\sqrt{-6\,z+x+8}=1$
```
(%i3) eq2:subst(z^2+1,x,eq1);
```
(%o3) $\sqrt{z^2-4\,z+4}+\sqrt{z^2-6\,z+9}=1$

Спрощуємо отриманий результат:

```
(%i4) radcan(%);
```
(%o4) $2\,z-5=1$
```
(%i5) solve([%],z);
```
(%o5) $[z=3]$
```
(%i6) solve([sqrt(x-1)=3],[x]);
```
(%o6) $[x=10]$

Виконаємо перевірку:

```
(%i7) ev(eq,%[1]);
```
(%o7) $1=1$

Значна частина тригонометричних рівнянь шкільного курсу також розв'язні у **Maxima**, але безпосередній розв'язок вдається одержати далеко не завжди.

Приклади:

```
(%i1) solve([sin(%pi/6-x)=sqrt(3)/2],[x]);
solve: using arc-trig functions to get a solution.
Some solutions will be lost.
```
(%o1) $[x=\frac{\pi}{6}]$

Більшість тригонометричних рівнянь у **Maxima** (крім найпростіших) доводиться розв'язувати зведенням їх до алгебраїчного.

Логарифмічні і показникові рівняння також розв'язуються у **Maxima** шляхом заміни змінних і зведення до алгебраїчного (див. вище специфічні функції для спрощення логарифмічних виразів).

7.2 Деякі фізичні задачі

Застосування систем символьної математики у викладанні фізики і хімії надає змогу зосередитися на змістовній частині матеріалу, що викладається. Крім того, учні отримують можливість розв'язувати набагато складніші задачі, ніж при розрахунках вручну. Наявність у **Maxima** чітко вираженої алгоритмічної мови (на відміну від **Matcad**) істотно знижує ризик підміни понять, коли пробіли власного підходу до розв'язання задачі учні підмінюють наявністю помилок і неточностей у програмному забезпеченні.

Ідеї розглянутих задач узяті з відомих підручників з використання **MathCad**, однак, на думку автора, використання **Maxima** може бути не менш, а в багатьох випадках і більш ефективним.

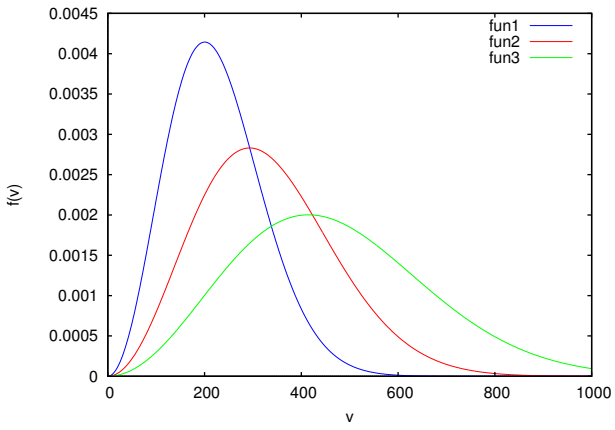


Рис. 7.1. Розподіл Максвела за швидкостями молекул повітря для різних температур.

7.2.1 Обчислення середньої квадратичної швидкості молекул

Вираз, що містять змінні, по суті може використовуватися як функція користувача у **Maxima**.

Розгляньмо можливість обчислення середньоквадратичної швидкості молекул для різних газів. Використовувана формула: $v = \sqrt{\dfrac{3RT}{M}}$, де $R = 8.314$ Дж/(моль·К), T – абсолютна температура, M – молярна маса.

Обчислимо середньоквадратичну швидкість молекул CO_2 ($M = 0.044$ кг/моль) при температурі 273 К:

```
(%i1) v:sqrt(3*R*T/M);
```
$$(\%o1)\quad \sqrt{3}\sqrt{\frac{R\,T}{M}}$$
```
(%i2) vCO2:float(v),M=0.044,T=273,R=8.314;
(%o2) 393.3875604633079
```

Розрахунок для декількох різних газів нескладно виконати, варіюючи молярну масу:

```
(%i3) vPov:float(v),M=0.029,T=273,R=8.314;
(%o3) 484.5604478145187
(%i4) v2:float(v),M=0.002,T=273,R=8.314;
(%o4) 1845.151213315592
```

7.2.2 Розподіл Максвела

Аналогічно попередньому розрахунку створимо вираз, що описує розподіл Максвела (див. блок команд **Maxima** нижче).

```
(%i1) fun:4*%pi*(M/2/%pi/R/T)^(3/2)*exp(-M*v^2/2/R/T)*v^2;
```
$$(\%o1)\quad \frac{3\sqrt{2}\,e^{-\frac{3}{2}}\,R\left(\frac{M}{R\,T}\right)^{\frac{3}{2}}T}{\sqrt{\pi}\,M}$$

Для аналізу отриманих виразів до формули розподілу Максвела підставляємо тільки температуру:

```
(%i2) fun70:fun,T=70;
```
$$(\%o2)\quad \frac{105\,2^{\frac{3}{2}}\,e^{-\frac{3}{2}}\left(\frac{M}{R}\right)^{\frac{3}{2}}R}{70^{\frac{3}{2}}\sqrt{\pi}\,M}$$
```
(%i3) fun150:fun,T=150;
```
$$(\%o3)\quad \frac{9\,2^{\frac{3}{2}}\,e^{-\frac{3}{2}}\left(\frac{M}{R}\right)^{\frac{3}{2}}R}{5\,6^{\frac{3}{2}}\sqrt{\pi}\,M}$$
```
(%i4) fun300:fun,T=300;
```
$$(\%o4)\quad \frac{\sqrt{3}\,e^{-\frac{3}{2}}\left(\frac{M}{R}\right)^{\frac{3}{2}}R}{5\sqrt{2}\sqrt{\pi}\,M}$$

Для побудови графіка залежності функції розподілу від температури підставляємо молярну масу повітря і величину універсальної газової сталої (див. результати на рис. 7.1):

```
(%i5) plot2d([fun70,fun150,fun300],[v,0,1000]),M=0.029,R=8.314;
```

Можна вивчити вплив температури на форму кривої, а також на розташування максимуму функції розподілу. За допомогою інтегрування $f(v)$ можна порахувати частку молекул, що мають швидкості у певному інтервалі, а також визначити середню і середню квадратичну швидкості молекул.

Приклад:

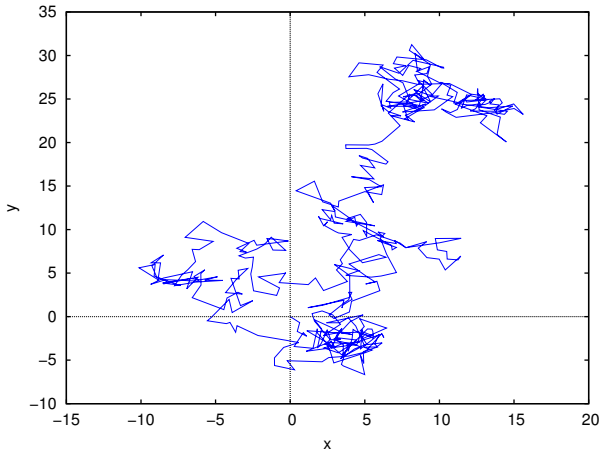


Рис. 7.2. Траєкторія броунівського руху модельної частки.

```
(%i6) integrate(v*v*fun,v,0,inf);
Is M,R,T positive, negative, or zero?
p;
```

(%o6) $\frac{3\,R^{\frac{5}{2}}\left(\frac{M}{R\,T}\right)^{\frac{3}{2}}T^{\frac{5}{2}}}{M^{\frac{5}{2}}}$

```
(%i7) radcan(%);
```

(%o7) $\frac{3\,R\,T}{M}$

Таким чином, $\langle v^2\rangle = \frac{3RT}{M}$, звідки середньоквадратична швидкість молекул газу $\langle\sqrt{v^2}\rangle = v_{sq} = \sqrt{\frac{3RT}{M}}$.

7.2.3 Броунівський рух

Наявність генератора випадкових чисел дає можливість моделювати рух броунівської частки.

Ейнштейн перший розрахував параметри броунівського руху, показавши, що нерегулярне переміщення часток, змулених у рідині, викликане випадковими ударами сусідніх молекул внаслідок теплового руху. Відповідно до теорії Смолуховського-Ейнштейна, середнє значення квадрата зсуву броунівської частки (s^2) за час t прямо пропорційне температурі T і обернено пропорційне в'язкості рідини h, розміру частки r і сталій Авогадро N_A: $s^2 = \frac{2RTt}{6\pi hrN_A}$, де R – газова стала.

Броунівські частки мають розмір порядку 0,1-1 мкм, тобто від однієї тисячної до однієї десятитисячної частки міліметра.

Побудуємо трохи спрощену модель броунівського руху, припускаючи, що зсув частки за кожною із координат – нормально розподілена випадкова величина з нульовим математичним очікуванням. Для генерації випадкових чисел використаємо пакет `distrib`, що включає необхідні функції (використаний генератор `random_normal`).

```
(%i1) load("distrib")$ x:0$ y:0$ xy:[[0,0]]$ m:0$ s:1$ Nmax:500$
for i:1 thru Nmax do (x:x+random_normal(m,s), y:y+random_normal(m,s),
xy:append(xy,[[x,y]]))$ plot2d( [discrete,xy])$
```

Результат побудови графіка наведена на рис. 7.2.

7.3 Приклад побудови статистичної моделі

Розгляньмо побудову задачі із практичним змістом.

У таблиці 7.1 наведені дані (узяті зі статті В.Ф. Очкова: http://twt.mpei.ac.ru/ochkov/) про залежності ціни старого автомобіля від його пробігу і «віку» (часу використання). У статті-першоджерелі задача дослідження цієї залежності вирішувалося засобами **MathCad**.

Розгляньмо її розв'язання засобами **Maxima**.

Надалі вважаємо, що вихідні дані для розв'язання підготовлено у форматі файла `cars.txt`. Для зчитування використаємо пакет `numericalio`. У пам'яті дані представляються матрицею, а для побудови окремих графіків – списками (змінні age, mile, price – див. нижче).

```
(%i1) load("draw")$
(%i2) load("numericalio")$
(%i3) data:read_matrix("cars.txt")$
(%i4) age:makelist(data[k,1],k,1,30)$
```

Табл. 7.1. Вартість старого автомобіля залежно від його віку і пробігу

Вік (у роках)	**Пробіг (у милях)**	**Ціна (у $)**	**Вік (у роках)**	**Пробіг (у милях)**	**Ціна (у $)**
11.5	88000	1195	13.5	103000	750
10.5	82000	1295	10.5	65000	1495
12.5	97000	800	10.5	70000	1495
8.5	51000	2295	10.5	80000	1495
9.5	79000	1995	6.5	57000	2695
13.5	120000	495	11.5	101000	895
3.5	39000	4995	10.5	78000	1295
6.5	52000	2695	9.5	84000	1995
4.5	39000	3995	4.5	46000	3675
12.5	92000	795	11.5	108000	975
7.5	41000	3495	13.5	124000	850
10.5	77000	1595	6.5	56000	3495
12.5	83000	895	9.5	67000	2495
4.5	38000	3990	6.5	43000	3400
13.5	92000	795	11.5	78000	1295

```
(%i5) mile:makelist(data[k,2],k,1,30)$
(%i6) price:makelist(data[k,3],k,1,30)$
```

Найпростішу лінійну регресію можна побудувати, використовуючи функцію `simple_linear_regression` (пакет `stats`). Побудуємо залежність ціни автомобіля від його віку і пробігу:

```
(%i21) xy:makelist([age[k],price[k]],k,1,30)$
(%i22) simple_linear_regression(xy);
```

$$
\text{(\%o22)}\quad \left(\begin{array}{c}
SIMPLE\ \ LINEAR\ \ REGRESSION \\
model = 5757.594446543255 - 392.7181715149224\,x \\
correlation = -0.9688177942467208 \\
v\_estimation = 95364.34912839333 \\
b\_conf\_int = [-431.5987157329751, -353.8376272968697] \\
hypotheses = H0:\ \ b\ \ =\ \ 0\ \ , H1:\ \ b\ \ \#\ \ 0 \\
statistic = 20.69021212080514 \\
distribution = [student\_t, 28] \\
p\_value = 0.0
\end{array}\right)
$$

Побудуємо аналогічну залежність ціни автомобіля від пробігу, але не у лінійній, а в експоненційній формі:

```
(%i26) xy:makelist([mile[k],float(log(price[k]))],k,1,30)$
(%i27) simple_linear_regression(xy);
```

$$
\text{(\%o27)}\quad \left(\begin{array}{c}
SIMPLE\ \ LINEAR\ \ REGRESSION \\
model = 9.174960600286802 - 2.374771512074816\,10^{-5}\,x \\
correlation = -0.9301125564244438 \\
v\_estimation = 0.05467789749118319 \\
b\_conf\_int = [-2.737778081063126\,10\hat{}-5, -2.011764943086506\,10^{-5}] \\
hypotheses = H0:\ \ b\ \ =\ \ 0\ \ , H1:\ \ b\ \ \#\ \ 0 \\
statistic = 13.4005809840375 \\
distribution = [student\_t, 28] \\
p\_value = 1.054711873393899\,10^{-13}
\end{array}\right)
$$

Отримані залежності представлені у вигляді виразів **Maxima**:

```
(%i28) fun1:5757.6-392.7*x$
(%i29) exp(9.175);
(%o29) 9652.76807161659
(%i30) fun2:9653*exp(-2.375*10^(-5)*x);
```

$$\text{(\%o30)}\quad 9653\,e^{-2.375\,10^{-5}\,x}$$

Проілюструємо отримані результати графічно (рис. 7.3 і рис. 7.4):

```
(%i34) draw2d(terminal=eps,key="Table",xlabel="Age",ylabel="Price",point_size=3,point_type=3,
points(age,price), key="price=f(age)",explicit(fun1,x,0,15));
(%i41) draw2d(terminal=eps,key="Table",xlabel="Mile",ylabel="Price",point_size=3,point_type=3,
points(mile,price), key="price=f(mile)",explicit(fun2,x,0,125000));
```

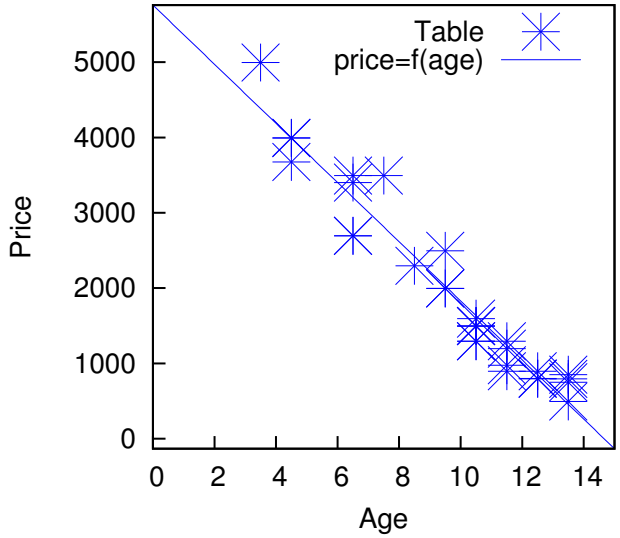


Рис. 7.3. Залежність ціни старого автомобіля від його віку.

