МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

О. Ю. Заковоротний, Т. О. Орлова,
Д. В. Гриньов, В. М. Сергієнко

ОСНОВИ КОМП'ЮТЕРНОЇ МАТЕМАТИКИ

Лабораторний практикум
для студентів денної та заочної форм навчання за спеціальністю
123 «Комп’ютерна інженерія»

Харків
НТУ «ХПІ»
2023

УДК 004.89
О 75

Рецензенти:
В. Д. Ковальов, доктор технічних наук, професор, Лауреат премії Кабінету Міністрів України, ректор Донбаської державної машинобудівної академії;
О. А. Серков, доктор технічних наук, професор, заслужений винахідник України, академік Академії наук прикладної радіоелектроніки, НТУ "ХПІ".

Рекомендовано вченою радою Національного технічного університету «Харківський політехнічний інститут» для студентів за спеціальністю 123 «Комп'ютерна інженерія» (протокол № 6 від 23.09.2022 р.)

О 75 Основи комп'ютерної математики : лабораторний практикум / О. Ю. Заковоротний, Т. О. Орлова, Д. В. Гриньов, В. М. Сергієнко. – Харків : НТУ «ХПІ», 2023. – 272 с.

ISBN 978-617-05-0436-4

Наведено відомості про пакет Matlab для освоєння розв'язку задач вищої математики засобами комп'ютерної техніки, оволодіння навичками графічного представлення розв'язків математичних задач, закріплення математичних знань з таких галузей, як лінійна алгебра, векторний та спектральний аналіз, диференціальне та інтегральне числення, диференціальні рівняння.

Призначено для студентів денної та заочної форм навчання за спеціальністю «Комп'ютерна інженерія».

Іл. 28. Табл. 24. Бібліог. 18 назв.

УДК 004.89

ISBN 978-617-05-0436-4

ЗМІСТ

Лабораторна робота 1

ПРОСТІ ОБЧИСЛЕННЯ В ПАКЕТІ MATLAB

Мета лабораторної роботи : отримання і закріплення знань, формування практичних навичок роботи з пакетом MATLAB при обчисленні виразів алгебри з використанням вбудованих математичних функцій.

1.1. Короткі відомості з теорії

1.1.1. Робоче середовище пакету MATLAB

Пакет MATLAB був створений компанією Math Works. Робота багатьох вчених і програмістів спрямована на постійне розширення його можливостей і вдосконалення закладених алгоритмів. Нині MATLAB є потужним і універсальним засобом рішення математичних завдань, що виникають в різних областях людської діяльності.

Робоче середовище MATLAB має зручний інтерфейс для доступу до багатьох допоміжних елементів.

При запуску MATLAB на екрані з'являється робоче середовище, зображене на рис. 1.1.

Робоче середовище містить наступні елементи:

– меню;

– панель інструментів з кнопками і списком, що розкривається;

– вікно з вкладками *Launch Pad* та *Workspace*, з якого можна отримати простий доступ до різних модулів *ToolBox* та до вмісту робочого середовища;

– вікно з вкладками *Command History* та *Current Directory*, призначене для перегляду та повторного виклику раніше введених команд, а також для встановлення поточного каталогу;

– командне вікно *Command Window* з командним рядком, в якому знаходиться миготливий курсор;

– рядок стану.

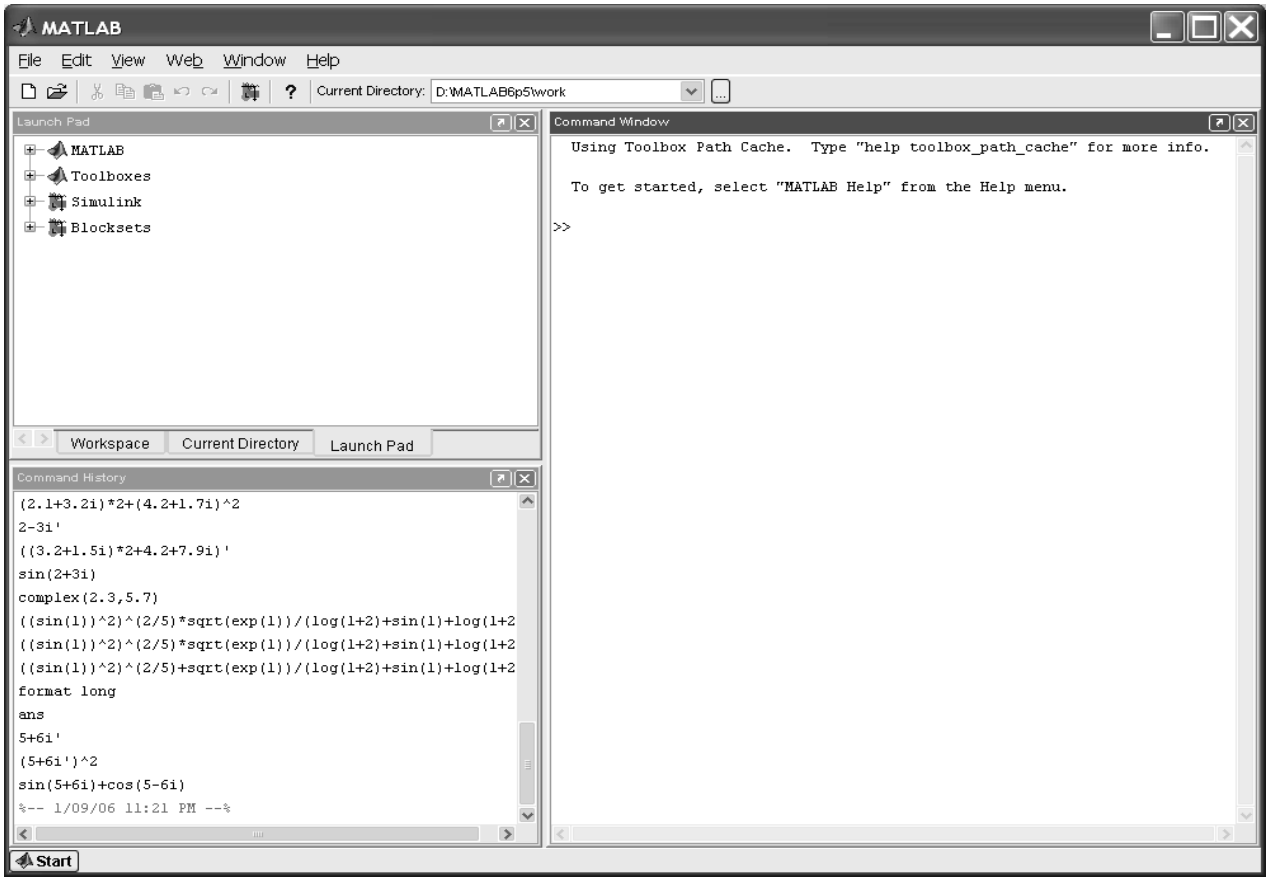


Рис. 1.1. Робоче середовище пакету MATLAB

Усі команди, описані у цій лабораторній роботі, слід набирати у командному рядку. Сам символ >>, який означає запрошення командного рядка, наведений у прикладах, набирати не потрібно. Для перегляду робочої області зручно використовувати скролінг або клавіші <*Home*>, <*End*> для переміщення вліво або вправо і <*PageUp*>, <*PageDown*> для переміщення вгору або вниз. Про використання клавіш <*up*>, <*down*>, <*rigth*>, <*left*> буде сказано додатково. Якщо раптом після переміщення робочої області командного вікна зник командний рядок з миготливим курсором, просто натисніть <*Enter*>.