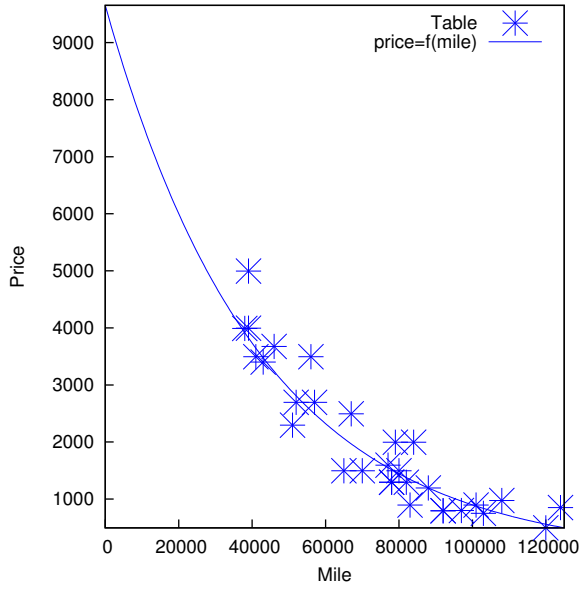


Рис. 7.4. Залежність ціни старого автомобіля від його пробігу.

Для побудови моделі у вигляді залежності ціни автомобіля від пробігу і віку одночасно доцільно використати складнішу функцію `lsquares_estimates` (пакет `lsquares`). Шукана модель була представлена рівнянням:

$$Price = a + b * Age + c * Mile + d * Mile^2$$

Необхідні команди **Maxima**:

```
(%i5) lsquares_estimates(data,[x,y,z],z=a+b*x+c*y+d*y^2,[a,b,c,d]);
```

(%o5) $[[a = \frac{36712000090549571}{5117101479342}, b = -\frac{80056614985946}{284283415519}, c = -\frac{194393701258481}{3411400986228000}, d = \frac{2937180994967}{10234202958684000000}]]$

```
(%i6) float(%);
```

(%o6) $[[a = 7174.374055069611, b = -281.6084604857839, c = -0.05698353903374546, d = 2.869965552593152\,10^{-7}]]$

Слід зазначити, що сильно нелінійні задачі розв'язуються за допомогою `lsquares_estimate` повільно, тому результати побудови моделі сильно залежать від обґрунтованості постановки задачі оцінювання. Графічна ілюстрація представлена на рис. 7.5.

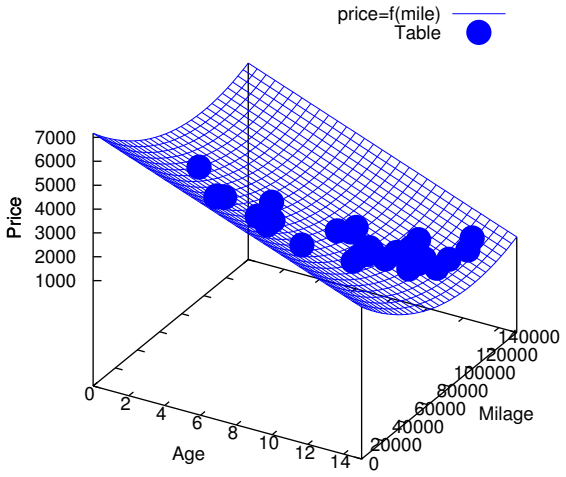


Рис. 7.5. Ілюстрація залежності відгуку (ціни старого автомобіля) від двох незалежних факторів (віку і пробігу автомобіля).

Розділ 8

Реалізація деяких обчислювальних методів

8.1 Програмування методів розв'язання нелінійних рівнянь у Maxima

Потреба у відшуканні коренів нелінійних рівнянь зустрічається у цілому ряді задач: розрахунках систем автоматичного керування і регулювання, власних коливань машин і конструкцій, у задачах кінематичного аналізу і синтезу, плоских і просторових механізмів та інших задачах.

Нехай дано нелінійне рівняння $f(x) = 0$, і потрібно розв'язати це рівняння, тобто знайти його корінь $\bar{x}$.

Якщо функція має вигляд многочлена степеня m, $f(x) = a_0x^m + a_1x^{m-1} + a_2x^{m-2} + \cdots + a_{m-1}x + a_m$, де a_i – коефіцієнти многочлена, $i = \overline{1,m}$, то рівняння $f(x) = 0$ має m коренів (основна теорема алгебри).

Якщо функція $f(x)$ містить у собі тригонометричні або експоненційні функції від деякого аргументу x, то рівняння $f(x) = 0$ називається трансцендентним рівнянням. Такі рівняння зазвичай мають нескінченну множину розв'язків.

Як відомо, не всяке рівняння може бути розв'язане точно. У першу чергу це стосується більшості трансцендентних рівнянь.

Доведено також, що не можна побудувати формулу, за якою можна було б розв'язувати довільні алгебраїчні рівняння степеня, вище четвертого.

Однак точне розв'язання рівняння не завжди є необхідним. Задачу відшукання коренів рівняння можна вважати практично розв'язаною, якщо ми зуміємо знайти корені рівняння із заданим рівнем точності. Для цього використовуються наближені (обчислювальні) методи розв'язування.

Більшість наближених методів розв'язування рівнянь, які використовують на практиці, є, по суті, способами уточнення кореня. Для їхнього застосування необхідне знання інтервалу ізоляції $[a, b]$, у якому лежить корінь рівняння, розташування кого уточнюється.

Процес визначення інтервалу ізоляції $[a, b]$, що містить тільки один з коренів рівняння, називається відділенням цього кореня.

Процес відділення кореня виконують виходячи з фізичного змісту прикладної задачі, графічно, за допомогою таблиць значень функції $f(x)$ або за допомогою спеціальної програми відділення коренів. Процедура відділення кореня заснована на відомій властивості неперервних функцій: якщо функція неперервна на замкнутому інтервалі $[a, b]$ і на його кінцях має різні знаки, тобто $f(a)f(b) < 0$, то між точками a і b є хоча б один корінь рівняння $f(x) = 0$. Якщо при цьому функція $f(x)$ на відрізку $[a, b]$ монотонна, то зазначений корінь єдиний.

Процес визначення коренів алгебраїчних і трансцендентних рівнянь складається із двох етапів:

- відділення коренів, – тобто визначення інтервалів ізоляції $[a, b]$, усередині якого лежить кожен корінь рівняння;
- уточнення коренів, – тобто звуження інтервалу $[a, b]$ до величини рівній заданому рівню точності ε.

Для алгебраїчних і трансцендентних рівнянь придатні ті самі методи уточнення наближених значень дійсних коренів:

- метод ділення навпіл (метод дихотомії);
- метод простих ітерацій;
- метод Ньютона (метод дотичних);
- модифікований метод Ньютона (метод січних);
- метод хорд тощо.

8.1.1 Метод ділення навпіл

Розгляньмо наступну задачу: дано нелінійне рівняння $f(x) = 0$, потрібно знайти корінь рівняння, що належить інтервалу $[a, b]$, із заданою точністю ε.

Для уточнення кореня методом ділення навпіл послідовно здійснюємо такі операції:

- обчислюємо значення функції $f(x)$ у точках a і $t = (a + b)/2$;
- якщо $f(a)f(t) < 0$, то корінь перебуває в лівій половині інтервалу $[a, b]$, тому відкидаємо праву половину інтервалу і приймаємо $b = t$;
- якщо умова $f(a)f(t) < 0$ не виконується, то корінь перебуває у правій половині інтервалу $[a, b]$, тому відкидаємо ліву половину інтервалу $[a, b]$ за рахунок присвоювання $a = t$.

В обох випадках новий інтервал $[a, b]$ у 2 рази менший попереднього.

Процес скорочення довжини інтервалу невизначеності циклічно повторюється доти, доки довжина інтервалу [a, b] не стане рівною або меншою заданої точності, тобто $|b - a| < \varepsilon$.

Реалізація методу ділення навпіл у вигляді функції **Maxima** представлена у наступному прикладі:

- власне функція `bisect`, у яку передається вираз f, що визначає рівняння, яких необхідно розв'язати

```
(%i1) bisect(f,sp,eps):=block( [a,b],a:sp[1],b:sp[2],p:0,
while abs(b-a)>eps do (
    p:p+1, c:(a+b)/2,
    fa:float(subst(a,x,f)), fc:float(subst(c,x,f)),
    if fa*fc<0 then b:c else a:c
),
float(c));
```

(%o1) bisect $(f, sp, eps) :=$ block$([a, b], a : sp_1, b : sp_2, p : 0,$
while $|b - a| > eps$ do$(p : p + 1, c : \frac{a+b}{2}, fa :$ float (subst $(a, x, f))$,
$fc :$ float (subst $(c, x, f))$, if $fa\, fc < 0$ then $b : c$ else $a : c), $float $(c))$

- послідовність команд, що організує звертання до bisect і результати обчислень

```
(%i2) f:exp(-x)-x$
a:-1$ b:2$ eps:0.000001$ xrez:bisect(f,[-2,2],eps)$
print("Розв'язок ",xrez," Нев'язка ",subst(xrez,x,f));
```

Розв'язок 0.56714344024658 Нев'язка $-2.348157265297246\ 10^{-7}$

(%o2) $-2.348157265297246\ 10^{-7}$

У представленому прикладі розв'язується рівняння $e^{-x} - x = 0$. Пошук кореня здійснюється на відрізку $[-2, 2]$ з точністю 0.000001.

Слід зазначити особливість програмування для **Maxima**, що полягає у тому, що розв'язуване рівняння задається у вигляді математичного виразу (тобто фактично текстового рядка). Числове значення нев'язки рівняння обчислюється за допомогою функції `subst`, за допомогою якої виконується підставляння значень $x = a$ або $x = c$ у заданий вираз. Обчислення нев'язки після розв'язування здійснюється також шляхом підставляння результату розв'язування $xrez$ у вираз f.

8.1.2 Метод простих ітерацій

У ряді випадків досить зручним прийомом уточнення кореня рівняння є метод послідовних наближень (метод ітерацій).

Нехай з точністю ε необхідно знайти корінь рівняння $f(x) = 0$, що належить інтервалу ізоляції $[a, b]$. Функція $f(x)$ і її перша похідна неперервні на цьому відрізку.

Для застосування цього методу вихідне рівняння $f(x) = 0$ слід привести до вигляду $x = \varphi(x)$.

Як початкове наближення може бути вибрана будь-яка точка інтервалу $[a, b]$.

Далі ітераційний процес пошуку кореня будується за такою схемою:

$$x_1 = f(x_0),$$
$$x_2 = f(x_1),$$
$$\dots$$
$$x_n = f(x_{n-1}).$$

У результаті ітераційний процес пошуку реалізується рекурентною формулою. Процес пошуку припиняється, щойно виконується умова $|x_n - x_{n-1}| \leqslant \varepsilon$ або число ітерацій перевищить задане число N.

Для того, щоб послідовність $x_1, x_2, \dots, x_n$ наближалася до шуканого кореня, необхідно, щоб виконувалася умова збіжності $|\varphi'(x)| < 1$.

Приклад реалізації методу ітерацій наведено нижче:

```
(%i1) f:exp(-x)-x$
beta:0.1$ x1:1$ x0:0$ eps:0.000001$ p:0$
while abs(x1-x0)>eps do
    (x0:x1, p:p+1, x1:float(x0+beta*(subst(x0,x,f))))$
    print("Кількість ітерацій  ",p,"   ","Розв'язок ",float(x1)," Нев'язка ",float(abs(x1-x0)));
```

Кількість ітерацій 67 Розв'язок 0.56714848327814 Нев'язка $9.650298036234517\ 10^{-7}$

8.1.3 Метод Ньютона (метод дотичних)

Розглянуті раніше методи розв'язування нелінійних рівнянь є методами прямого пошуку. У них для знаходження кореня використовується знаходження значення функції у різних точках інтервалу $[a, b]$.

Метод Ньютона належить до градієнтних методів, у яких для знаходження кореня використовується значення похідної.

Розгляньмо нелінійне рівняння $f(x) = 0$, для якого потрібно знайти корінь на інтервалі $[a, b]$ з точністю ε.

Метод Ньютона заснований на заміні вихідної функції $f(x)$, на кожному кроці пошуку дотичною, проведеною до цієї функції. Перетин дотичної з віссю x дає наближення кореня.

Виберемо початкову точку $x_0 = b$ (кінець інтервалу ізоляції). Знаходимо значення функції у цій точці і проводимо до неї дотичну, перетин якої з віссю x дає перше наближення кореня x_1:

$$x_1 = x_0 - h_0, \text{ де } h_0 = \frac{f(x_0)}{\operatorname{tg}\alpha} = \frac{f(x_0)}{f'(x_0)}.$$

Тому $x_1 = x_0 - \dfrac{f(x_0)}{f'(x_0)}$.

У результаті, ітераційний процес збіжності до кореня реалізується рекурентною формулою

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}.$$

Процес пошуку продовжуємо доти, доки не буде виконано умову $|x_{n+1} - x_n| \leqslant \varepsilon$, звідки $\left|\dfrac{f(x_n)}{f'(x_n)}\right| \leqslant \varepsilon$.

Метод забезпечує швидку збіжність, якщо виконується умова $f(x_0) \cdot f''(x_0) > 0$, тобто першу дотичну рекомендується проводити у тій точці інтервалу $[a, b]$, де знаки функції $f(x_0)$ і її кривини $f''(x_0)$ збігаються.

Приклад реалізації методу Ньютона у **Maxima** наведено нижче.

```
(%i1) newton(f,x0,eps):=block([df,xn,xn0,r,p],
      xn0:x0, df:diff(f,x),
      p:0, r:1,
      while abs(r)>eps do (
      p:p+1, xn:xn0-float(subst(xn0,x,f)/subst(xn0,x,df)),
      print("x0,x1 ",xn0,xn),r:xn-xn0, xn0:xn
),
[xn,p])$
```

Послідовність команд для звертання до функції `newton` і результати обчислень представлені в наступному прикладі:

```
(%i2) f:exp(-x)-x$
eps:0.000001$ xrez:newton(f,1,eps)$
print("Розв'язок ",xrez[1]," Кількість ітерацій ",xrez[2],
" Нев'язка ",subst(xrez[1],x,f));
x0,x1 1.53788284273999
x0,x1 0.53788284273999 0.56698699140541
x0,x1 0.56698699140541 0.56714328598912
x0,x1 0.56714328598912 0.56714329040978
Розв'язок 0.56714329040978 Кількість ітерацій 4 Нев'язка 0.0
```

Особливості наведеного прикладу – проміжний друк результатів і повернуте значення у вигляді списку, що надає змогу одночасно одержати як значення кореня, так і необхідну для досягнення заданої точності кількість ітерацій. Істотному зменшенню кількості ітерацій сприяє і аналітичне обчислення похідної.