Важливо запам'ятати, що набір будь-якої команди або виразу повинен закінчуватися натисканням клавіші <*Enter*>, щоб програма MATLAB виконала цю команду або обчислила вираз.

Зауваження 1

Якщо в робочому середовищі MATLAB відсутні деякі описані вікна, то треба в меню *View* вибрати відповідні пункти: *Command Window, Command History, Current Directory, Workspace, Launch Pad*.

1.1.2. Арифметичні обчислення

Вбудовані математичні функції MATLAB дозволяють знаходити значення різних виразів. MATLAB надає можливість керування форматом виведення результату. Команди для обчислення виразів мають вигляд, властивий всім мовам програмування високого рівня.

Наберіть у командному рядку 1+2 та натисніть <*Enter*>. В результаті в командному вікні MATLAB відображається таке:

```
» 1+2
ans =
     3
```

Що зробила програма MATLAB? Спочатку вона обчислила суму 1 + 2, потім записала результат у спеціальну змінну *ans* і вивела її значення, що дорівнює 3, у командне вікно. Нижче відповіді розташований командний рядок з миготливим курсором, що означає, що MATLAB готова до подальших обчислень. Можна набирати в командному рядку нові вирази та знаходити їх значення.

Якщо потрібно продовжити роботу з попереднім виразом, наприклад, обчислити (1 + 2) / 4.5, то найпростіше скористатися наявним результатом, який зберігається в змінній *ans*. Наберіть у командному рядку ans/4.5 (при введенні десяткових дробів використовується крапка) і натисніть <*Enter*>, виходить:

```
» ans/4.5
ans =
    0.6667
```

Зауваження 2

Вигляд, у якому виводиться результат обчислень, залежить від формату виводу, встановленого у MATLAB. Далі пояснюється, як поставити основні формати виведення.

1.1.3. Формати виведення результату обчислень

Необхідний формат виведення результату визначається користувачем у меню MATLAB. Виберіть *Preferences* у меню *File*. На екрані з'явиться діалогове вікно *Preferences*. Для встановлення формату виводу слід переконатися, що у списку лівої панелі вибрано *Command Window*. Завдання формату проводиться з списку *Numeric format* панелі *Text display*, що розкривається.

Розберемо поки тільки формати, що найчастіше використовуються. Виберіть *short* у списку *Numeric format* у MATLAB. Закрийте діалогове вікно, натиснувши кнопку <*OK*>. Наразі встановлено короткий формат з плаваючою точкою *short* для виведення результатів обчислень, при якому на екрані відображаються лише чотири цифри після десяткової точки. Наберіть у командному рядку 100/3 та натисніть <*Enter*>.

Результат виводиться у форматі *short*:

```
» 100/3
ans =
    33.3333
```

Цей формат виводу збережеться для всіх наступних обчислень, якщо не буде встановлено інший формат. У MATLAB можлива ситуація, коли при відображенні занадто великого чи малого числа результат не вкладається у формат *short*. Обчисліть 100000/3, результат виводиться в експоненційній формі:

```
» 100000/3
ans =
    3.3333e+004
```

Те саме відбудеться і при знаходженні 1/3000:

```
» 1/3000
ans =
    3.3333e–004
```

Однак початкове встановлення формату зберігається і при подальших обчисленнях, для невеликих чисел виведення результату знову відбуватиметься у форматі *short*.

У попередньому прикладі пакет MATLAB вивів результат обчислень в експоненційній формі. Запис 3.3333e-004 означає 3.3333*10-4 або 0.00033333. Аналогічно можна набирати числа у виразах. Наприклад, простіше набрати 10e9 або l.0e10, ніж 1 000 000 000, а результат буде той самий. Пробіл між цифрами та символом e при введенні не допускається, т.к. це призведе до повідомлення про помилку:

```
» 10 e9
??? 10 e9
Missing operator, comma, or semi-colon.
```

Якщо потрібно отримати результат обчислень більш точно, то слід вибрати в списку формат *long*, що розкривається. Результат буде відображатися у довгому форматі із чотирнадцятьма цифрами після десяткової точки. Формати *short e* та *long e* призначені для виведення результату в експоненційній формі з чотирма та п'ятнадцятьма цифрами після десяткової точки відповідно. Інформацію про формати можна отримати, набравши в командному рядку команду *help* із аргументом *format*:

```
» help format
```

У командному вікні з'являється опис кожного формату.

Формат виведення можна задавати безпосередньо з командного рядка за допомогою команди *format*. Наприклад, для встановлення довгого з плаваючою точкою формату виведення результатів обчислень слід ввести команду *format long* e у командному рядку:

```
» format long e
» 1.25/3.11
ans =
    4.019292604501608e-001
```

Зверніть увагу, що команда *help format* виводить на екран назву форматів великими літерами. Проте команда, яку треба запровадити, складається із малих літер. До цієї особливості вбудованої довідки *help* треба звикнути. MATLAB розрізняє великі та малі літери. Спроба набору команди великими літерами призведе до помилки:

```
» FORMAT LONG E
??? FORMAT LONG.
Missing operator, comma, or semi-colon.
```

Для зручнішого сприйняття результату MATLAB виводить результат обчислень через рядок після виразу, що обчислюється. Однак іноді буває зручно розмістити більше рядків на екрані, для чого слід вибрати перемикач *compact* (*File, Numeric display*) з списку, що розкривається. Додавання порожніх рядків забезпечується вибором *loose* з списку *Numeric display*, що розкривається.

Зауваження 3

Усі проміжні обчислення MATLAB з подвійною точністю, незалежно від того, який формат виведення встановлений.

1.1.4. Використання елементарних функцій

Припустимо, що потрібно обчислити значення наступного виразу:

$$e^{-2,5} \cdot (\ln 11,3)^{0,3} - \sqrt{(\sin 2,45\pi + \cos 3,78\pi)/(\operatorname{tg} 3,3)}\,.$$

Введіть у командному рядку цей вираз відповідно до правил MATLAB і натисніть <*Enter*>:

```
» exp(–2.5)*log(11.3)^0.3– sqrt((sin(2.45*pi)+cos(3.78*pi))/
tan(3.3))
```

Відповідь виводиться у командне вікно:

```
ans =
    -3.2105
```

При введенні виразу використано вбудовані функції MATLAB для обчислення експонентів, натурального логарифму, квадратного кореня та тригонометричних функцій.

Які вбудовані елементарні функції можна використовувати та як їх викликати? Наберіть у командному рядку команду *help eifun*, при цьому в командне вікно виводиться список усіх вбудованих елементарних функцій з коротким описом. Аргументи функцій розміщують у круглі дужки, імена функцій набираються малими літерами. Для введення числа π достатньо набрати *pi* в командному рядку.

Арифметичні операції в MATLAB виконуються у звичайному порядку, властивому більшості мов програмування:

- зведення у ступінь ^;

- множення та розподіл *, /;

- додавання та віднімання +, -.

Для зміни порядку виконання арифметичних операторів слід використовувати круглі дужки.

Якщо тепер потрібно обчислити значення виразу, схожого на попереднє, наприклад

$$e^{-2,5} \cdot (\ln 11,3)^{0,3} - ((\sin 2,45\pi + \cos 3,78\pi) / (\operatorname{tg} 3,3))^2,$$

то необов'язково його знову набирати у командному рядку. Можна скористатися тим, що MATLAB запам'ятовує всі команди, що вводяться. Для повторного занесення в командний рядок служать клавіші <*up*>, <*down*>. Обчисліть цей вираз, зробивши такі кроки:

1. Натисніть клавішу <↑>, при цьому в командному рядку з'явиться введений вираз.

2. Внесіть до нього необхідні зміни, замінивши знак мінус на плюс і квадратний корінь на зведення в квадрат (для переміщення рядком з виразом служать клавіші <*right*>, <*left*>, <*Home*>, <*End*>).

3. Обчисліть змінений вираз, натиснувши <*Enter*>.

```
» exp(-2.5)*log(11.3)^0.3-
((sin(2.45*pi)+cos(3.78*pi))/tan(3.3))^2
ans =
    121.2446
```

Якщо необхідно отримати більш точний результат, слід виконати команду format long e, потім натискати клавішу <↑> доти, поки в командному рядку не з'явиться необхідний вираз, і обчислити його, натиснувши <*Enter*>

```
» format long e
» exp(-2.5)*log(11.3)^0.3+((sin.(2.45*pi)+cos(3.78*pi))/
tan(3.3))^2
ans =
    1.212446016556763e+002
```

Вивести результат останнього виявленого виразу в іншому форматі можна без повторного обчислення. Слід змінити формат командою *short*, а потім переглянути значення змінної *ans*, набравши її в командному рядку і натиснувши <*Enter*>:

```
» format short
» ans
ans =
    121.2446
```

У робочому середовищі MATLAB для виклику раніше введених команд є зручний засіб – вікно *Command History* з історією команд. Історія команд містить час та дату кожного сеансу роботи з MATLAB. Щоб активувати вікно *Command History*, необхідно вибрати вкладку з однойменною назвою. Поточна команда у вікні зображена на блакитному тлі. Якщо клацнути на будь-якій команді у вікні лівою кнопкою миші, то ця команда стає поточною. Для її виконання в MATLAB треба застосувати подвійне клацання миші або вибрати рядок із командою за допомогою клавіш <*up*>, <*down*> і натиснути клавішу <*Enter*>. Зайву команду можна забрати з вікна. Для цього її потрібно зробити поточною та видалити за допомогою кнопки <*Delete*>. Можна виділити кілька команд, що йдуть поспіль, за допомогою комбінації клавіш <*Shift*>+<*up*>, <*Shift*>+<*down*> і виконати їх за допомогою <*Enter*> або видалити клавішею <*Delete*>. Виділення послідовно команд можна проводити лівою кнопкою миші з одночасним утримуванням клавіші <*Shift*>. Якщо команди не йдуть одна за одною, для їх виділення слід використовувати ліву кнопку миші з утримуванням клавіші <*Ctrl*>.

При натисканні правою кнопкою миші по області вікна *Command History* з'являється спливаюче меню. Вибір пункту *Copy* призводить до копіювання команди в буфер Windows. За допомогою *Evaluate Selection* можна виконати зазначену групу команд. Для видалення поточної команди призначено пункт *Delete Selection*. Для видалення всіх команд до поточної *Delete to Selection*, для видалення всіх команд *Delete Entire History*.

При обчисленнях можливі деякі виняткові ситуації, наприклад, ділення на нуль, які в більшості мов програмування призводять до помилки. При розподілі позитивного числа на нуль у MATLAB виходить *inf* (нескінченність), а при розподілі негативного числа на нуль виходить *-inf* (мінус нескінченність) і видається попередження:

```
» 1/0
Warning: Divide by zero.
ans = Inf
```

При розподілі нуля на нуль виходить *NaN* (не число) і також видається попередження:

```
» 0/0
Warning: Divide by zero.
ans = NaN
```

При обчисленні, наприклад, *sqrt*(-1) жодної помилки чи попередження не виникає. MATLAB автоматично переходить до комплексних чисел:

```
»sqrt(–1.0)
ans = 0 + 1.0000i
```

1.1.5. Робота з комплексними числами

При наборі комплексних чисел у командному рядку MATLAB можна використовувати або *i*, або *j*, а самі числа при множенні, діленні та зведенні в ступінь необхідно укладати у круглі дужки:

```
»(2.1+3.2i)*2+(4.2+1.7i)^2
ans =
    18.9500 + 20.6800i
```

Якщо не використовувати дужки, то множитися або зводитися в ступінь буде тільки уявна частина, і вийде невірний результат:

```
» 2.1+3.2i*2+4.2+1.7i^2
ans =
    3.4100 + 6.4000i
```

Для обчислення комплексно-сполученого числа застосовується апостроф, який слід набирати відразу за числом, без пропуску:

```
» 2–3i'
ans =
    2.0000 + 3.0000i
```

Якщо необхідно знайти комплексно-сполучений вираз, то вихідний вираз має бути поміщений у круглі дужки:

```
»((3.2+1.5i)*2+4.2+7.9i)'
ans =
    10.6000 – 10.9000i
```

MATLAB дозволяє використовувати комплексні числа як аргументи вбудованих елементарних функцій:

```
» sin(2+3i)
ans =
    9.1545 – 4.1689i
```

Конструювання комплексного числа з його дійсної та уявної частини виконується за допомогою функції *complex*:

```
» complex(2.3, 5.8)
ans =
    2.3000 + 5.8000i
```

1.2. Індивідуальні завдання

1. Відповідно до номера *N* за списком у журналі групи, записаному у вигляді *N* = *CM*, де *C* – старша цифра, *M* – молодша цифра, обчисліть вираз, заданий за допомогою табл. 1.1 та табл. 1.2.

2. Замініть у виразах табл. 1.1 та табл. 1.2 знак арифметичної операції множення на знак операції додавання і обчисліть нові вирази без їх повного набору в командному рядку, скориставшись тим, що MATLAB запам'ятовує команди.

Таблиця 1.1 – Індивідуальне завдання за молодшою цифрою M

Молодша цифра *M* номера у журналі групи	0 або 5	1 або 6	2 або 7	3 або 8	4 або 9
Вираз	$\frac{A^{2/5}\cdot\sqrt{C}}{B+D}$	$\frac{\sqrt{A^{-2}}\cdot B^{1,7}}{D+C\cdot A}$	$\frac{A\cdot\sqrt{B+C}}{D^{1/5}/C}$	$\frac{C^{3/7}+\sqrt{D}}{B\cdot\sqrt[3]{A^5}}$	$\frac{A\cdot\sqrt[5]{D}}{C+B^4}$

Таблиця 1.2 – Індивідуальне завдання за старшою цифрою C

Старша цифра *C* номера у журналі групи	*A*	*B*	*C*	*D*
0	$(\sin(N))^2$	$\ln(N+2)$	$\exp(N/N^2)$	$A+B$
1	$(\cos(N))^4$	$(\ln(N^4))^2$	$\exp(-N)+1$	$A+C$
2	$(\mathrm{tg}(N))^2$	$(\log(N))^4$	$1/\exp(N)$	B/C
3	$(\mathrm{ctg}(N))^4$	$\log(N+N^3+9)$	$N/\exp(-N)$	$C/(A+B)$

3. Отримайте результати обчислень пункту 2 індивідуального завдання у форматах *short* та *long*.

4. Отримайте комплексне число (*a* + *bi*), де *a*, *b* – відповідно число літер у вашому імені та прізвищі. При цьому визначте також:

- комплексно-сполучене число (*a* + *bi*);

- обчисліть квадрат комплексно-сполученого числа;
- обчисліть добуток вихідного комплексного числа та комплексно-сполученого числа;
- обчисліть вираз $sin(a + bi) + cos(a – bi)$.

5. Оформіть звіт з лабораторної роботи.