8.1.4 Модифікований метод Ньютона (метод січних)

У цьому методі для обчислення похідних на кожному кроці пошуку використовується обчислювальне диференціювання за формулою:

$$f'(x) = \frac{\Delta f(x)}{\Delta x}.$$

Тоді рекурентна формула методу Ньютона набуває вигляду:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n} = x_n - \frac{f(x_n)\Delta x}{\Delta f(x_n)} = x_n - \frac{f(x_n)\Delta x}{f(x_n + \Delta x) - f(x_n)},$$

де $\Delta x \approx \varepsilon$.

Реалізацію цього методу у **Maxima** наведено нижче:

```
(%i1) secant(f,sp,eps):=block([x0,x1,d,y,r],
      x0:sp[1] ,x1:sp[2],
      p:0, r:x1-x0, d:float(subst(x0,x,f)),
      while abs(r)>eps do (
          p:p+1, y:float(subst(x1,x,f)), r:r/(d-y)*y,
          d:y, x1:x1+r
      ),
      x1)$
(%i2) f:exp(-x)-x$
eps:0.000001$ xrez:secant(f,[-2,2],eps)$ print("Розв'язок ",xrez," Нев'язка ",subst(xrez,x,f))$
```

Розв'язок 0.56714329040978 Нев'язка $-1.1102230246251565\,10^{-16}$

Особливості програмування для **Maxima**, використані у цьому прикладі, аналогічні наведеним вище у прикладі, пов'язаному із методом ділення навпіл.

8.1.5 Метод хорд

Метод заснований на заміні функції $f(x)$ на кожному кроці пошуку хордою, перетин якої з віссю x дає наближення кореня.

При цьому у процесі пошуку сімейство хорд може будуватися:

а) при фіксованому лівому кінці хорд, тобто $z = a$, тоді початкова точка $x_0 = b$;

б) при фіксованому правому кінці хорд, тобто $z = b$, тоді початкова точка $x_0 = a$.

У результаті ітераційний процес збіжності до кореня реалізується рекурентною формулою:

$$x_{n+1} = x_n - \frac{f(x_n)}{f(x_n) - f(a)}(x_n - a) \text{для випадку а);}$$

$$x_{n+1} = x_n - \frac{f(x_n)}{f(x_n) - f(b)}(x_n - b) \text{для випадку б).}$$

Процес пошуку триває доти, доки не буде виконано умову

$$|x_{n+1} - x_n| \leqslant \varepsilon \text{ або } h \leqslant \varepsilon.$$

Метод забезпечує швидку збіжність, якщо $f(z) \cdot f''(z) > 0$, тобто хорди фіксуються у тому кінці відрізка $[a, b]$, де знаки функції $f(z)$ і її кривини $f''(z)$ збігаються.

8.2 Числове інтегрування

Задача числового інтегрування полягає у заміні початкової підінтегральної функції $f(x)$, для якої важко або неможливо записати первісну у аналітичній формі, деякою наближеною функцією $\varphi(x)$. Такою функцією звичайно є поліном (кусковий поліном) $\varphi(x) = \sum_{i=1}^{n} c_i \varphi_i(x)$.

Таким чином

$$I = \int_a^b f(x)dx = \int_a^b \varphi(x)dx + R,$$

де $R = \int_a^b r(x)dx$ – апріорна похибка методу на інтервалі інтегрування, а $r(x)$ – апріорна похибка методу на окремому кроці інтегрування.

8.2.1 Огляд методів інтегрування

Методи обчислення однократних інтегралів називаються квадратурними (для кратних інтегралів – кубатурними), і діляться на такі групи:

- Методи Ньютона-Котеса. Тут $\varphi(x)$ – поліном різних степенів. До цієї групи належать метод прямокутників, трапецій, Сімпсона.
- Методи статистичних випробувань (методи Монте-Карло). Тут вузли сітки для квадратурного або кубатурного інтегрування вибираються за допомогою датчика випадкових чисел, відповідь має імовірнісний характер. В основному застосовуються для обчислення кратних інтегралів.

- Сплайнові методи. Тут $\varphi(x)$ – кусковий поліном з умовами зв'язку між окремими поліномами за допомогою системи коефіцієнтів.
- Методи найвищої алгебраїчної точності. Забезпечують оптимальне розміщення вузлів сітки інтегрування і вибір вагових коефіцієнтів $\rho(x)$ у задачі $\int\limits_a^b \varphi(x)\rho(x)dx$ (характерний приклад – метод Гауса).

8.2.2 Метод прямокутників

Розрізняють метод лівих, правих і середніх прямокутників. Суть методу зрозуміла з назви. На кожному кроці інтегрування функція апроксимується поліномом нульового степеня – відрізком, паралельним осі абсцис.

Формули методу прямокутників можна одержати з аналізу розкладу функції $f(x)$ у ряд Тейлора поблизу деякої точки:

$$f(x)|_{x=x_i} = f(x_i) + (x - x_i)f'(x_i) + \frac{(x-x_i)^2}{2!}f''(x_i) + \dots$$

Розгляньмо діапазон інтегрування від x_i до $x_i + h$, де h – крок інтегрування. Обчислимо інтеграл від досліджуваної функції на цьому проміжку:

$$\int\limits_{x_i}^{x_i+h} f(x)dx = x \cdot f(x_i)|_{x_i}^{x_i+h} + \frac{(x-x_i)^2}{2}f'(x_i)\Big|_{x_i}^{x_i+h} + \frac{(x-x_i)^3}{3\cdot 2!}f''(x_i)\Big|_{x_i}^{x_i+h} + \dots =$$

$$f(x_i)h + \frac{h^2}{2}f'(x_i) + O(h^3) = f(x_i)h + r_i.$$

Таким чином, на базі аналізу ряду Тейлора отримана формула правих (або лівих) прямокутників і апріорна оцінка похибки r на окремому кроці інтегрування. Основний критерій, за яким судять про точність алгоритму — степінь при величині кроку у формулі апріорної оцінки похибки. У випадку рівномірного кроку h на всьому діапазоні інтегрування загальна формула має вигляд

$$\int\limits_a^b f(x)dx = h\sum_{i=0}^{n-1} f(x_i) + R,$$

де n – кількість відрізків розбиття інтервалу інтегрування, $R = \sum\limits_{i=0}^{n-1} r_i = \frac{h}{2}\cdot h\sum\limits_{i=0} f'(x_i) = \frac{h}{2}\int\limits_a^b f'(x)dx$. Отримана оцінка справедлива за наявності неперервної похідної підінтегральной функції $f'(x)$.

8.2.3 Метод середніх прямокутників

У цьому методі на кожному інтервалі значення функції обчислюється у середній точці відрізка $[x_i, x_i + h]$, тобто $\int\limits_{x_i}^{x_i+h} f(x)dx = hf(\bar{x}) + r_i$.

Розклад функції у ряд Тейлора показує, що у випадку середніх прямокутників точність методу суттєво підвищується:

$$r = \frac{h^3}{24}f''(\bar{x}), \quad R = \frac{h^2}{24}\int\limits_a^b f''(x)dx.$$

Приклад функції **Maxima**, що реалізує метод середніх прямокутників, представлений нижче:

```
(%i1) intpr(f,n,a,b):=block([h,i,s,_x],h:(b-a)/n, _x:a+h/2, s:0,
      for i:1 thru n do (s:s+float(subst(_x,x,f)),_x:_x+h),s:s*h);
(%i2) intpr(x^2,100,-1,1);
(%o2) 0.6666
```

8.2.4 Метод трапецій

Апроксимація у цьому методі здійснюється поліномом першого степеня. На одиничному інтервалі

$$\int\limits_{x_i}^{x_i+h} f(x)dx = \frac{h}{2}\left(f(x_i) + f(x_i + h)\right) + r_i.$$

Якщо сітка є рівномірною ($h = const$),

$$\int\limits_a^b f(x)dx = h\left(\frac{1}{2}f(x_0) + \sum_{i=1}^{n-1} f(x_i) + \frac{1}{2}f(x_n)\right) + R$$

При цьому $r_i = -\frac{h^3}{12} f''(x_i)$, а $R = -\frac{h^3}{12} \int\limits_a^b f''(x)dx$.

Похибка методу трапецій у два рази вища, ніж у методу середніх прямокутників. Однак на практиці знайти середнє значення на елементарному інтервалі можна тільки у функцій, заданих аналітично (а не таблично), тому використовувати метод середніх прямокутників вдається далеко не завжди. Через різні знаки похибки у формулах трапецій і середніх прямокутників точне значення інтеграла звичайно лежить між двома цими оцінками.

Приклад реалізації методу трапецій у вигляді функції наведено нижче:

```
(%i1) f:x^2;
(%o1) x^2
(%i2) inttrap(f,n,a,b):=block([h,i,s],h:(b-a)/n,
      s:(float(subst(a,x,f))+float(subst(b,x,f)))/2,
      for i:1 thru n-1 do (s:s+float(subst(a+i*h,x,f))), s:s*h);
(%i3) inttrap(f,100,-1,1);
(%o3) 0.6668
```

Підінтегральна функція задається у вигляді виразу **Maxima**. Вираз, що визначає підінтегральну функцію, можна задавати і безпосередньо при звертанні до методу, як у наступному прикладі:

```
(%i4) inttrap(x*sin(x),100,-1,1);
(%o4) 0.60242947746101
```

8.2.5 Метод Сімпсона

При використанні цього методу підінтегральна функція $f(x)$ заміняється інтерполяційним поліномом другого степеня $P(x)$ – параболою, що проходить через три сусідніх вузли. Розгляньмо два кроки інтегрування ($h = \text{const} = x_{i+1} - x_i$), тобто три вузли x_0, x_1, x_2, через які проведемо параболу, скориставшись рівнянням Ньютона:

$$P(x) = f_0 + \frac{x - x_0}{h}(f_1 - f_0) + \frac{(x - x_0)(x - x_1)}{2h^2}(f_0 - 2f_1 + f_2).$$

Нехай $z = x - x_0$, тоді

$$\begin{aligned} P(z) &= f_0 + \frac{z}{h}(f_1 - f_0) + \frac{z(z-h)}{2h^2}(f_0 - 2f_1 + f_2 = \\ &= f_0 + \frac{z}{2h}(-3f_0 + 4f_1 - f_2) + \frac{z^2}{2h^2}(f_0 - 2f_1 + f_2). \end{aligned}$$

Скориставшись отриманим співвідношенням, обчислимо інтеграл за даним інтервалом:

$$\int\limits_{x_0}^{x_2} P(x)dx = \int\limits_0^{2h} P(z)dz = 2hf_0 + \frac{(2h)^2}{4h}(-3f_0 + 4f_1 - f_2) + \frac{(2h)^3}{6h^2}(f_0 - 2f_1 + f_2) =$$

$$= 2hf_0 + h(-3f_0 + 4f_1 - f_2) + \frac{4h}{3}(f_0 - 2f_1 + f_2) = \frac{h}{3}(6f_0 - 9f_0 + 12f_1 - 3F_2 + 4f_0 - 8f_1 + 4f_2).$$

У підсумку $\int\limits_{x_0}^{x_2} f(x)dx = \frac{h}{3}(f_0 + 4f_1 + f_2) + r$.

Для рівномірної сітки і парної кількості кроків n формула Сімпсона набуває вигляду:

$$\int\limits_a^b f(x)dx = \frac{h}{3}(f_0 + 4f_1 + 2f_2 + 4f_3 + \cdots + 2f_{n-2} + 4f_{n-1} + f_n) + R,$$

де $r = -\frac{h^5}{90} f^{IV}(x_i)$ а $R = -\frac{h^4}{180} \int_a^b f^{IV}(x)dx$ у припущенні неперервності четвертої похідної підінтегральної функції.

Реалізацію методу Сімпсона засобами **Maxima** представлено у наступному прикладі:

```
(%i1) intsimp(f,n,a,b):=block([h,h2,i,_x,s],h:(b-a)/n, h2:h/2,
      s:(float(subst(a,x,f))+float(subst(b,x,f)))/2+2*float(subst(a+h2,x,f)),_x:a,
      for i:1 thru n-1 do (_x:_x+h,
          s:s+2*float(subst(_x+h2,x,f))+float(subst(_x,x,f))), s:s*h/3);
(%i2) intsimp(x^2,100,-1,1);
(%o2) 0.66666666666667
```

8.3 Методи розв'язування систем лінійних рівнянь

8.3.1 Загальна характеристика і класифікація методів розв'язування

Розгляньмо систему лінійних алгебраїчних рівнянь (скорочено – СЛАР):

$$A \cdot \bar{x} = \bar{f}, \tag{8.1}$$

де A – матриця $m \times m$, $\bar{x} = (x_1, x_2, \ldots, x_m)^T$ – шуканий вектор, $f = (f_1, f_2, \ldots, f_m)^T$ – заданий вектор. Будемо припускати, що визначник матриці A відмінний від нуля, тобто розв'язок системи (8.1) існує.

Методи обчислювального розв'язання системи (8.1) діляться на дві групи: прямі методи («точні») і ітераційні методи.

Прямими методами називаються методи, що надають змогу отримати розв'язок системи (8.1) за скінченну кількість арифметичних операцій. До цих методів належать метод Крамера, метод Гауса, LU-метод тощо.

Ітераційні методи (методи послідовних наближень) полягають у тому, що розв'язок системи (8.1) знаходять як границю послідовних наближень $\bar{x}^{n)}$ при $n \to \infty$, де n – номер ітерації. При використанні методів ітерації звичайно задається деяке мале число ε і обчислення виконуються доти, доки не буде виконана оцінка $||\bar{x}^{(n)} - \bar{x}|| < \varepsilon$. До цих методів належать метод Зейделя, Якобі, метод верхніх релаксацій тощо.

Варто зазначити, що реалізація прямих методів на комп'ютері приводить до розв'язання з похибкою, оскільки усі арифметичні операції над змінними із рухомою крапкою виконуються з округленням. Залежно від властивостей матриці початкової системи ці похибки можуть досягати значних величин.