Лабораторна робота 2

ПРОСТІ ОБЧИСЛЕННЯ З ВИКОРИСТАННЯМ ЗМІННИХ І ВЕКТОРІВ В ПАКЕТІ MATLAB

Мета лабораторної роботи: отримання і закріплення знань, формування практичних навичок роботи з пакетом MATLAB при простих обчисленнях з використанням змінних і векторів.

2.1. Короткі відомості з теорії

2.1.1. Використання змінних в пакеті MATLAB

Як і в усіх мовах програмування, в MATLAB передбачена можливість роботи зі змінними. Причому користувач не повинен піклуватися про те, яких значень набуватиме змінна (комплексні, речові або тільки цілі). Для того, щоб присвоїти, наприклад, змінній z значення 1.45, досить написати в командному рядку *z* = 1,45, при цьому MATLAB відразу ж виведе значення *z* :

```
» z = 1.45
z =
    1.4500
```

Тут знак рівності використовується як оператор привласнення. Часто не дуже зручно після кожного привласнення отримувати ще і результат. Тому в MATLAB передбачена можливість завершувати оператор привласнення крапкою з комою для стримання (пригнічення) виведення результату в командне вікно. Ім'ям змінної може бути будь-яка послідовність букв і цифр без пропуску, що розпочинається з букви. Рядкові і прописні букви розрізняються, наприклад, MZ і mz є двома різними змінними. Кількість сприйманих MATLAB символів в імені змінної складає 31.

В якості вправи на використання змінних знайдіть значення наступного вираження:

$$\left(\sin 1{,}3\pi/\ln 3{,}4 + \sqrt{\text{tg } 2{,}75/\text{th } 2{,}75}\right)/\left(\sin 1{,}3\pi/\ln 3{,}4 - \sqrt{\text{tg } 2{,}75/\text{th } 2{,}75}\right).$$

Наберіть послідовність команд, приведену нижче (зверніть увагу на крапку з комою в перших двох операторах привласнення для пригнічення виведення проміжних значень на екран):

```
» x = sin(1.3*pi)/log(3.4);
» y = sqrt(tan(2.75)/tanh(2.75));
» z = (x+y)/(x-y)
Z =
    0.0243 - 0.9997i
```

Останній оператор привласнення не завершується крапкою з комою для того, щоб відразу отримати значення початкового вираження. Звичайно, можна було б ввести відразу усю формулу і отримати той же результат:

```
»(sin(1.3*pi)/log(3.4)+sqrt(tan(2.75)/tanh(2.75)))/…
(sin(1.3*pi)/log(3.4)-sqrt(tan(2.75)/tanh(2.75)))
ans =
    0.0243 - 0.9997i
```

Зверніть увагу, наскільки перший запис компактніший і ясніший за другий! У другому варіанті формула не поміщалася в командному вікні на одному рядку і довелося записати її в два рядки, для чого, у кінці першого рядка, поставлені три точки.

Зауваження 1

Для введення довгих формул або команд в командний рядок слід поставити три точки (підряд, без пропусків), натиснути клавішу <*Enter*> і продовжити набір формули на наступному рядку. Так можна розмістити вираження на декількох

рядках. MATLAB вичислить увесь вираз або виконає команду після натиснення на <*Enter*> у останньому рядку (у якому немає трьох підряд точок).

MATLAB запам'ятовує значення усіх змінних, визначених під час сеансу роботи. Якщо після введення прикладу, приведеного вище, були виконані ще якісь обчислення і виникла необхідність вивести значення *x*, то слід просто набрати *x* в командному рядку і натиснути <*Enter*>:

```
» x
     -0.6611
```

Змінні, визначені вище, можна використати і в інших формулах. Наприклад, якщо тепер необхідно вичислити вираження

$$\left(\sin 1{,}3\pi / \ln 3{,}4 + \sqrt{\operatorname{tg} 2{,}75 / \operatorname{th} 2{,}75}\right)^{3/2},$$

то достатньо ввести наступну команду:

```
» (x+y)^(3/2)
ans =
     -0.8139 + 0.3547i
```

Виклик функцій в MATLAB має достатню гнучкість. Наприклад, вичислити $e^{3{,}5}$ можна, викликавши функцію *exp* з командного рядка:

```
» exp(3.5)
ans =
     33.1155
```

Інший спосіб полягає у використанні оператора привласнення:

```
» t = exp(3.5)
t =
     33.1155
```

Припустимо, що частина обчислень зі змінними виконана, а інші доведеться доробити під час наступного сеансу роботи з MATLAB. В цьому випадку знадобиться зберегти змінні, визначені в робочому середовищі.

2.1.2. Збереження робочого середовища

Найпростіший спосіб зберегти значення усіх змінних – використати в меню *File* пункт *Save Workspace As*. При цьому з'являється діалогове вікно *Save*, у якому слід вказати каталог і ім'я файлу. За замовчанням пропонується зберегти файл в підкаталозі work основного каталогу MATLAB. Залиште доки цей каталог. Надалі буде пояснено, як встановлювати шляхи до каталогів в MATLAB для пошуку файлів. Зручно давати файлам імена, що містять дату роботи, наприклад *work20-06-22*. MATLAB збереже результати роботи у файлі *work20-06-22.mat*.

Тепер можна закрити MATLAB одним з наступних способів :

- вибрати в меню *File* пункт *Exit* MATLAB;
- натиснути клавіші <*Ctrl*>+<*Q*>;
- набрати команду *Exit* в командному рядку і натиснути <*Enter*>;
- натиснути на кнопку з хрестиком в правому верхньому кутку вікна програми MATLAB.

У наступному сеансі роботи для відновлення значень змінних слід відкрити файл *work20-06-22.mat* за допомогою пункту *Open* меню *File*. Тепер усі змінні, визначені в минулому сеансі, стали доступними. Їх можна використати в командах, що знову вводяться.

Збереження і відновлення змінних робочого середовища можна виконати і з командного рядка. Для цього служать команди *save* і *load*. У кінці сеансу роботи з MATLAB потрібно виконати команду

```
» save work20-06-22
```

Детальну інформацію про команди *save* і *load* можна отримати, набравши в командному рядку *help save* або *help load*.

```
» load work20-06-22
```

Зауваження 2

Змінні у файлах з розширенням *mat* зберігаються в двійковому виді. Перегляд цих файлів у будь-якому текстовому редакторові не дасть ніякої інформації про змінні і їх значення.

У MATLAB є можливість записувати виконувані команди і результати в текстовий файл (вести журнал роботи), який потім можна легко прочитати або роздрукувати з текстового редактора. Для початку ведення журналу служить команда *diary*. В якості аргументу команди *diary* слід задати ім'я файлу, в якому зберігатиметься журнал роботи. Команди, що набирають далі, і результати їх виконання записуватимуться в цей файл, наприклад, послідовність команд

```
» diary d20-06-12.txt
» al = 3;
» a2 = 2.5;
» a3 = al + a2
» a3 =
    5.5000
» save work20-06-12
» quit
```

виконує наступні дії:

- Відкриває файл *d20-06-22.txt*.

- Робить обчислення.

- Зберігає змінні в двійковому файлі *work20-06-22.mat*.

- Зберігає на диску у подкаталозі *work* кореневого каталогу MATLAB журнал роботи у файлі *d20-06-22.txt* і закриває MATLAB.

Подивіться вміст файлу *d20-06-22.txt* в якому-небудь текстовому редакторові, наприклад, в стандартній програмі *Windows* Блокнот (*NotePad*). У файлі опиниться наступний текст:

```
a1 = 3;
a2 = 2.5;
a3 = a1+a2
a3 =
    5.5000
save work20-02-22
quit
```

Запустіть знову MATLAB і введіть команду *load work20 -06-22* або відкрийте файл *work20-06-22.mat* за допомогою меню, як описано вище.

2.1.3. Перегляд змінних

При роботі з досить великою кількістю змінних необхідно знати, які змінні вже використані, а яких немає. Для цієї мети служить команда *who*, що виводить в командне вікно MATLAB список використаних змінних:

```
» who
Your variables are:
a1   a2   a3
```

Команда *whos* дозволяє отримати більш детальну інформацію про змінні у вигляді таблиці:

```
» whos
Name   Size   Bytes  Class
a1     1x1    8      double
a2     1x1    8      double
a3     1x1    8      double
Grand total is 3 elements using 24 bytes
```

Перший стовпчик *Name* складається з імен використаних змінних. Те, що міститься в стовпчику *Size*, по суті, визначається основним принципом роботи MATLAB. Програма MATLAB усі дані представляє у вигляді масивів. Змінні *a*l, *a*2 і *a*3 є двомірними масивами розміру один на один. Кожна зі змінних займає по вісім байтів, як вказано в стовпчику *Bytes*. Нарешті, в останньому стовпчику *Class* вказаний тип змінних - *double array*, тобто масив складається з чисел подвійної

точності. У рядку під таблицею написано, що у результаті три елементи, тобто змінні, займають двадцять чотири байти. Виявляється, що представлення усіх даних в MATLAB у вигляді масивів дає певні переваги.

Для звільнення з пам'яті усіх змінних використовується команда *clear*. Якщо в аргументах вказати список змінних (через пропуск), то тільки вони будуть звільнені з пам'яті, наприклад:

```
» clear a1 a3
» who
Your variables are:
a2
```

2.1.4. Робота з масивами

Дуже важливо правильно зрозуміти, як використовувати масиви. Без цього неможлива ефективна робота в MATLAB, зокрема побудова графіків, рішення завдань лінійної алгебри, обробки даних, статистики і багатьох інших. У цьому підрозділі описані обчислення з векторами.

Масив – впорядкована, пронумерована сукупність однорідних даних. У масиву має бути ім'я. Масиви розрізняються по числу розмірності або вимірів : одновимірні, двомірні, багатовимірні. Доступ до елементів здійснюється за допомогою індексу. У MATLAB нумерація елементів масивів розпочинається з одиниці. Це означає, що індекси мають бути більші або дорівнюють одиниці.

Важливо зрозуміти, що вектор, вектор-рядок або матриця є математичними об'єктами, а одновимірні, двомірні або багатовимірні масиви - способи зберігання цих об'єктів в комп'ютері. Далі використовуватимуться слова вектор і матриця, якщо більший інтерес представляє сам об'єкт, чим спосіб його зберігання. Вектор може бути записаний в стовпчик (вектор-стовпець) і в рядок (вектор-рядок). Вектор-стовпці і вектор-рядки часто називатимуться просто векторами, відмінність буде зроблена в тих випадках, якщо важливий спосіб зберігання вектору в MATLAB.

Введення, складання і віднімання векторів.

Роботу з масивами розпочнемо з простого прикладу - обчислення суми векторів:

$$a = \begin{pmatrix} 1{,}3 \\ 5{,}4 \\ 6{,}9 \end{pmatrix},\ b = \begin{pmatrix} 7{,}1 \\ 3{,}5 \\ 8{,}2 \end{pmatrix}.$$

Для зберігання векторів використайте масиви *a* і *b*. Введіть масив *a* в командному рядку, використовуючи квадратні дужки і розділяючи елементи вектору крапкою з комою:

```
» a = [1.3; 5.4; 6.9]
a = 1.3000
    5.4000
    6.9000
```

Оскільки введене вираження не завершене крапкою з комою, то пакет MATLAB автоматично вивів значення змінної *a*.

Введіть тепер другий вектор

```
» b = [7.1; 3.5; 8.2];
```

Для знаходження суми векторів використовується знак +. Обчисліть суму, запишіть результат в масив *c* і виведіть його елементи в командне вікно:

```
» c = a + b
c =
    8.4000
    8.9000
    15.1000
```

Дізнайтесь розмірність і розмір масиву *a* за допомогою вбудованих функцій *ndims* і *size* :

```
» ndims(a)
ans =
     2
» size(a)
ans = 3  1
```

Отже, вектор *a* зберігається в двомірному масиві *a* з розмірністю три на один (вектор-стовпець з трьох рядків і одного стовпця). Аналогічні операції можна виконати і для масивів *b* і *c*. Оскільки числа в пакеті MATLAB представляються у вигляді двомірного масиву один на один, то при складанні векторів використовується той же знак плюс, що і для складання чисел.

Введення вектор-рядка здійснюється в квадратних дужках, проте елементи слід розділяти пропусками або комами. Операції складання, віднімання і обчислення елементарних функцій від вектор-рядків робляться так само, як і з вектор-стовпцями, в результаті виходять вектор-рядки того ж розміру, що і початкові. Наприклад:

```
» s1 = [3 4 9 2]
s1 = 3 4 9 2
» s2 = [5 3 3 2]
s2 = 5 3 3 2
» s3 = s1 + s2
s3 = 8 7 12 4
```

Зауваження 3

Якщо розміри векторів, до яких застосовується складання або віднімання, не співпадають, то видається повідомлення про помилку.

Для знаходження різниці векторів слід застосовувати знак мінус, з множенням – складніше. Введіть два вектор-рядки:

```
» v1 = [2 -3 4 1];
» v2 = [7 5 -6 9];
```

Операція **.*** (не вставляйте пропуск між точкою і зірочкою!) призводить до поелементного множення векторів однакової довжини. В результаті виходить вектор з елементами, рівними добутку відповідних елементів початкових векторів :

```
» u = v1.*v2
u = 14  -15  -24   9
```

За допомогою .^ виконується поелементне зведення в ступінь:

```
» p = v1.^2
p =
    4   9   16   1
```

Показником ступеня може бути вектор тієї ж довжини, що і зводиться в ступінь. При цьому кожен елемент першого вектору зводиться в ступінь, рівного відповідному елементу другого вектору:

```
» p = vl.^v2
p =
    128.0000 -243.0000   0.0002   1.0000
```

Ділення відповідних елементів векторів однакової довжини виконується з використанням операції ./

```
» d = v1./v2
d =
    0.2857 -0.6000 -0.6667   0.1111
```

Зворотне поелементне ділення (ділення елементів другого вектору на відповідні елементи першого) здійснюється за допомогою операції .\

```
» dinv = vl.\v2
dinv =
    3.5000 -1.6667 -1.5000   9.0000
```

Отже, точка в MATLAB використовується не лише для введення десяткових дробів, але і для вказування того, що ділення або множення масивів однакового розміру має бути виконане поелементно.

До поелементних відносяться і операції з вектором і числом. Складання вектору і числа не призводить до повідомлення про помилку. MATLAB додає число до кожного елементу вектору. Те саме справедливо й для віднімання:

```
» v = [4 6 8 10];
» s = v + 1.2
s =
    5.2000 6.2000 9.2000 11.2000
» r = 1.2 - v
r =
    -2.8000 -4.8000 -6.8000 -8.8000
» r1 = v - 1.2
r1 =
    2.8000 4.8000 6.8000 8.8000
```

Множити вектор на число можна як справа, так і зліва:

```
» v = [4 6 8 10];
» p = v*2
p = 8  12  16  20
» pi = 2*v
pi = 8  12  16  20
```

Ділити за допомогою знаку / можна вектор на число:

```
» p = v/2
p =
    2  3  4  5
```

Спроба ділення числа на вектор призводить до повідомлення про помилку:

```
» p = 2/v
??? Error using ==> /
Matrix dimensions must agree.
```

Якщо вимагається розділити число на кожен елемент вектору і записати результат в новий вектор, то слід використати операцію ./

```
» w = [4 2 6];
» d = 12./w
d = 3 6 2
```

Усі вищеописані операції застосовуються як до вектор-рядків, так і до вектор-стовпців.