8.3.2 Метод Гауса

Запишемо систему (8.1), у розгорнутому вигляді

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1m}x_m &= f_1, \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2m}x_m &= f_2, \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots & \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mm}x_m &= f_m. \end{cases}$$

Метод Гауса полягає у послідовному виключенні невідомих із цієї системи. Припустімо, що $a_{11} \neq 0$. Послідовно множачи перше рівняння на $-\dfrac{a_{i1}}{a_{11}}$ й складаючи з i-м рівнянням, виключимо x_1 із всіх рівнянь крім першого. Одержимо систему

$$\begin{cases} a_{11}x_1 + & a_{12}x_2 + \cdots + a_{1m}x_m &= f_1, \\ & a^{(1)}_{22}x_2 + \cdots + a^{(1)}_{2m}x_m &= f^{(1)}_2, \\ & \cdots\cdots\cdots\cdots\cdots & \\ & a^{(1)}_{m2}x_2 + \cdots + a^{(1)}_{mm}x_m &= f^{(1)}_m, \end{cases}$$

де

$$a^{(1)}_{ij} = a_{ij} - \frac{a_{i1}a_{1j}}{a_{11}}, f^{(1)}_i = f_i - \frac{a_{i1}f_1}{a_{11}}, i, j = 2, 3, \ldots, m.$$

Аналогічним чином з отриманої системи виключимо x_2. Послідовно виключаючи усі невідомі, отримаємо систему трикутного вигляду

$$\begin{cases} a_{11}x_1 + & a_{12}x_2 + & \ldots & + a_{1m}x_m &= f_1, \\ & a^{(1)}_{22}x_2 & + \cdots + & a^{(1)}_{2m}x_m &= f^{(1)}_2, \\ & \cdots\cdots & & & \\ & & a^{(m-1)}_{m-1,m-1}x_{m-1} + & a^{(m-1)}_{m-1,m}x_m &= f^{(m-1)}_m, \\ & & & a^{(m-1)}_{mm}x_m &= f^{(m-1)}_m, \end{cases}$$

Описана процедура називається прямим ходом методу Гауса. Помітимо, що її виконання було можливо за умови, що всі $a^{(l)}_{i,i}, l = 1, 2, \ldots, m-1$ не дорівнюють нулю. Виконуючи послідовні підставляння у останній системі, (починаючи з останнього рівняння) можна одержати всі значення невідомих.

$$x_m = \frac{f^{(m-1)}_m}{a^{(m-1)}_{mm}},$$
$$x_i = \frac{1}{a^{(i-1)}_{ii}} \left(f^{(i-1)}_i - \sum_{j=i-1}^{m} a^{(i-1)}_{ij} x_j \right).$$

Ця процедура називається зворотний хід методу Гауса.

Метод Гауса може бути легко реалізований на комп'ютері. При виконанні обчислень, як правило, проміжні значення матриці A непотрібні. Тому обчислювальна реалізація методу зводиться до перетворення елементів масиву розмірності $m \times (m+1)$, де $m+1$-й стовпець містить елементи правої частини системи.

Для контролю похибки реалізації методу використовуються, так звані, контрольні суми. Схема контролю ґрунтується на наступному очевидному твердженні. Збільшення значення всіх невідомих на одиницю рівносильно заміні даної системи контрольною системою, у якій вільні члени дорівнюють сумам всіх коефіцієнтів відповідного рядка. Створимо додатковий стовпець, що зберігає суму елементів матриці за рядками. На кожному кроці реалізації прямого ходу методу Гауса будемо виконувати перетворення і над елементами цього стовпця, і порівнювати їхнє значення із сумою за рядком перетвореної матриці. Якщо значення не збігаються, розрахунок переривається.

Один з основних недоліків методу Гауса пов'язаний з тим, що при його реалізації накопичується обчислювальна погрішність. Для систем порядку m число дій множення і ділення близьке до $\frac{m^3}{3}$ і швидко росте з величиною m.

Для того, щоб зменшити ріст обчислювальної похибки застосовуються різні модифікації методу Гауса. Наприклад, метод Гауса з вибором головного елемента за стовпцями, у цьому випадку на кожному етапі прямого ходу рядки матриці переставляються таким чином, щоб діагональний кутовий елемент був максимальним. При виключенні відповідного невідомого з інших рядків ділення буде виконуються на найбільший з можливих коефіцієнтів і, отже, відносна погрішність буде найменшою.

Приклад реалізації методу Гауса у **Maxima** реалізовано у наведеній нижче функції нижче (застосовано метод без вибору головного елемента):

```
(%i1) gauss(a0,b0,n):=block([a,b,i,j,k,d],
      a:copymatrix(a0),b:copymatrix(b0),x:copymatrix(b0),
      for i:1 thru n-1 do
      (
        for k:i+1 thru n do
        (
          d:a[k,i]/a[i,i],
          for j:i+1 thru n do (a[k,j]:a[k,j]-a[i,j]*d),
          b[k,1]:b[k,1]-b[i,1]*d
        )
      ),
      for i:n thru 1 step -1 do
      (
        for j:i+1 thru n do (
          b[i,1] :b[i,1]-a[i,j]*x[j,1]),
        x[i,1]:b[i,1]/a[i, i]
      ),
     x)$
```

Приклад виклику функції, що реалізує метод Гауса:

```
(%i2) aa:matrix([3,1,1],[1,3,1],[1,1,3]); bb:matrix([6],[6],[8]); zz:gauss(aa,bb,3);
```

$$\text{(\%o2)} \begin{pmatrix} 3 & 1 & 1 \\ 1 & 3 & 1 \\ 1 & 1 & 3 \end{pmatrix}$$

$$\text{(\%o3)} \begin{pmatrix} 6 \\ 6 \\ 8 \end{pmatrix}$$

$$\text{(\%o4)} \begin{pmatrix} 1 \\ 1 \\ 2 \end{pmatrix}$$

Перевірка обчислень показує, що перемножування матриці `aa` на вектор розв'язку `zz` дає вектор, що збігається з вектором правих частин `bb`.

```
(%i5) aa.zz;
```

$$\text{(\%o5)} \begin{pmatrix} 6.0 \\ 6.0 \\ 8.0 \end{pmatrix}$$

Існує метод Гауса з вибором головного елемента за усією матрицею. У цьому випадку переставляються не тільки рядки, але і стовпці. Використання модифікацій методу Гауса приводить до ускладнення алгоритму збільшенню числа операцій і відповідно до росту часу, потрібного для виконання обчислень.

Виконувані у методі Гауса перетворення прямого ходу, що привели матрицю A системи до трикутного вигляду надають змогу обчислити визначник матриці:

$$\det A = \begin{vmatrix} a_{11} & a_{a12} & \dots & a_{1m} \\ 0 & a_{22}^{(1)} & \dots & a_{2m}^{(1)} \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & a_{mm}^{(m-1)} \end{vmatrix} = a_{11} \cdot a_{22}^{(1)} \dots a_{mm}^{(m-1)}.$$

Метод Гауса надає змогу знайти і зворотну матрицю. Для цього необхідно розв'язати матричне рівняння

$$A \cdot X = I,$$

де I – одинична матриця. Його розв'язування зводиться до розв'язування m систем

$$A\bar{x}^{(j)} = \bar{\delta}^{(j)}, \quad j = 1, 2, \ldots, m,$$

у вектора $\bar{\delta}^{(j)}$ j-а компонента дорівнює одиниці, а інші компоненти дорівнюють нулю.

8.3.3 Метод квадратного кореня

Метод квадратного кореня заснований на розкладанні матриці A у добуток

$$A = S^T S,$$

де S – верхня трикутна матриця з додатними елементами на головній діагоналі, S^T – транспонована до неї матриця. Нехай A – матриця розміром $m \times m$. Тоді

$$\left(S^T S\right)_{ij} = \sum_{k=1}^{m} s_{ik}^T s_{kj}. \tag{8.2}$$

З умови (8.2) виходять рівняння

$$\sum_{k=1}^{m} s_{ik}^T s_{kj} = a_{ij}, \quad i, j = 1, 2, \ldots, m \tag{8.3}$$

Оскільки матриця A симетрична, не обмежуючи загальності, можна вважати, що у системі (8.3) виконується нерівність $i \leqslant j$. Тоді (8.3) можна переписати у вигляді

$$\sum_{k=1}^{i-1} s_{ik}^T s_{kj} + s_{ii} s_{ij} + \sum_{k=i+1}^{m} s_{ik}^T s_{kj} = a_{ij},$$

$$s_{ii} s_{ij} + \sum_{k=1}^{i-1} s_{ik}^T s_{kj} = a_{ij}, i \leqslant j.$$

Зокрема, при $i = j$ отримаємо

$$|s_{ii}|^2 = a_{ii} - \sum_{k=1}^{i-1} |s_{ki}|^2,$$

$$s_{ii} = \left(\left| a_{ii} - \sum_{k=1}^{i-1} |s_{ki}|^2 \right| \right)^{1/2}.$$

Далі, при $i < j$ отримаємо

$$s_{ij} = \frac{a_{ij} - \sum\limits_{k=1}^{i-1} s_{ik}^T s_{kj}}{s_{ii}}$$

За наведеними формулами знаходяться рекурентно усі ненульові елементи матриці S.

Зворотний хід методу квадратного кореня полягає у послідовному розв'язування двох систем рівнянь із трикутними матрицями.

$$S^T y = f,$$

$$Sx = y.$$

Розв'язки цих систем знаходять за рекурентними формулами:

$$\begin{cases} y_i = \dfrac{f_i - \sum\limits_{k=1}^{i-1} s_{ki} y_k}{s_{ii}}, \quad i = 2, 3, \ldots, m \\ y_1 = \dfrac{f_1}{s_{11}} \end{cases}$$

$$\begin{cases} x_i = \dfrac{y_i - \sum\limits_{k=i+1}^{m} s_{ik} x_k}{s_{ii}}, \quad i = m-1, m-2, \ldots, 1 \\ x_m = \dfrac{y_m}{s_{mm}} \end{cases}$$

Усього метод квадратного кореня при факторизації $A = S^T S$ потребує приблизно $\dfrac{m^3}{3}$ операцій множення і ділення і m операцій добування квадратного кореня.

Приклад функції, що реалізує метод квадратного кореня:

```
(%i1) holetsk(a0,b0):=block([L,Lt,x,y,i,j,k,n],
      n:length(a0), L:zeromatrix(n,n),
      for i:1 thru n do (
        for j:1 thru i-1 do (
        s:0,
        for k:1 thru j-1 do (s:s+L[i,k]*L[j,k]),
          L[i,j] :1/L[j,j]*(a0[i,j]-s)
        ),
        s:0,
        for k:1 thru i-1 do (s:s+L[i,k]^2),
        L[i,i] : sqrt(a0 [i,i]-s)
      ),
      Lt:transpose(L),
      y:zeromatrix(n,1), x:zeromatrix(n,1),
      for i:1 thru n do (
        s:0,
      for k:1 thru i-1 do (s:s+L [i,k]*y[k,1]),
        y[i,1]: (b0[i,1]-s)/L[i,i]
    ),
    for i:n thru 1 step -1 do (
      s:0,
      for k:n thru i+1 step -1 do (s:s+Lt[i,k]*x[k,1]),
      x[i,1]:(y[i,1]-s)/Lt[i,i]
    ),x
    )$
```

Тест цієї функції (розв'язання системи $Ax = B$, B – матриця $n \times 1$, A – квадратна симетрична матриця $n \times n$, результат розв'язування – вектор $n \times 1$):

```
(%i2) A:matrix([4,1,1],[1,4,1],[1,1,4])$ B:matrix([1],[1],[1])$
```

Результати обчислень:

```
(%i4) x:holetsk(A,B);
```

$$\text{(\%o4)} \quad \begin{pmatrix} \frac{1}{6} \\ \frac{1}{6} \\ \frac{1}{6} \end{pmatrix}$$

Перевірка:

```
(%i5) A.x;
```

$$\text{(\%o5)} \quad \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

8.3.4 Коректність постановки задачі і поняття обумовленості

При використанні обчислювальних методів для розв'язання тих або інших математичних задач слід розрізняти властивості самої задачі і властивості обчислювального алгоритму, призначеного для її розв'язання.

Говорять, що задача поставлена коректно, якщо розв'язок існує і єдиний і якщо він неперервно залежить від вхідних даних. Остання властивість називається також стійкістю щодо вхідних даних.

Коректність вихідної математичної задачі ще не гарантує гарних властивостей обчислювального методу її розв'язання і вимагає спеціального дослідження.

Відомо, що розв'язання задачі (8.1) існує тоді і тільки тоді, коли $\det A \neq 0$. У цьому випадку можна визначити обернену матрицю A^{-1} і розв'язок записати у вигляді $\bar{x} = A^{-1}f$.

Дослідження на стійкість задачі (8.1) зводиться до дослідження залежності її розв'язку від правих частин f і елементів a_{ij} матриці A. Для того щоб можна було говорити про неперервну залежність вектора розв'язків від деяких параметрів, слід на множині m-вимірних векторів, що належить лінійному простору $\mathbb{H}$, ввести метрику.

У лінійній алгебрі пропонується визначення множини метрик L_p – норма $||\bar{x}||_p = \left(\sum\limits_{i=1}^{m} |x_i|^p \right)^{1/p}$, з якого просто отримати найпоширеніші метрики.

- при $p = 1$ $||\bar{x}||_1 = \sum\limits_{i=1}^{m} |x_i|$,
- при $p = 2$ $||\bar{x}||_2 = \left(\sum\limits_{i=1}^{m} x_i^2 \right)^{1/2}$,

- при $p \to \infty$ $||\bar{x}||_\infty = \lim_{p\to\infty} \left(\sum_{i=1}^{m} x_i^p \right)^{1/p}$.

Підлеглі норми матриць, що визначаються як $||A|| = \sup\limits_{0\neq x\in\mathbb{H}} \dfrac{||A\bar{x}||}{||\bar{x}||}$, відповідно записуються у такому вигляді:

$$||A||_1 = \max_{1\leqslant j\leqslant m} \sum_{i=1}^{m} |a_{ij}|,$$

$$||A||_\infty = \max_{1\leqslant i\leqslant m} \sum_{j=1}^{m} |a_{ij}|,$$

$$||A||_2 = \sqrt{\rho(A^T A)} = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{m} a_{ij}^2}.$$

Звичайно розглядають два види типи стійкості розв'язків системи (8.1):

- за правими частинами;
- за коефіцієнтами системи (8.1) і за правими частинами.

Поряд з початковою системою (8.1) розглянемо систему зі «збуреними» правими частинами:

$$A \cdot \tilde{x} = \tilde{f},$$

де $\tilde{f} = \bar{f} + \delta\bar{f}$ – збурена права частина системи, а $\tilde{x} = \bar{x} + \delta\bar{x}$ – збурений розв'язок.

Можна отримати оцінку, що виражає залежність відносної похибки розв'язку від відносної погрішності правих частин:

$$\frac{||\delta\bar{x}||}{||\bar{x}||} \leqslant M_A \frac{||\delta\bar{f}||}{||\bar{f}||},$$

де $M_A = ||A|| \cdot ||A^{-1}||$ – число обумовленості матриці A (у сучасній літературі це число позначають як $cond(A)$). Якщо число обумовленості велике ($M_A \sim 10^k$, $k > 2$), то говорять, що матриця A погано обумовлена. У цьому випадку малі збурення правих частин системи (8.1), викликані або неточністю задання початкових даних, або викликані погрішностями обчислення істотно впливають на розв'язок системи.