Особливість MATLAB представляти усі дані у вигляді масивів є дуже зручним. Нехай, наприклад, вимагається вичислити значення функції *sin* відразу для усіх елементів вектору *c* (який зберігається в масиві *c*) і записати результат у вектор *d*. Для отримання вектору *d* досить використати один оператор привласнення:

```
» d = sin(c)
d =
    0.8546
    0.5010
    0.5712
```

Отже, вбудовані в MATLAB елементарні функції пристосовуються до виду аргументів; якщо аргумент є масивом, то результат функції буде масивом того ж розміру, але з елементами, рівними значенню функції від відповідних елементів початкового масиву. Переконайтеся в цьому ще на одному прикладі. Якщо необхідно знайти квадратний корінь з елементів вектору *d* зі знаком мінус, то досить записати:

```
» sqrt(-d)
ans =
    0 + 0.9244i
    0 + 0.7078i
    0 + 0.7558i
```

Оператор привласнення не використовувався, тому пакет MATLAB записав відповідь в стандартну змінну *ans*.

Для визначення довжини вектор-стовпців або вектор-рядків служить вбудована функція *length* :

```
» length(s1)
ans =
     4
```

З декількох вектор-стовпців можна скласти один, використовуючи квадратні дужки і розділяючи елементи крапкою з комою:

```
» v1 = [1; 2];
» v2 = [3; 4; 5];
» v = [v1; v2]
v =
     1
     2
     3
     4
     5
```

Для об'єднання вектор-рядків також застосовуються квадратні дужки, але самі вектор-рядки відділяються пропусками або комами:

```
» v1 = [1 2];
» v2 = [3 4 5];
» v = [v1 v2]
v =
     1 2 3 4 5
```

Робота з елементами векторів

Доступ до елементів вектор-стовпця або вектор-рядка здійснюється за допомогою індексу, що береться в круглі дужки після імені масиву, в якому зберігається вектор. Якщо серед змінних робочого середовища є масив *v*, визначений вектор-рядком

```
» v = [1.3 3.6 7.4 8.2 0.9];
```

то для виведення, наприклад його четвертого елементу, використовується індексація:

```
» v(4)
ans =
    8.2000
```

Поява елементу масиву в лівій частині оператора привласнення призводить до зміни в масиві

```
» v(2) = 555
v =
    1.3000 555.0000 7.4000 8.2000 0.9000
```

З елементів масиву можна формувати нові масиви, наприклад

```
» u = [v(3); v(2); v(1)]
u =
    7.4000
    555.0000
    1.3000
```

Щоб помістити певні елементи вектору в інший вектор в заданому порядку треба скористатись індексацією за допомогою вектору. Запис в масив *w* четвертого, другого і п'ятого елементів масиву *v* проводиться таким чином:

```
» ind = [4 2 5];
» w = v(ind)
w =
    8.2000 555.0000 0.9000
```

MATLAB надає зручний спосіб звернення до блоків послідовно розташованих елементів вектор-стовпця або вектор-рядка. Для цього служить індексація за допомогою знаку двокрапки. Припустимо, що в масиві *w*, що

відповідає вектор-рядку з семи елементів, вимагається замінити нулями елементи з другого по шостий. Індексація за допомогою двокрапки дозволяє просто і наочно вирішити поставлене завдання:

```
» w = [0.1 2.9 3.3 5.1 2.6 7.1 9.8];
» w(2:6) = 0;
» w
w =
     0.1000 0 0 0 0 0 9.8000
```

Присвоювання *w* (2: 6) = 0 еквівалентно послідовності команд

```
w (2) = 0; w (3) = 0; w (4) = 0; w (5) = 0; w (6) = 0.
```

Індексація за допомогою двокрапки виявляється зручною при виділенні частини з великого об'єму даних в новий масив:

```
» w = [0.1 2.9 3.3 5.1 2.6 7.1 9.8];
» wl = w(3:5)
wl =
     3.3000  5.1000  2.6000
```

Складіть масив *w*2, що містить елементи *w*, окрім четвертого. В цьому випадку зручно використати двокрапку і зчеплення рядків :

```
» w2 = [w(l:3) w(5:7)]
w2 =
     0.1000 2.9000 3.3000 2.6000 7.1000 9.8000
```

Елементи масиву можуть входити у вирази. Знаходження, наприклад, середнього геометричного з елементів масиву *u* можна виконати таким чином:

```
» gm = (u(l)*u(2)*u(3))^(l/3)
gm =
     17.4779
```

Звичайно, цей спосіб не дуже зручний для довгих масивів. Для того, щоб знайти середнє геометричне, необхідно набрати у формулі усі елементи масиву. У MATLAB існує досить багато спеціальних функцій, що полегшують подібні обчислення.

Застосування функцій обробки даних до векторів

Перемножування елементів вектор-стовпця або вектор-рядка здійснюється за допомогою функції *prod*:

```
» z = [3; 2; 1; 4; 6; 5];
» p = prod(z)
p =
     720
```

Функція *sum* призначена для підсумовування елементів вектору. З її допомогою неважко вичислити середнє арифметичне елементів вектору *z* :

```
» sum(z)/length(z)
ans =
     3.5000
```

У MATLAB є і спеціальна функція *mean* для обчислення середнього арифметичного :

```
» mean(z)
ans =
     3.5000
```

Для визначення мінімального і максимального з елементів вектору служать вбудовані функції *min* і *max* :

```
» m1 = max(z)
m1 = 6
» m2 = min(z)
m2 = 1
```

Часто необхідно знати не лише значення мінімального або максимального елементу в масиві, але і його індекс (порядковий номер). В цьому випадку вбудовані функції *min* і *max* необхідно використати з двома вихідними аргументами, наприклад:

```
» [m, k] = min(z)
m =
     1
k =
     3
```

В результаті змінної *m* буде присвоєне значення мінімального елементу масиву *z*, а номер мінімального елементу занесений в змінну *k*.

Для отримання інформації про різні способи використання функцій слід набрати в командному рядку *help* і ім'я функції. MATLAB виведе в командне вікно всілякі способи звернення до функції з додатковими поясненнями.

У число основних функцій для роботи з векторами входить функція впорядкування вектору за збільшенням його елементів *sort*

```
» r = [9.4 -2.3 -5.2 7.1 0.8 1.3];
» R = sort(r)
R =
    -5.2000 -2.3000 0.8000 1.3000 7.1000 9.4000
```

Можна упорядкувати вектор по зменшенню, використовуючи цю ж функцію *sort* :

```
» R1 = -sort(r)
R1 =
     9.4000 7.1000 1.3000 0.8000 -2.3000 -5.2000
```

Впорядкування елементів в порядку зростання їх модулів здійснюється із залученням функції *abs*:

```
» R2 = sort(abs(r))
R2 = 0.8000 1.3000 2.3000 5.2000 7.1000 9.4000
```

Виклик *sort* з двома вихідними аргументами призводить до утворення масиву індексів відповідності елементів впорядкованого і початкового масивів:

```
» [rs, ind] = sort(r)
rs =
    -5.2000 -2.3000 0.8000 1.3000 7.1000 9.4000
ind =
    3   2   5   6   4   1
```

2.2. Індивідуальні завдання

1. Створіть журнал виконання лабораторної роботи.

2. Обчисліть значення функції

$$y(x) = \frac{\sin^2 x}{1+\cos x} + e^{-x}\ln x$$

в точках 0.2, 0.3, 0.5, 0.8, 1.3, 1.7, 2.5, *N*, *k*, де *N* – ваш номер за списком у журналы групи; *k* – чисельне значення вираження, заданого у таблиці 1.1 і таблиці 1.2 (з лабораторної роботи 1) відповідно до номера *N* за списком в журналі групи, записаному у вигляді *N* = *CM*, де *C* – старша цифра, *M* – молодша цифра.

3. Сформуйте вектор-рядок *v*, що містить усі значення аргументу *x* і останні п'ять значень функції *y*(*x*).

4. Отримайте вектор-рядок *v*1, додавши до кожного елементу вектор-рядка *v* число 2.1.

5. Обчисліть: *w* = *v* + *v*1; *w*1 = *v* – *v*1; *w*2 = *v*1 – *v*; *w*3 = *v* ./ *v*1; *w*4 = *v*1.**v*; *w*5 = *v*1.^*v*.

6. Упорядкуйте результати складання векторів *v* + *v*1 в порядку зростання модулів елементів вектору суми, зростання елементів вектору суми, убування елементів вектору суми.

7. Сформуйте з третіх і п'ятих елементів вектор-рядків *w*, *w*1, *w*2, *w*3, *w*4, *w*5 вектор-стовпець *ww*.

8. Визначте у векторі *ww* мінімальний і максимальний елемент, суму і добуток елементів вектору.

9. Присвойте елементам масиву *ww* з третього по шостій значення, рівні одиниці.

10. Використайте команди *who* і *whos* для отримання інформації про усі використані в лабораторній роботі змінні.

11. Наведіть з журналу виконання лабораторної роботи декілька перших і останніх рядків.

12. Оформіть звіт по лабораторній роботі.

Лабораторна робота 3

НАЙПРОСТІШІ ОБЧИСЛЕННЯ З ВИКОРИСТАННЯМ МАТРИЦЬ У ПАКЕТІ MATLAB

Мета лабораторної роботи: здобуття та закріплення знань, формування практичних навичок роботи з пакетом MATLAB при найпростіших обчисленнях з використанням змінних, векторів та матриць.

3.1. Короткі відомості з теорії

3.1.1. Різні способи введення матриць у пакеті MATLAB

Вводити невеликі за розміром матриці зручно прямо з командного рядка. Введіть матрицю розмірністю два на три

$$A=\begin{pmatrix} 3 & 1 & -1 \\ 2 & 4 & 3 \end{pmatrix}.$$

Для зберігання матриці використовуйте двомірний масив з ім'ям A. При введенні врахуйте, що матрицю A можна розглядати як вектор-стовпець з двох елементів, кожен з яких є вектор-рядком довжиною три, отже, рядки при наборі відокремлюються крапкою з комою:

```
» A =[3 1 -1; 2 4 3]
A =
     3   1   -1
     2   4   3
```

Для вивчення найпростіших операцій над матрицями наведемо ще кілька прикладів. Розглянемо інші методи введення. Введіть квадратну матрицю розміру три так, як описано нижче:

$$B = \begin{pmatrix} 4 & 3 & -1 \\ 2 & 7 & 0 \\ -5 & 1 & 2 \end{pmatrix}.$$

Почніть набирати в командному рядку

```
» B = [4 3 -1
```

Натисніть <*Enter*>. Зверніть увагу, що пакет нічого не обчислив. Курсор блимає на наступному рядку без символу >>. Продовжуйте введення матриці по рядкам, натискаючи наприкінці кожного рядка <*Enter*>. Останній рядок завершіть квадратною дужкою, що закривається, при цьому отримаємо:

```
2 7 0
-5 1 2]
B = 4 3 -1
     2 7 0
    -5 1 2
```

Ще один спосіб введення матриць полягає в тому, що матрицю можна трактувати як вектор-рядок, кожен елемент якої є вектор-стовпцем. Наприклад, матрицю два на три

$$C = \begin{pmatrix} 3 & -1 & 7 \\ 4 & 2 & 0 \end{pmatrix}$$

можна ввести за допомогою команди:

```
» C = [[3; 4] [-1; 2] [7; 0]]
C = 3 -1  7
    4  2  0
```

Подивіться змінні робочого середовища, набравши у командному рядку *whos*:

```
A     2x3          48 double array
B     3x3          72 double array
C     2x3          48 double array
```

Отже, у робочому середовищі міститься три матриці, дві прямокутні та одна квадратна.

3.1.2. Звертання до елементів матриць у пакеті MATLAB

Доступ до елементів матриць здійснюється за допомогою двох індексів - номерів рядка та стовпця, укладених у круглі дужки, наприклад

```
» C(2, 3)
ans =
    0
```

Елементи матриць можуть входити до складу виразів:

```
» C(1, 1) + C(2, 2) + C(2, 3)
ans = 5
```

Розташування елементів матриці у пам'яті комп'ютера визначає ще один спосіб звернення до них. Матриця *A* розміру *m* на *n* зберігається у вигляді вектору довжини *mn*, в якому елементи матриці розташовані один за одним по стовпцях

```
[A(1,1) A(2,1)...A(m,1)...A(1,n) A(2,n)...A(m,n)].
```

Для доступу до елементів матриці можна використовувати один індекс, який визначає порядковий номер елемента матриці у векторі.

Матриця *C*, визначена у попередньому підрозділі, міститься у векторі

```
[C(1,1) C(2,1) C(1,2) C(2,2) C(1,3) C(2,3)],
```

який має шість компонентів. Доступ до елементів матриці здійснюється так:

```
» C(1)
ans =
     3
» C(5)
ans =
     7
```

3.1.3. Операції над матрицями в пакеті MATLAB: додавання, віднімання, множення, транспонування та зведення в ступінь

При використанні матричних операцій слід пам'ятати, що для додавання або віднімання матриці повинні бути одного розміру, а при перемноженні число стовпців першої матриці має дорівнювати числу рядків другої матриці. Складання і віднімання матриць, як і чисел і векторів, здійснюється за допомогою знаків плюс і мінус. Знайдіть суму та різницю матриць *C* і *A*, визначених вище:

```
» S = A+C
S =
     6   0   6
     6   6   3
» R = C-A
R =
     0   -2   8
     2   -2   -3
```

Слідкуйте за збігом розмірності, інакше отримайте повідомлення про помилку:

```
» S = A+B
??? Error using ==> ±
Matrix dimensions must agree.
```

Для множення матриць призначена зірочка:

```
» P = C*B
```

```
P =
    -25   9   11
    20   26   -4
```

Множення матриці на число теж здійснюється за допомогою зірочки, причому множити на число можна записуючі його як праворуч, так і ліворуч:

```
» P = A*3
P =
    9   3 -3
    6  12   9
» P = 3*A
P =
    9   3 -3
    6  12   9
```

Транспонування матриці, як і векторів, проводиться з допомогою .' (крапка та апостроф), а символ ' означає комплексне сполучення. Для дійсних матриць ці операції призводять до однакових результатів:

```
» B'
ans =
     4  2 -5
     3  7  1
    -1  0  2
» B.'
ans =
     4  2 -5
     3  7  1
    -1  0  2
```

Зауваження 1

Якщо матриця $A \equiv (a_{ik}),\ i = \overline{1,n},\ k = \overline{1,m}$, є довільною матрицею розміру $n \times m$, то матриця, транспонована відносно A, є матриця розміру $m \times n$: $A' \equiv (a_{ki}),\ k = \overline{1,m},\ i = \overline{1,n}$. Таким чином, рядки матриці A стають стовпцями матриці A', а стовпці матриці A стають рядками матриці A'.