Якщо збурення внесено до матриці A, то для відносних збурень розв'язку має місце наступна оцінка:

$$\frac{||\delta\bar{x}||}{||\bar{x}||} \leqslant \frac{M_A}{1 - M_A \dfrac{||\delta A||}{||A||}} \left(\frac{||\delta A||}{||A||} + \frac{||\delta\bar{f}||}{||\bar{f}||} \right).$$

У **Maxima** матричні норми обчислюються за допомогою функції `mat_norm`. Синтаксис виклику: `mat_norm(M, type)`, де `M` – матриця, `type` – тип норми, `type` може бути дорівнює `1` (норма $||A||_1$), `inf` (норма $||A||_\infty$), `frobenius` (норма $||A||_2$).

Приклад обчислення зазначених типів норми у **Maxima**:

```
(%i1) A:matrix([1,2,3],[4,5,6],[7,8,9]);
(%o1) ⎛1  2  3⎞
      ⎜4  5  6⎟
      ⎝7  8  9⎠
(%i2) mat_norm(A,1);
(%o2) 18
(%i3) mat_norm(A,inf);
(%o3) 24
(%i4) mat_norm(A,frobenius);
(%o4) √285
```

Обчислимо число обумовленості для погано і добре обумовлених матриць:

```
(%i1) A:matrix([1,1],[0.99,1]);
(%o1) ⎛   1   1⎞
      ⎝0.99  1⎠
(%i2) nr:mat_norm(A,frobenius);
(%o2) 1.995018796903929
(%i3) A1:invert(A);
(%o3) ⎛ 99.99999999999991  - 99.99999999999991⎞
      ⎝ -98.99999999999991  99.99999999999991⎠
(%i4) nr1:mat_norm(A1,frobenius);
(%o4) 199.5018796903927
(%i5) MA:nr*nr1;
(%o5) 398.0099999999997
```

Таким чином, для цієї погано обумовленої матриці число обумовленості досягає майже 400.

У такий самий спосіб (з використанням норми Фробеніуса) можна встановити, що для матриці $B = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ число обумовленості складає $M_B = 2$.

8.3.5 Щодо обчислювальних затрат

Один з важливих факторів, що визначають вибір того або іншого методу при розв'язанні конкретних задач, є обчислювальна ефективність методу. З огляду на те, що операція додавання виконується набагато швидше, ніж операція множення і ділення, звичайно обмежуються підрахунком останніх. Для розв'язання методом Гауса без вибору головного елемента потрібно $\frac{m^3}{3}+m^2-\frac{m}{3}$ множень і ділень, розв'язання СЛАР методом квадратного кореня потребує $\frac{m^3}{6}+\frac{3m^2}{2}+\frac{m}{3}$ множень і ділень і m операцій взяття кореня. За великих значень розмірності m можна вважати, що обчислювальні затрати на операції множення і ділення у методі Гауса складають $O\left(\frac{m^3}{3}\right)$, а у методі квадратного кореня – $O\left(\frac{m^3}{6}\right)$.

8.4 Ітераційні методи

У наближених або ітераційних методах розв'язфння системи лінійних алгебраїчних рівнянь розв'язок є границею ітеративної послідовності, отриманої за допомогою цих методів. До цієї групи методів належать: метод простої ітерації, метод Зейделя тощо. Ітераційні методи вигідні для системи спеціального виду, зі слабко заповненою матрицею дуже великого порядку $10^3 \ldots 10^5$.

Для ітераційних методів характерно те, що для їхньої реалізації потрібне початкове наближення значень невідомих, розв'язок шукають у вигляді послідовності наближень з поступовим покращенням, і крім того, ітераційний процес повинен бути збіжним. В обчислювальній практиці процес ітерації звичайно триває доти, доки два послідовних наближення не збігатимуться у межах заданої точності.

8.4.1 Матричне формулювання ітераційних методів розв'язування систем лінійних рівнянь

При використанні СКМ **Maxima** цілком обґрунтовані використання матричного формулювання ітераційних методів.

Розгляньмо розв'язання системи $Ax = f$ (A – квадратна матриця, f – вектор правих частин, x – вектор невідомих). Позначимо $A = L + D + U$, де L – нижня трикутна матриця з нульовими діагональними елементами; D – діагональна матриця; U – верхня трикутна матриця з нульовими діагональними елементами. Для розв'язання цієї системи розглянемо ітераційний процес:

$$x^{i+1} = x^i - H_i(Ax^i - f),$$

де $\det H_i = 0$, або

$$x^{i+1} = P_i x^i + d_i, \quad x(0) = x_0,$$

де $P_i = I - P_i$ – оператор і-го кроку ітераційного процесу; $d_i = P_i A$.

Ітераційний процес збіжний, якщо послідовність $\{x_i\}$ збігається до розв'язку x^* за довільного x_0.

Якщо матриця не залежить від номера ітерації, ітераційний процес називається стаціонарним:

$$x^{i+1} = Px^i + d. \tag{8.4}$$

Необхідною і достатньою умовою збіжності стаціонарного процесу є виконання умови $\rho(P) < 1$, де $\rho(P)$ – спектральний радіус матриці P (найбільше за модулем власне число матриці P).

З використанням уведених позначень метод простої ітерації (метод Якобі) задається формулою:

$$P = I - D^{-1}A, \quad D = diag(a_{ii}), \quad H = D^{-1},$$

а метод Гауса-Зейделя – формулою:

$$P = -(D + L^{-1}U, \quad H = (D + L)^{-1}.)$$

Розгляньмо розрахункові співвідношення за елементами для методів Якобі і Гауса-Зейделя.

Всі елементи головної діагоналі матриці $I = D^{-1}A$ дорівнюють нулю, інші елементи рівні $-\frac{a_{ij}}{a_{ii}}$, $i, j = \overline{1, n}$. Вільний член рівняння (8.4) дорівнює $\frac{f_i}{a_{ii}}$.

Таким чином, для методу Якобі ітераційний процес записується у вигляді $x^{k+1} = Cx^k + E$, де $C_{ij} = -\frac{a_{ij}}{a_{ii}}$; $k = 0, 1, \ldots$, $i, j = \overline{1, n}$; $E_i = \frac{f_i}{a_i i}$.

Для методу Гауса-Зейделя $x^{k+1} = -(D+L)^{-1}Ux^k + (D+L)^{-1}f$, або $(D+L)x^{k+1} = -Ux^k + b$, $x^{k+1} = Bx^{k+1} + Ex^k + e$, де $B = \begin{pmatrix} 0 & 0 & \dots & 0 \\ c_{21} & 0 & \dots & 0 \\ c_{31} & c_{32} & \dots & 0 \\ \dots & \dots & \dots & \dots \\ c_{n-1,1} & c_{n-1,2} & \dots & 0 \\ c_{n1} & c_{n2} & \dots & 0 \end{pmatrix}$, $E = \begin{pmatrix} 0 & c_{12} & \dots & \dots & c_{1n} \\ 0 & 0 & \dots & \dots & c_{2n} \\ 0 & & \ddots & & \vdots \\ \vdots & \dots & \dots & \ddots & c_{n-1,n} \\ 0 & \dots & \dots & \dots & 0 \end{pmatrix}$

Розгляньмо розв'язання конкретної системи рівнянь $Ax = b$ методом Якобі:

$$\begin{aligned} 8x_1 - x_2 + 2x_3 &= 8, \\ x_1 + 9x_2 + 3x_3 &= 18, \\ 2x_1 - 3x_2 + 10x_3 &= -5. \end{aligned}$$

Обчислюємо елементи матриці B і вектора e:

$$B = \begin{pmatrix} 0 & \frac{1}{8} & -\frac{2}{8} \\ -\frac{1}{9} & 0 & -\frac{3}{9} \\ -0,2 & 0,3 & 0 \end{pmatrix}, e = \begin{pmatrix} 1 \\ 2 \\ 0,5 \end{pmatrix}$$

Обчислимо значення x за формулою $x^{k+1} = Bxk^{+}e$. Для розв'язання використана наступна послідовність команд **Maxima**:

– перетворення заданих матриць

```
(%i1) A:matrix([8,-1,2],[1,9,3],[2,-3,10])$
      b: matrix([8],[18],[5])$
      A0:matrix([A[1,1],A[1,1],A[1,1]],
                [A[2,2],A[2,2],A[2,2]],
                [A[3,3],A[3,3],A[3,3]])$
      B:-A/A0+diagmatrix(3,1)$
      e: b/matrix([A[1,1]],[A[2,2]],[A[3,3]])$ x:e$
```

– власне обчислення розв'язку

```
(%i7) xt:float(B.x+e)$
      xt:float(B.xt+e)$
      xt:float(B.xt+e)$
      xt:float(B.xt+e)$
      xt:float(B.xt+e)$
      xt:float(B.xt+e)$
      xt:float(B.xt+e)$
      x0:xt$
      xt:float(B.xt+e)$
      x1:xt$
      r:x1-x0$
      float(r); /* оцінка збіжності */
      float(A.xl-b); /* оцінка нев'язки */
```

$$\text{(\%o18)} \quad \begin{pmatrix} -1.2272477756924971 & 10^{-5} \\ -1.3018148195786949 & 10^{-4} \\ 6.2047575160040225 & 10^{-5} \end{pmatrix}$$

$$\text{(\%o19)} \quad \begin{pmatrix} 2.5427663227794994 & 10^{-4} \\ 1.738702477211973 & 10^{-4} \\ 3.6599949035931445 & 10^{-4} \end{pmatrix}$$

8.4.2 Метод простої ітерації

Для розв'язання системи лінійних алгебраїчних рівнянь (8.1) ітераційним методом її слід привести до нормального вигляду:

$$\bar{x} = P\bar{x} + \bar{g}. \tag{8.5}$$

Стаціонарне ітераційне правило одержуємо, якщо матриця B і вектор $\bar{g}$ не залежать від номера ітерації: $\bar{x}^{k+1} = P\bar{x}^k + \bar{g}$. Нестаціонарне ітераційне правило отримуємо якщо матриця B або вектор $\bar{g}$ змінюються з ростом номера ітерації: $\bar{x}^{k+1} = P_k\bar{x}^k + \bar{g}^k$.

Стаціонарне ітераційне правило звичайно називають методом простої ітерації. Границя ітераційної послідовності є точним розв'язком системи (8.5) або (8.1).

Для того, щоб метод простої ітерації збігався при будь-якому початковому наближенні, необхідно і достатньо, щоб всі власні значення матриці B були за модулем менші одиниці.

Через те, що перевірити сформульовану вище умову достатньо складно на практиці застосовують такі достатні ознаки:

- для того, щоб метод простої ітерації збігався, достатньо, щоб якась норма матриці P була меншою одиниці;
- для того, щоб метод простої ітерації збігався, достатньо, щоб виконувалося одна з наступних умов:

$$\sum_{j=1}^{n} |P_{ij}| < 1,\ i = \overline{1,n};$$

$$\sum_{i=1}^{n} |P_{ij}| < 1,\ j = \overline{1,n};$$

$$\sum_{i,j=1}^{n} |P_{ij}|^2 < 1.$$

Для визначення швидкості збіжності можна скористатися наступною теоремою: якщо якась норма матриці P, узгоджена з даною нормою вектора, менше одиниці, то має місце така оцінка погрішності методу простої ітерації:

$$||\bar{x}^* - \bar{x}^k|| < ||P||^k \cdot ||\bar{x}^k|| + \frac{||P||^k \cdot ||\bar{g}||}{1 - ||P||den},$$

де $\bar{x}^*$ – точний розв'язок системи (8.1).

Інакше кажучи, умова збіжності виконується, якщо виконується умова домінування діагональних елементів матриці початкової системи A за рядками або стовпчиками: $\sum\limits_{i\neq j=1}^{m} |a_{ij}| < |a_{ii}|$ або $\sum\limits_{j\neq i=1}^{m} |a_{ij}| \leqslant |a_{jj}|$.

У цьому випадку легко можна перейти від системи вигляду (8.1) до системи (8.5). Для цього розділимо i-е рівняння системи на a_{ii} і виразимо x_i:

$$x_i = \frac{f_i}{a_{ii}} - \frac{a_{11}}{a_{ii}} x_1 - \cdots - \frac{a_{1n}}{a_{ii}} x_n,$$

тобто для матриці P буде виконано одну з умов збіжності, де

$$P = \begin{pmatrix} 0 & -\frac{a_{12}}{a_{11}} & \dots & -\frac{a_{1n}}{a_{11}} \\ -\frac{a_{21}}{a_{22}} & 0 & \dots & -\frac{a_{2n}}{a_{22}} \\ \dots & \dots & \dots & \dots \\ -\frac{a_{n1}}{a_{nn}} & -\frac{a_{n2}}{a_{nn}} & \dots & 0 \end{pmatrix}$$

Приклад реалізації методу простої ітерації засобами **Maxima** з виведенням проміжних результатів наведено у скрипті нижче:

```
(%i1) iterpr(a0,b0,x,n,eps) :=block([a,b,x0,i,j,s,sum,p],
      a:copymatrix(a0), b:copymatrix(b0), x0:copymatrix(x),
      sum:1, p:0,
      while sum>eps do (
        sum:0, p:p+1, print("p= ",p," x= ",float(x)),
        for i:1 thru n do (
          s:b[i,1],
          for j:1 thru n do (s:s-a[i,j]*x0[j,1]),
            s:s/a[i,i], x[i,1]:x0[i,1]+s, sum:sum+abs(s)
        ),
        x0:copymatrix(x)
      ),
      float(x))$
```

8.4.3 Метод Зейделя

У методі Зейделя система (8.1) також приводиться до системи (8.5). Але при обчисленні наступного компонента вектора використовуються вже обчислені компоненти цього вектора.

Ітераційна формула методу у скалярній формі записується у такий спосіб:

$$x_i^{(k+1)} = \sum_{j=1}^{i} p_{ij} x_i^{(k+1)} + \sum_{j=i+1}^{m} p_{ij} x_j^k + g_i.$$

Встановимо зв'язок між методом Зейделя і методом простої ітерації. Для цього матрицю B представимо у вигляді суми двох матриць: $P = H + F$, де

$$H = \begin{pmatrix} 0 & 0 & 0 & \dots & 0 \\ p_{21} & 0 & 0 & \dots & 0 \\ p_{31} & p_{32} & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ p_{m1} & p_{m2} & p_{m3} & \dots & 0 \end{pmatrix}; \quad F = \begin{pmatrix} p_{11} & p_{12} & p_{13} & \dots & p_{1m} \\ 0 & p_{22} & p_{23} & \dots & p_{2m} \\ 0 & 0 & p_{33} & \dots & p_{3m} \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & p_{mm} \end{pmatrix}.$$

Ітераційна формула методу Зейделя у матричній формі записується у такому вигляді:

$$\bar{x}^{(k+1)} = H\bar{x}^{(x+1)} + F\bar{x}^{(k)} + \bar{g}, \text{ або } (I - H)\bar{x}^{(k+1)} = F\bar{x}^{(k)} + \bar{g},$$

звідки

$$\bar{x}^{(k+1)} = (I - H)^{-1}F\bar{x}^{(k)} + (I - H)^{-1}\bar{g},$$

тобто метод Зейделя еквівалентний методу простої ітерації з матрицею $(I - H)^{-1}F$.

Виходячи з отриманої аналогії методів Зейделя і простої ітерації, можна сформулювати таку ознаку збіжності методу Зейделя: для того, щоб метод Зейделя збігався, необхідно і достатньо, щоб всі власні значення матриці $(I - H)^{-1}F$ за модулем були меншими за одиницю.

Інакше кажучи, щоб метод Зейделя збігався, необхідно і достатньо, щоб усі корені характеристичного рівняння за модулем були меншими за одиницю, оскільки

$$\left|(I - H)^{-1}F - \lambda I\right| = \left|(I - H)^{-1}(I - H)\left[(I - H)^{-1}F - \lambda I\right]\right| = \left|(I - H)^{-1}\right| |F + \lambda H - \lambda I| = |F + \lambda H - \lambda I| = 0.$$

Сформулюємо достатню ознаку збіжності: для того, щоб метод Зейделя збігався, достатньо, щоб виконувалося одна з умов:

- $||P||_1 = \max\limits_i \sum\limits_{j=1}^{m} |p_{ij}| < 1$;
- $||P||_1 = \max\limits_j \sum\limits_{i=1}^{m} |p_{ij}| < 1$;
- $||P||_3 = \sqrt{\sum\limits_{i,j=1}^{m} |p_{ij}|^2} < 1$.

При використанні методу Зейделя ітераційний процес збігається до єдиного розв'язку швидше методу простих ітерацій. Приклад реалізації методу Зейделя:

```
(%i1) seidel(a0,b0,x,n,eps):=block([a,b,i,j,s,sum,p],
      a:copymatrix(a0), b:copymatrix(b0),
      sum:1, p:0,
      while sum>eps do (
        sum:0, p:p+1, print("p= ",p),
        for i:1 thru n do
        (
          s:b[i,1],
          for j:1 thru n do (s:s-a[i,j]*x[j,1]),
          s:s/a[i,i], x[i,1]:x[i,1]+s, sum:sum+abs(s)
        )
      ),
      float(x))$
```

Приклад розв'язання простої системи методом Зейделя:

```
(%i2) aa:matrix([3,1,1],[1,3,1],[1,1,3]); bb:matrix([6],[6],[8]); x:matrix([3],[3],[3]);
      zz:seidel(aa,bb,x,3,0.0000001);
```

(%o2) $\begin{pmatrix} 3 & 1 & 1 \\ 1 & 3 & 1 \\ 1 & 1 & 3 \end{pmatrix}$

(%o3) $\begin{pmatrix} 6 \\ 6 \\ 8 \end{pmatrix}$

(%o4) $\begin{pmatrix} 3 \\ 3 \\ 3 \end{pmatrix}$

$p = 1$ $p = 2$ $p = 3$ $p = 4$ $p = 5$ $p = 6$ $p = 7$ $p = 8$ $p = 9$ $p = 10$ $p = 11$ $p = 12$

(%o5) $\begin{pmatrix} 1.000000000753427 \\ 0.99999999214211 \\ 2.000000002368154 \end{pmatrix}$

8.5 Розв'язування звичайних диференціальних рівнянь

8.5.1 Методи розв'язування задачі Коші

Серед задач, з якими доводиться мати справу в обчислювальній практиці, значну частину складають різні задачі, що зводяться до розв'язування звичайних диференціальних рівнянь. Звичайно доводиться вдаватися до допомоги наближених методів розв'язання подібних задач. У випадку звичайних диференціальних рівнянь залежно від того, чи ставляться додаткові умови в одній або декількох точках відрізка зміни незалежної змінної, задачі звичайно поділяються на одноточкові (задачі з початковими умовами або задачі Коші) і багатоточкові. Серед багатоточкових задач найчастіше у прикладних питаннях зустрічаються так звані граничні задачі, коли додаткові умови ставляться на кінцях розглянутого відрізка.

Надалі обмежимося розглядом обчислювальних методів розв'язання задачі Коші. Для простоти викладу методів розв'язання задачі будемо розглядати випадок одного звичайного диференціального рівняння першого порядку.

Нехай на відрізку $x_0 \leqslant x \leqslant L$ потрібно знайти розв'язок $y(x)$ диференціального рівняння

$$y' = f(x, y), \tag{8.6}$$

що задовольняє при $x = x_0$ початковій умові $y(x_0) = y_0$.

Будемо вважати, що умови існування і єдиності розв'язку поставленої задачі Коші виконані.

На практиці знайти загальний або частинний розв'язок задачі Коші вдається для досить обмеженого кола задач, тому доводиться розв'язувати цю задачу приблизно.

Відрізок $[x_0, L]$ вкривається сіткою (розбивається на інтервали) найчастіше з постійним кроком h ($h = x_{n+1} - x_n$), і за якимось правилом розв'язування знаходиться значення $y_{n+1} = y(x_{n+1})$. Таким чином, результатом розв'язання задачі Коші обчислювальними методами є таблиця, що складається із двох векторів: $x = (x_0, x_1, \ldots, x_n)$ – вектора аргументів і відповідного йому вектора значень шуканої функції $y = (y_0, y_1, \ldots, y_n)$.

Обчислювальні методи (правила), у яких для знаходження значення функції у новій точці використовується інформація тільки щодо однієї (попередньої) точки, називаються однокроковими.

Обчислювальні методи (правила), у яких для знаходження значення функції у новій точці використовується інформація щодо декількох (попередніх) точок, називаються багатокроковими.

Із загального курсу звичайних диференціальних рівнянь широкого поширення набув аналітичний метод, заснований на ідеї розкладу у ряд розв'язку розглянутої задачі Коші. Особливо часто для цих цілей використовується ряд Тейлора. У цьому випадку обчислювальні правила будуються особливо просто.

Наближений розв'язок $y_m(x)$ початкової задачі шукають у вигляді

$$y_m(x) = \sum_{i=0}^{m} \frac{(x - x_0)^i}{i!} y^{(i)}(x_0), \quad x_0 \leqslant x \leqslant L. \tag{8.7}$$

Тут $y^{(0)}(x_0) = y(x_0)$, $y^{(1)}(x_0) = y'(x_0) = f(x_0, y_0)$, а значення $y^{(i)}(x_0)$, $i = 2, 3, ..., m$ знаходять за формулами, отриманим послідовним диференціюванням заданого рівняння:

$$y^{(2)}(x_0) = y''(x_0) = f_x(x_0, y_0) + f(x_0, y_0) f_y(x_0, y_0); \tag{8.8}$$

$$y^{(3)}(x_0) = y'''(x_0) = f_{x^2}(x_0, y_0) + 2f(x_0, y_0) f_y(x_0, y_0) + \tag{8.9}$$

$$+ f^2(x_0, y_0) f_{y^2}(x_0, y_0) + f_y(x_0, y_0) \left[f_x(x_0, y_0) + f(x_0, y_0) f_y(x_0, y_0) \right]; \tag{8.10}$$

$$\tag{8.11}$$

$$y^{(m)}(x_0) = F_m \left(f; f_x; f_{x^2}; f_{xy}; f_{y^2}; \ldots f_{x^{m-1}}; f_{y^{m-1}} \right) \Big|_{x=x_0, y=y_0}. \tag{8.12}$$

Для значень x, близьких до x_0, метод рядів (8.7) при досить великому значенні m зазвичай дає добре наближення точного розв'язку $y(x)$ задачі (8.6). Однак з ростом відстані $|x - x_0|$ похибка наближення шуканої функції рядом зростає за абсолютною величиною (при тій самій кількості членів ряду), і правило (8.7) стає зовсім неприйнятним, коли x виходить із області збіжності відповідного ряду (8.7) Тейлора.

Якщо у виразі (8.7) обмежитися $m = 1$, то для обчислення нових значень $y(x)$ немає потреби переобчислювати значення похідної, правда, і точність розв'язку буде невисока.

При використанні системи комп'ютерної алгебри природнішим виглядає метод послідовних наближень Пікара.

Розгляньмо інтегрування одиничного диференціального рівняння $\dfrac{dy}{dx} = f(x, y)$ на відрізку $[x_0, x]$ з початковою умовою $y(x_0) = y_0$. За формального інтегрування отримаємо:

$$\int_{x_0}^{x} \frac{dy}{dx} dx = \int_{x_0}^{x} f(x, y) dx.$$

Процедура послідовних наближень методу Пікара реалізується відповідно до такої схеми:

$$y_{n+1}(x) - y(x_0) = \int_{x_0}^{x} f(x, y_n(x)) dx.$$

Як приклад розглянемо розв'язання рівняння $y' = -y$ при $y(0) = 1$, $x_0 = 0$:

```
(%i1) rp:-y$ y0:1$ x0:0$ /*rp - права частина рівняння */
(%i4) y1:y0+integrate(subst(y0,y,rp),x,x0,x);
 /* y1 -- перше наближення */
(%o4) 1 − x
(%i5) y2:y0+integrate(subst(y1,y,rp),x,x0,x);
 /* y2 -- друге наближення */
(%o5) x^2−2x/2 + 1
(%i6) y3:y0+integrate(subst(y2,y,rp),x,x0,x);
(%o6) 1 − x^3−3x^2+6x/6
(%i7) expand(%); /* Очевидним розв'язком розглянутого ЗДР є
                    експонента y=exp(-x).
                    У результаті використання методу Пікара
                    отримуємо розв'язок у вигляді ряду Тейлора */
(%o7) −x^3/6 + x^2/2 − x + 1
```

8.5.2 Метод рядів, що не потребує обчислення похідних правої частини рівняння

Природно поставити задачу про таке вдосконалення наведеного вище однокрокового методу, яке б зберігало основні його переваги, але не було б пов'язане зі знаходженням значень похідних правої частини рівняння

$$y_m(x_{n+1}) \approx \sum_{i=0}^{m} \frac{h^i}{i!} y^{(i)}(x_n), \tag{8.13}$$

де $x_{n+1} = x_n + h$.

Щоб виконати останню умову, похідні $y^{(i)}(x)$, $i = 2, 3, \ldots, m$, що входять у праву частину рівняння (8.13), можна замінити за формулами обчислювального диференціювання їхніми наближеними виразами через значення функції y' і врахувати, що $y'(x) = f[x, y(x)]$.

8.5.2.1 Метод Ойлера

У випадку $m = 1$ наближена рівність (8.13) не вимагає обчислення похідних правої частини рівняння і надає змогу з погрішністю порядку h^2 знаходити значення $y(x_n + h)$ розв'язки цього рівняння за відомим його значенням $y(x_n)$. Відповідне однокрокове правило можна записати у вигляді

$$y_{n+1} = y_n + hf_n. \tag{8.14}$$

Це правило (8.14) уперше було побудовано Ойлером, отже носить його ім'я. Іноді його називають також правилом ламаних або методом дотичних. Метод Ойлера має відносно низький порядок точності – h^2 на одному кроці. Практична оцінка погрішності наближеного розв'язку може бути отримана за правилом Рунґе.

Приклад реалізації методу Ойлера засобами **Maxima** наведено нижче:

```
(%i1) euler1(rp,fun,y0,x0,xend,h):=block([OK,_x,_y,_y1,rez],
         _x:x0, _y:y0, rez:[_y], OK:-1, eps:0.1e-7,
         while OK<0 do (
           if ((_x+h>xend) or (abs(_x+h-xend)<eps))
             then (h:xend-_x,_x:xend, OK:1)
             else (_x:_x+h),
             _y1:makelist (float(_y[i]+h*subst([fun[i]=_y [i] ,x=_x] .
               rp[i])),i,1,length(_y)),rez:append(rez, [_y1]),
             -y:_y1
           ),
           rez
         )$
```

Праві частини розв'язуваних диференціальних рівнянь передаються до функцію `eulerl` у списку `rp`. Типово вважається, що список назв залежних змінних – `fun`, назва незалежної змінної – `x`. Початкові значення незалежної і залежних змінних – список `y0` і скалярна величина `x0`, границя інтервалу інтегрування — величина `xend`, крок інтегрування – `h`.

Наступний приклад – виклик функції `eulerl`. Наведено розв'язання системи із трьох диференціальних рівнянь на інтервалі $[0, l]$ із кроком $h = 0.1$:

$$\begin{cases} \dfrac{dy}{dx} = -2y, \\ \dfrac{dv}{dx} = -5v, \\ \dfrac{dz}{dx} = 3x. \end{cases}$$

З початковими умовами $y(0) = l.0$; $v(0) = 1.0$; $z(0) = 0$ розв'язком рівнянь цієї системи будуть такі функції:

$$\begin{cases} y(x) = e^{-2x}, \\ v(x) = e^{-5x}, z(x) = \dfrac{3}{2}x^2. \end{cases}$$

```
(%i2) euler1([-2*y,-5*v,3*x],[y,v,z],[1,1,0],0,1,0.1);
(%o2) [[1,1,0], 0.8,0.5,0.03],[0.64,0.25,0.09],[0.512,0.125,0.18],
[0.4096,0.0625, 0.3], [0.32768,0.03125,0.45], [0.262144,0.015625,0.63],
[0.2097152,0.0078125,0.84], [0.16777216,0.00390625,1.08],
[0.134217728, 0.001953125, 1.35], [0.1073741824, 9.7656249999999913 10^-4,1.65]]
```

Перевірити розв'язання можна порівнюючи графіки точного розв'язку і множини обчислених наближених значень. Приклад послідовності команд, що дозволяють виділити окремі компоненти розв'язку системи ЗДР і побудувати графік точного і наближеного розв'язку третього рівняння системи, представлений нижче (точні розв'язки – списки `yf`, `vf`, `zf`; наближені розв'язки – списки `yr`, `vr`, `zr`; список значень незалежної змінної – `xg`).

```
(%i3) rez:euler1([-2*y,-5*v,3*x],[y,v,z], [1,1,0],0,1.0,0.1)$
      n:length(rez)$
      yr:makelist(rez[k][1],k,1,n)$
      vr:makelist(rez[k][2],k,1,n)$
      zr:makelist(rez[k][3],k,1,n)$
      xg:makelist(0.1*(k-1),k,1,n)$
      yf:makelist(exp(-2*xg[k]),k,1,n)$
      vf:makelist(exp(-5*xg[k]),k,1,n)$
      zf:makelist(3*xg[k]^2/2,k,1,n)$
      plot2d([[discrete,xg,zr],[discrete,xg,zf]],[style,points,lines])$
```

Зменшення кроку h призводить до зменшення погрішності розв'язку (у наведеному прикладі – крок 0.1).

8.5.2.2 Метод Рунґе-Кутти

Ідею методу викладемо на прикладі задачі Коші:

$$y' = f(x, y); x_0 \leqslant x \leqslant b; y(x_0) = y_0.$$

Інтегруючи це рівняння у межах від x до $x + h$ $(0 < h < 1)$, отримаємо рівність

$$y(x + h) = y(x) + \int_x^{x+h} f\,[t, y(t)]\,dt, \tag{8.15}$$

яка за допомогою останнього інтеграла пов'язує значення розв'язку розглянутого рівняння у двох точках, віддалених одна від одної на відстань кроку h.

Для зручності запису виразу (8.15) використаємо позначення $\Delta y = y(x + h) - y(x)$ і заміну змінної інтегрування $t = x + h$.

Остаточно отримаємо:

$$\Delta y = h \int_0^1 f\,[x + \alpha h, y(x + \alpha h)]\,d\alpha. \tag{8.16}$$

Залежно від способу обчислення інтеграла у виразі (8.16) отримують різні методи обчислювального інтегрування звичайних диференціальних рівнянь.

Розгляньмо лінійну комбінацію величин φ_i, $i = 0, 1, \ldots, q$, що буде аналогом квадратурної суми і надасть змогу обчислити наближене значення приросту Δy:

$$\Delta y \approx \sum_{i=0}^{q} a_i \varphi_i,$$

де

$$\begin{aligned} &\varphi_0 = hf(x, y); \\ &\varphi_1 = hf(x + \alpha_1 h; y + \beta_{10}\varphi_0); \\ &\varphi_2 = hf(x + \alpha_2 h; y + \beta_{20}\varphi_0 + \beta_{21}\varphi_1); \\ &\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \end{aligned}$$

Метод четвертого порядку для $q = 3$, що є аналогом широко відомої в літературі чотириточкової квадратурної формули «трьох восьмих», має вигляд

$$\Delta y \approx \frac{1}{8}\left(\varphi_0 + 3\varphi_1 + 3\varphi_2 + \varphi_3\right),$$

де

$$\varphi_0 = hf(x_n, y_n);$$
$$\varphi_1 = hf(x_n + \frac{h}{3}; y_n + \frac{\varphi_0}{3});$$
$$\varphi_2 = hf(x_n + \frac{2}{3}h; y_n - \frac{\varphi_0}{3} - \varphi_1);$$
$$\varphi_3 = hf(x_n + h; y_n + \varphi_0 - \varphi_1 + \varphi_2).$$

Особливо широко відомо інше обчислювальне правило типу Рунґе-Кутти четвертого порядку точності:

$$\Delta y = \frac{1}{6}\left(\varphi_0 + 2\varphi_1 + 2\varphi_2 + \varphi_3\right),$$

де

$$\varphi_0 = hf(x_n, y_n);$$
$$\varphi_1 = hf(x_n + \frac{h}{2}; y_n + \frac{\varphi_0}{2});$$
$$\varphi_2 = hf(x_n + \frac{h}{2}; y_n + \frac{\varphi_1}{2});$$
$$\varphi_3 = hf(x_n + h; y_n + \varphi_2).$$

Метод Рунґе-Кутти має погрішність четвертого порядку ($\sim h^4$).

Функцію **Maxima**, яка реалізує метод Рунґе-Кутти 4-го порядку, наведено у наступному прикладі (із друком проміжних результатів):

```
(%i1) rk4(rp,fun,y0,x0,xend,h):=block(
         [OK,n,h1,_x,_y,_k1,_k2,_k3,_k4,rez],
         _x:x0,_y:y0,rez:[_y],OK:-1,h1:h,n:length(_y),
         while OK<0 do (
           if (_x+h1>=xend) then (h1:xend-_x, OK:1),
           _k1:makelist(float(h1*subst([fun[i]=
             float(_y[i]),x=float(_x)],rp[i])),i,1,n),
           _k2:makelist(float(h1*subst([fun[i]=
             float(_y[i]+_k1[i]/2),x=float(_x+h1/2)],
                   rp[i])),i,1,n),
           _k3:makelist(float(h1*subst([fun[i]=
             float(_y[i]+_k2[i]/2),x=float(_x+h1/2)],
                   rp[i])),i,1,n),
           _k4:makelist(float(h1*subst([fun[i]=
             float(_y[i]+_k3[i]),x=float(_x+h1)],rp[i])),i,1,n),
           _y1:makelist(float(_y[i]+
             (_k1[i]+2*_k2[i]+2*_k3[i]+_k4[i])/6),i,1,n),
           rez:append(rez,[_y1]),
           print("x= ",_x, " y= ",_y),
           _x:_x+h1,
           _y:_y1
         ), rez
       )$
```

Приклад виклику функції `rk4` представлений наступною послідовністю команд (розв'язуваласяя та ж система, що і при тестуванні методу Ойлера):

```
(%i2) rk4([-2*y,-5*v,3*x],[y,v,z],[1,1,1],0,1,0.1);
x= 0 y= [1,1,1]
x= 0.1 y= [0.81873333333333,0.60677083333333,1.015]
x= 0.2 y= [0.67032427111111,0.36817084418403,1.06]
x= 0.3 y= [0.54881682490104,0.22339532993458,1.135]
x= 0.4 y= [0.44933462844064,0.13554977050718,1.24]
x= 0.5 y= [0.3678852381253,0.082247647208783,1.375]
x= 0.6 y= [0.30119990729446,0.04990547343658,1.54]
x= 0.7 y= [0.24660240409888,0.030281185705008,1.735]
x= 0.8 y= [0.20190160831589,0.018373740284549,1.96]
x= 0.9 y= [0.16530357678183,0.011148649703906,2.215]
x= 1.0 y= [0.13533954843051,0.0067646754713805,2.5]
```

Додатки

Табл. 8.1. Короткий список основних функцій **Maxima**

Функція або змінна	**Короткий опис**
’, ’’, , %	найпростіші команди, див. стор. 16
addcol	Функція додає стовпець до матриці, див. стор. 23-26
addrow	Функція додає рядок до матриці, див. стор. 23-26
algsys	Функція розв’язує поліноміальні системи рівнянь. Допускаються системи з одного рівняння з однією невідомою. Крім того, допускаються недовизначені системи, див. стор. 45
allroots	Функція, що знаходить і друкує всі (у тому числі і комплексні) корінь поліноміального рівняння з дійсними або комплексними коефіцієнтами, див. стор. 44-47
antidiff	Функція виконує інтегрування виразів із довільними функціями, перед її першим викликом варто завантажити пакет antid, див. стор. 65-68
append	Функція надає змогу склеювати два списки, див. стор. 19
arrayinfo	Функція виводить інформацію про масив — його тип, кількість індексів, розмір, див. стор. 21
arrays	Змінна містить список назв масивів першого і другого типів, визначених на даний момент, див. стор. 22
array	Функція визначає масив з даною назвою, визначеною кількістю індексів і заданим розміром, див. стор. 21
assume	Функція вводить інформацію про змінну до бази даних, див. стор. 26-29
atom	Функція повертає true, якщо аргумент не має структури, тобто складових частин (наприклад, число або змінна не мають структури)
atvalue	Функція надає змогу задати значення функції і її похідних при деяких значеннях аргументів, див. стор. 82
at	Функція обчислює значення виразу у заданій точці з урахуванням властивості, див. стор. 82
augmented_lagrangian_method	Функція здійснює мінімізацію цільової функції із обмеженнями, див. стор. 107
batch	Функція запускає файл із програмою. Оператори виконуються один за одним або до кінця файлу, або до синтаксичної помилки, або до некоректної операції, див. стор. 104
bc2	Функція надає змогу врахувати крайові умови у розв’язках диференціальних рівнянь другого порядку, див. стор. 81
cabs	Функція повертає модуль комплексного виразу, див. стор. 39
carg	Функція повертає аргументу комплексного виразу, див. стор. 39
cfdisrep	Функція перетворить список (як правило результат виконання функції cf) у властиво ланцюговий дріб, див. стор. 75
cf	Функція створює ланцюговий дріб, що апроксимує даний вираз. Вираз має складатися із цілих чисел, квадратних коренів цілих чисел і знаків арифметичних операцій. Результат, що повертається – список, див. стор. 75
cfdirep	Функція перетворить список у власне ланцюговий дріб, див. стор. 75
changevar	Реалізує заміну змінних в інтегралі, див. стор. 65-68
charpoly	Функція є певною мірою надлишковою — вона обчислює характеристичний поліном матриці (корені цього полінома — власні значення матриці), див. стор. 42

Продовження на наступній сторінці

Табл. 8.1 – *Продовження*

Функція або змінна	**Короткий опис**
`closefile`	Функція припиняє виведення до файла, див. стор. 104
`col`	Функція виокремлює заданий стовпець матриці, див. стор. 25
`combine`	Функція поєднує доданки з ідентичним знаменником, див. стор. 28
`compile`	Функція спочатку транслює функцію **Maxima** мовою **Lisp**, а потім компілює цю функцію **Lisp** до двійкових кодів і завантажує їх до пам'яті, див. стор. 101
`conjugate`	Функція для обчислення комплексно спряжених виразів, див. стор. 39
`cons`	Функція надає змогу додавати елемент на початок списку, див. стор. 19
`contrib_ode`	Функція розв'язує диференціальні рівняння (більше можливостей, ніж в `ode2`), див. стор. 83
`copylist`	Функція створює копію списку, див. стор. 19
`create_list`	Функція створює список, див. стор. 19
`copymatrix`	Функція створює копію матриці, див. стор. 25
`cspline`	Функція будує сплайн-інтерполяцію, див. стор. 106
`define`	Функція надає змогу перетворити вираз на функцію, див. стор. 32
`demoivre`	Функція заміняє все експоненти із уявними показниками на відповідні тригонометричні функції, див. стор. 39
`denom`	Функція виокремлює знаменник виразу, див. стор. 29
`depends`	Функція надає змогу оголошувати, що змінна залежить від однієї або декількох інших змінних, див. стор. 64
`desolve`	Функція розв'язує диференціальні рівняння і системи диференціальних рівнянь методом перетворення Лапласа, див. стор. 81
`determinant`	Функція обчислює детермінант матриці, див. стор. 42
`diff`	Функція виконує диференціювання, див. стор. 54
`display2d`	Змінна вмикає або вимикає «двовимірне» малювання дробів, степенів тощо. Від початку встановлене значення `true`, див. стор. 103
`display`	Функція виводить значення своїх аргументів разом з їхньою назвою, кожен в окремому рядку, див. стор. 102
`disp`	Функція виводить значення своїх аргументів, причому кожне значення виводиться в окремому рядку, див. стор. 102
`divide`	Функція надає змогу обчислити частку і остачу від ділення одного многочлена на інший, див. стор. 29
`draw2d`	будує двовимірні графіки, див. стор. 118
`draw3d`	будує тривимірні графіки, див. стор. 118
`echelon`	Функція перетворить матрицю до верхньої трикутної, див. стор. 43
`eigenvalues`	Функція аналітично обчислює власні значення матриці, див. стор. 42
`eigenvectors`	Функція аналітично обчислює власні значення і власні вектори матриці, якщо це можливо, див. стор. 42
`eliminate`	Функція виключає із системи рівнянь зазначені змінні. Рівняння, що залишилися, приводяться до вигляду з нульовою правою частиною, яка опускається, див. стор. 47
`endcons`	Функція надає змогу додавати елемент у кінець списку, див. стор. 19
`ev`	Функція є основною функцією, що обробляє вираз, див. стор. 27
`expand`	Функція розкриває дужки, див. стор. 27
`exponentialize`	Функція приводить комплексний вираз до експоненціальної форми, див. стор. 39
`express`	Функція перетворює диференціальні оператори у вираз, див. стор. 64
`factor`	Функція представляє у вигляді добутку деяких співмножників заданий вираз, див. стор. 28
`factorsum`	Функція факторизує окремі доданки у виразі, див. стор. 28
`fillarray`	Функція надає змогу заповнювати одноіндексні масиви третього типу зі списку, див. стор. 22
`find_root`	Функція знаходить корінь рівняння на заданому інтервалі методом ділення відрізка навпіл, див. стор. 104
`first`	Функція виділяє перший елемент списку, див. стор. 20

Продовження на наступній сторінці

Табл. 8.1 – *Продовження*

Функція або змінна	**Короткий опис**
`float`	Функція перетворює будь-які числа у виразах у числа машинної точності, див. стор. 17
`fourier`	Функція надає змогу обчислити коефіцієнти ряду Фур'є, див. стор. 94
`foursimp`	Функція надає змогу спростити коефіцієнти ряду Фур'є, див. стор. 94
`fullratsimp`	Функція викликає функцію `ratsimp` доти, доки вираз не перестане мінятися, див. стор. 30
`genmatrix`	Функція повертає матрицю заданої розмірності, складену з елементів індексного масиву, див. стор. 24
`gfactorsum`	Функція представляє у вигляді співмножників доданки у сумах із комплексних виразів, див. стор. 28
`gfactor`	Функція представляє у вигляді співмножників вираз з комплексними числами, див. стор. 28
`gradef`	Функція визначає результат диференціювання функції за своїми аргументами, див. стор. 64
`gramschmidt`	Функція обчислює ортонормовану систему векторів, див. стор. 42
`ic1`	Функція надає змогу врахувати початкову умову у розв'язках диференціальних рівнянь першого порядку, див. стор. 76
`ic2`	Функція надає змогу врахувати початкові умови в розв'язках диференціальних рівнянь другого порядку, див. стор. 76
`ident`	Функція повертає одиничну матрицю заданої розмірності, див. стор. 25
`ilt`	Функція реалізує обернене перетворення Лапласа, див. стор. 68
`imagpart`	Функція повертає дійсну частину виразу, див. стор. 39
`integrate`	Функція виконує інтегрування заданого виразу за вказаною змінною (довільна стала не додається). Можна також указати межі інтегрування — у цьому випадку обчислюється визначений інтеграл, див. стор. 65-68
`invert`	Функція виконує обернення матриці, див. стор. 42
`join`	Функція виконує компонування списків, див. стор. 19
`kill`	Функція знищує всю інформацію (як властивості, так і пов'язане значення) про об'єкт або декілька об'єктів, див. стор. 18
`lagrange`	Функція будує інтерполяцію поліномом Лагранжа, див. стор. 106
`lambda`	створює лямбда-вираз (безіменну функцію). Лямбда-вираз може використовуватися у деяких випадках як звичайна функція, див. стор. 100
`laplace`	Функція реалізує пряме перетворення Лапласа, див. стор. 68
`last`	Функція виокремлює останній елемент списку, див. стор. 20
`lbfgs`	Функція здійснює мінімізацію цільової функції, див. стор. 106
`length`	Функція повертає довжину списку, див. стор. 19
`lhs`	Функція виділяє ліву частину рівняння, див. стор. 46
`limit`	Функція здійснює обчислення границь, див. стор. 49
`linearinterpol`	Функція будує лінійну інтерполяцію, див. стор. 105
`linsolve`	Функція розв'язує системи лінійних і поліноміальних рівнянь. Допускаються недовизначені системи, див. стор. 45
`listarray`	Функція виводить вміст масивів першого і другого типів, див. стор. 21
`load`	Функція завантажує той або інший файл: load(somefile); тип завантаження залежить від типу файла (макрос **Maxima**, програма на **Lisp**, бінарний файл), див. стор. 103
`logcontract`	Функція ущільнює логарифми у заданому виразі, див. стор. 32
`make_array`	Функція створює масиви третього типу, вміст яких виводиться автоматично, див. стор. 22
`makelist`	Функція надає змогу створювати списки, див. стор. 19
`map`	Функція застосовує задану функцію до кожного елемента списку, див. стор. 20
`matrix`	Функція повертає матрицю, задану поелементно, див. стор. 23
`matrixmap`	Функція для заповнення матриці значеннями деякої функції, див. стор. 26

Продовження на наступній сторінці

Табл. 8.1 – *Продовження*

Функція або змінна	**Короткий опис**
`max`	перебирає свої аргументи і знаходить максимальне число, див. стор. 26
`member`	Функція повертає `true`, якщо її перший аргумент є елементом заданого списку, і `false` у протилежному випадку, див. стор. 20
`min`	перебирає свої аргументи і знаходить мінімальне число, див. стор. 26
`minor`	Обчислює мінори матриці, див. стор. 44
`mnewton`	Функція знаходить корінь системи рівнянь багатовимірним методом Ньютона. Для використання функції необхідно спочатку завантажити пакет `mnewton`, див. стор. 105
`multthru`	Функція множить кожний доданок у сумі на множник, причому при множенні дужки у виразі не розкриваються, див. стор. 29
`newton`	Функція знаходить корінь зазначеної функції методом Ньютона, див. стор. 104
`nroots`	Функція, що повертає кількість дійсних корінь поліноміального рівняння з дійсними коефіцієнтами, які локалізовані у зазначеному інтервалі, див. стор. 46
`num`	Функція виокремлює чисельник, див. стор. 29
`ode2`	Функція розв'язує диференціальні рівняння першого і другого порядків, див. стор. 76
`odelin`	Функція розв'язує однорідні лінійні рівняння першого і другого порядку, і повертає фундаментальне розв'язки ЗДР див. стор. 84
`pade`	Функція апроксимує ланку ряду Тейлора дробово-раціональною функцією, див. стор. 74
`plog`	Представляє основну гілку комплексного логарифма, див. стор. 39
`plot2d`, `wxplot2d`	будує двовимірні графіки, див. стор. 34
`plot3d`, `wxplot3d`	будує тривимірні графіки, див. стор. 34
`polarform`	Функція приводить комплексний вираз до тригонометричної форми, див. стор. 39
`polyfactor`	Змінна визначає форму виведення даних функцією `allroots`, див. стор. 46
`powerseries`	Функція будує розклад у степеневий ряд, див. стор. 71
`print`	виводить значення усіх своїх аргументів в один рядок, див. стор. 103
`product`	Функція реалізує цикл множення, див. стор. 20
`properties`	Функція виводить властивості змінної, див. стор. 65-68
`radcan`	Функція спрощує вираз із вкладеними степенями і логарифмами, див. стор. 32
`ratexpand`	Функція розкриває дужки у виразі. Відрізняється від функції `expand` тим, що приводить вираз до канонічної форми, див. стор. 30
`ratfac`	Змінна включає або виключає часткову факторизацію виразів при зведенні їх до CRE. Від початку встановлене значення `false`, див. стор. 30
`ratsimpexpons`	Змінна управляє спрощенням показників степеня у виразах, див. стор. 30
`ratsimp`	Функція приводить всі фрагменти (зокрема аргументи функцій) виразу, що не є дробово-раціональною функцією, до канонічного подання, виконуючи спрощення, які не виконує функція `rat`. Повторний виклик функції може змінити результат, тобто спрощення не виконується до кінця, див. стор. 30
`ratsubst`	Функція Реалізує синтаксичну підстановку для раціональних виразів, див. стор. 31
`ratvars`	Функція надає змогу змінити алфавітний порядок «пріоритету» змінних, прийнятий типово, див. стор. 30
`rat`	Функція приводить вираз до канонічного подання і додає до нього мітку /R/. Вона спрощує будь-який вираз, розглядаючи його як дробово-раціональну функцію, тобто працює з арифметичними операціями і з піднесенням до цілого степеня, див. стор. 29
`read`	основна функція для зчитування виразів, що вводять користувачем,, див. стор. 102
`read_matrix`, `read_list`	Функція для введення масивів чисел, див. стор. 125

Продовження на наступній сторінці

Табл. 8.1 – *Продовження*

Функція або змінна	**Короткий опис**
`realpart`	Функція повертає дійсну частину комплексного виразу, див. стор. 39
`realroots`	Функція видає дійсні корені поліноміального рівняння з дійсними коефіцієнтами, див. стор. 46
`rectform`	Функція приводить комплексний вираз до алгебраїчної форми, див. стор. 39
`remarray`	Функція знищує масив або масиви, див. стор. 23
`remove`	Функція вилучає властивість змінної, див. стор. 65-68
`residue`	Функція надає змогу обчислювати лишки на комплексній площині, див. стор. 39
`rest`	Функція виділяє остачу після видалення першого елемента списку, див. стор. 20
`reverse`	Функція змінює порядок елементів у списку на зворотний, див. стор. 20
`rhs`	Функція виокремлює праву частину рівняння, див. стор. 46
`rk`	Функція реалізує метод Рунґе-Кутти розв'язання ЗДР, див. стор. 85
`romberg`	Функція знаходить числове значення визначеного інтеграла функції на заданому відрізку. При цьому використовується алгоритм Ромберга, див. стор. 108
`row`	Функція виокремлює заданий рядок матриці, див. стор. 25
`save`	зберігає поточні значення робочої області у файл, див. стор. 103
`solve`	Функція розв'язує рівняння і системи рівнянь, див. стор. 33
`sublist`	Функція становить список з тих елементів вихідного списку, для яких задана логічна функція повертає значення `true`.
`submatrix`	Функція виділяє з матриці підматрицю, див. стор. 26
`subst`	Функція реалізує синтаксичне підставляння, див. стор. 29
`sum`	Функція реалізує цикл підсумовування, див. стор. 20
`taylor`	Функція повертає розклад функції у ряд Тейлора, див. стор. 70
`tlimit`	Функція відрізняється від функції `limit` тільки алгоритмом – вона використовує розклад виразу у ряд Тейлора, див. стор. 52
`totalfourier`	Функція надає змогу обчислити і побудувати ряд Фур'є див. стор. 95
`translate`	Функція транслює функцію **Maxima** мовою **Lisp**, див. стор. 101
`transpose`	Функція транспонує матрицю, див. стор. 41
`trigexpand`	Змінна керує роботою функції `trigexpand`, див. стор. 31
`trigexpand`	Функція розкладає всі тригонометричні функції від сум у суми добутків тригонометричних функцій, див. стор. 31
`trigreduce`	Функція згортає всі добутки тригонометричних функцій у тригонометричні функції від сум, див. стор. 31
`trigsimp`	Функція лише застосовує до виразу правило $\sin^2 x + \cos^2 x = 1$, див. стор. 31
`trirat`	Функція намагається звести вираз із тригонометричними функціями до універсального канонічного вигляду (загалом, намагається спростити вираз), див. стор. 31
`uniteigenvectors`	Функція відрізняється від функції `eigenvectors` тим, що повертає нормовані на одиницю власні вектори, див. стор. 42
`writefile`	Функція починає запис вихідних даних **Maxima** до зазначеного файла, див. стор. 104
`xthru`	Функція приводить вираз до загального знаменника, не розкриваючи дужок і не факторизуючи складових, див. стор. 28
`zeromatrix`	Функція повертає матрицю заданої розмірності, складену з нулів, див. стор. 25
`'`	Одинарні лапки `'` запобігають обчисленню, див. стор. 16
`''`	Двоє одинарних лапок `''a` викликають додаткове обчислення в момент обробки `a`, див. стор. 16

Табл. 8.2. Перелік основних пакетів розширення **Maxima**

Найменування пакета	**Короткий опис функцій пакета**
`augmented_lagrangian`	Мінімізація функції декількох змінних з обмеженнями методом невизначених множників Лагранжа (використовується разом з lbfgs)
`bode`	Побудова діаграм Боде (вузькоспеціальний пакет)
`contrib_ode`	Додаткові функції для аналітичного розв'язання звичайних диференціальних рівнянь
`descriptive`	Описова статистика, оцінка параметрів розподілу (генеральної сукупності) за вибіркою (див. стор. 125-129)
`diag`	Пакет для операцій з деякими видами діагональних матриць
`distrib`	Пакет, що містить функції для розрахунку різних розподілів імовірностей і їхніх параметрів (нормальний розподіл, розподіл Стьюдента тощо)
`draw`	Інтерфейс **Maxima**-Gnuplot. Призначений для приготування ілюстрацій поліграфічної якості
`dynamics`	Різні функції, зокрема графічні, пов'язані із моделюванням динамічних систем і фракталів
`f90`	Експорт коду **Maxima** у код на Фортран90
`ggf`	Пакет включає єдину функцію, що надає змогу оперувати з твірними функціями послідовностей (вузькоспеціальний пакет)
`graphs`	Пакет, що містить функції для роботи із графами
`grobner`	Функції для того, щоб працювати з базисом Грьобнера (Gröbner)
`impdiff`	обчислення похідних неявних функцій декількох змінних
`implicit_plot`	Графіки неявних функцій
`interpol`	Пакет, що містить функції інтерполяції (лінійної, поліномами Лагранжа, сплайнами)
`lapack`	Функції пакета Lapack для розв'язання задач лінійної алгебри
`lbfgs`	пакет мінімізації функцій декількох змінних квазиньютоновским методом (L-BFGS)
`lindstedt`	Пакет, розрахований на інтерпретацію деяких типів початкових умов для ЗДР, що описує коливання
`lsquares`	Функції для оцінки параметрів різних залежностей методом найменших квадратів (див. стор. 136-136)
`makeOrders`	Пакет включає одну функцію для операцій з поліномами
`mnewton`	Метод Ньютона для розв'язування систем нелінійних рівнянь
`numericalio`	Читання і запис файлів (переважно з матричними числовими даними)
`opsubst`	Пакет містить одну функцію `opsubst`, що надає змогу виконувати заміну у виразах (за можливостями мало відрізняється від `subst`)
`orthopoly`	Пакет, що містить функції для операцій з ортогональними поліномами (Лежандра, Чебишова тощо)
`plotdf`	Пакет, що надає змогу будувати поле напрямків для розв'язування автономних систем (цікавий, але досить вузькоспеціалізований пакет)
`romberg`	Пакет, що містить низку функцій для числового інтегрування
`simplex`	Пакет, призначений для розв'язування задач лінійного програмування
`solve_rec`	Пакет, що містить функції для спрощення рекурентних виразів
`stats`	Пакет, що містить функції для статистичної перевірки гіпотез (про рівність математичних очікувань або дисперсій вибірок тощо — див. стор. 130-134)
`stirling`	Розрахунок гамма-функції
`stringproc`	Пакет, що містить функції для обробки рядків
`unit`	Пакет, що містить функції для операцій з одиницями виміру
`vect`	Пакет, що містить функції для операцій з векторами, обчислення похідних за напрямком, градієнтів, дивергенцій, вихорів
`zeilberger`	Функції для гіпергеометричного підсумовування

Табл. 8.3. Список основних математичних констант, доступних у **Maxima**

Позначення у Maxima	**Математичний зміст**
`%e`	основа натуральних логарифмів
`%i`	уявна одиниця ($\sqrt{-1}$)
`inf`	додатна нескінченність (на дійсній осі)
`minf`	від'ємна нескінченність (на дійсній осі)
`infinite`	нескінченність (на комплексній площині)
`%phi`	Золотий переріз (φ)
`%pi`	Стала π – відношення довжини кола до його діаметра
`%gamma`	Стала Ойлера (γ)
`false`, `true`	логічні (булеві) величини

Табл. 8.4. Список основних математичних функцій, доступних у **Maxima**

Позначення у Maxima	**Математичний зміст**
`abs`	абсолютна величина
`acos`	арккосинус
`acosh`	обернений гіперболічний косинус
`acot`	арккотангенс
`acsc`	арккосеканс
`asec`	арксеканс
`asin`	арксинус
`asinh`	обернений гіперболічний синус
`atan`	арктангенс
`atanh`	обернений гіперболічний тангенс
`ceiling`	округлення до цілого з надлишком
`cos`	косинус
`cosh`	гіперболічний косинус
`cot`	котангенс
`csc`	косеканс
`exp`	експонента
`fix`	ціла частина
`float`	перетворення до формату із рухомою крапкою
`floor`	округлення до цілого з недостачею
`log`	натуральний логарифм
`sec`	секанс
`sin`	синус
`sinh`	гіперболічний синус
`sqrt`	квадратний корінь
`tan`	тангенс
`tanh`	гіперболічний тангенс

Покажчик

арифметичні операції, 17
число
  комплексне
    алгебраїчна форма, 39
    аргумент, 39
    експоненційна форма, 40
    лишки, 40
    модуль, 39
    тригонометрична форма, 39
функція
  апроксимація Паде, 74
    pade, 74
  асимптоти, 62
  функція декількох змінних, 64
    екстремум, 65
    критичні точки, 65
    оператор Лапласа, 64
    векторні оператори, 64
  ланцюговий дріб, 74
    cf, 75
  нескінченно мала, 51
  нескінченно велика, 52
інтерфейс
  Cantor, 115
  emacs, 114
  Sage, 117
  TeXmacs, 114
  wxMaxima, 109
  xMaxima, 112
метод
  Ньютона, 104
    для системи, 105
  Рунґе-Кутти, 85
  ділення навпіл, 155
  інтегрування
    romberg, 108
  інтерполяція
    Лагранжа, 106
    лінійна, 105
    сплайн, 106
  мінімізація
    обмеження, 107
    lbfgs, 106
  статистика
    дисперсії, 132
    гістограма, 126
    медіана, 132
    найменші квадрати, 136
    описова, 125
    порівняння, 131
    регресія, 134
векторний добуток, 65
введення-виведення
  файлове
    batch, 104
    load, 103
    save, 103
  матриці, 125
  у консолі, 102
    disp, 102
    display, 102
    grind, 102
    read, 102

float, 27