Комплексно-сполучену матрицю отримаємо з вхідної у два етапи: виконується транспонування вихідної матриці, а потім усі комплексні числа замінюються на комплексно-сполучені.

Поєднання та транспонування матриць, що містять комплексні числа, приведуть до створення різних матриць:

```
» K = [1-i, 2+3i; 3-5i, 1-9i]
K = 1.0000 - 1.0000i   2.0000 + 3.0000i
    3.0000 - 5.0000i   1.0000 - 9.0000i
» K'
ans =
    1.0000 + 1.0000i   3.0000 + 5.0000i
    2.0000 - 3.0000i   1.0000 + 9.0000i
» K.'
ans =
    1.0000 - 1.0000i   3.0000 - 5.0000i
    2.0000 + 3.0000i   1.0000 - 9.0000i
```

Зауваження 2

При введенні вектор-рядків їх елементи можна розділяти або пробілами, або комами. При введенні матриці K застосовані коми для наочного поділу комплексних чисел у рядку.

Зведення квадратної матриці в цілий ступінь здійснюється за допомогою оператора ^ (вимовляється як «карет»):

```
» B2 = B^2
B2 =
     27   32   -6
     22   55   -2
    -28   -6    9
```

Перевірте отриманий результат, помноживши матрицю саму себе.

Переконайтеся, що ви засвоїли найпростіші операції з матрицями у MATLAB. Знайдіть значення наступного виразу

$$(A + C)\cdot B^3\cdot(A - C)^{\mathrm{T}}.$$

Врахуйте пріоритет операцій: спочатку виконується транспонування, потім зведення в ступінь, потім множення, а додавання та віднімання проводяться в останню чергу

```
» (A+C)*B^3*(A-C)'
ans =
    1848   1914
    10290  3612
```

3.1.4. Множення матриць та векторів

Вектор-стовпець або вектор-рядок у MATLAB є матрицями, у яких один із розмірів дорівнює одиниці, тому всі вищеописані операції можна застосувати і для множення матриці на вектор-стовпець або вектор-рядки на матрицю. Наприклад, обчислення виразу

$$[1 \quad 3 \quad -2]\begin{pmatrix} 2 & 0 & 1 \\ -4 & 8 & -1 \\ 0 & 9 & 2 \end{pmatrix}\begin{bmatrix} -8 \\ 3 \\ 4 \end{bmatrix}$$

можна здійснити наступним чином:

```
» a = [1 3 -2];
» B = [2 0 1; -4 8 -1; 0 9 2];
» c = [-8; 3; 4];
» a*B*c
ans = 74
```

3.1.5. Розв'язання систем лінійних рівнянь

У математиці нічого не йдеться про поділ матриць та векторів, проте в MATLAB символ \ використовується для розв'язання систем лінійних рівнянь. Вирішимо систему з трьох рівнянь із трьома невідомими:

$$\begin{cases} 1{,}2x_1 + 0{,}3x_2 + 0{,}2x_3 = 1{,}3; \\ 0{,}5x_1 + 2{,}1x_2 + 1{,}3x_3 = 3{,}9; \\ -0{,}9x_1 + 0{,}7x_2 + 5{,}6x_3 = 5{,}4. \end{cases}$$

Введемо матрицю коефіцієнтів системи масив *A*, а вектор правої частини системи масив *b*. Вирішимо систему за допомогою символу \:

```
» x = A\b
x =  0.6465
     1.1293
     0.9270
```

Перевірте вірність відповіді, помножив матрицю коефіцієнтів системи *A* на вектор-стовпець *x*.

3.1.6. Блокові матриці

Найчастіше у додатках виникають так звані блокові матриці, тобто матриці, складені з підматриць, що не перетинаються (блоків). Розглянемо спочатку конструювання блокових матриць. Введіть матриці

$$A=\begin{pmatrix}-1 & 4\\ -1 & 4\end{pmatrix},\ B=\begin{pmatrix}2 & 0\\ 0 & 5\end{pmatrix},\ C=\begin{pmatrix}3 & -3\\ -3 & 3\end{pmatrix},\ D=\begin{pmatrix}8 & 9\\ 1 & 10\end{pmatrix}$$

та створіть з них блочну матрицю $K=\begin{pmatrix}A & B\\ C & D\end{pmatrix}$.

Враховуючи, що матриця *K* складається з двох рядків, у першому рядку матриці *A* та *B*, а у другому - *C* та *D*, блочну матрицю можна сформувати наступним чином:

```
» K = [A B; C D]
K =  -1   4   2   0
     -1   4   0   5
      3  -3   8   9
     -3   3   1  10
```

Блокову матрицю можна отримати й іншим способом, якщо вважати, що матриця *K* складається з двох стовпців, у першому - матриці *A* і *C*, а в другому – *B* і *D*:

```
» K = [[A; C] [B; D]]
```

Зворотним до конструювання блокових матриць є завдання вилучання блоків. Вилучання блоків матриць здійснюється індексацією за допомогою двокрапки. Введіть матрицю

$$P = \begin{pmatrix} 1 & 2 & 0 & 2 \\ 4 & 10 & 12 & 5 \\ 0 & 11 & 10 & 5 \\ 9 & 2 & 3 & 5 \end{pmatrix}$$

і потім вилучіть підматрицю з елементами a_{22}, a_{23}, a_{32}, a_{33}, задав номера рядків та стовбців за допомогою двокрапки:

```
»P1 = P(2:3,2:3)
P1 =
     10  12
     11  10
```

Для вилучання з матриці стовпця або рядка (тобто масиву, у якого один з розмірів дорівнює одиниці) слід у якості одного з індексів використовувати номер стовпця або рядка матриці, а інший індекс замінити двокрапкою без вказівки меж. Наприклад, запишіть другий рядок матриці *P* у вектор *p*

```
»p = P(2, :)
p =
    4 10 12 5
```

При вилучанні блоку до кінця матриці можна не вказувати її розміри, а використовувати елемент *end*:

```
» p = P(2, 2:end)
p =
    10 12 5
```

3.1.7. Видалення рядків та стовпців

У MATLAB парні квадратні дужки [ ] позначають порожній масив, який, зокрема, дозволяє видаляти рядки та стовпці матриці. Для видалення рядка слід присвоїти їй порожній масив. Видалимо, наприклад, перший рядок квадратної матриці:

```
» M =[2 0 3; 1 1 4; 6 1 3];
» M(1,:)=[];
» M
M =
     1 1 4
     6 1 3
```

Зверніть увагу на відповідну зміну розмірів масиву, яку можна перевірити за допомогою *size*:

```
» size(M)
ans =
     2 3
```

Аналогічно видаляються і стовпці. Для видалення кількох стовпців (або рядків), що йдуть поспіль, їм потрібно привласнити порожній масив. Видаліть другий і третій стовпець у масиві *M*

```
» M(:, 2:3) = []
M =
     1
     6
```

3.1.8. Заповнення матриць за допомогою індексації

Індексація суттєво економить час при введенні матриць, що мають певну структуру. Вище описано кілька способів введення матриць в MATLAB. Однак часто буває простіше згенерувати матрицю, ніж вводити її, особливо якщо вона має просту структуру. Розглянемо приклад такої матриці:

$$T = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & -1 & -1 \end{pmatrix}$$

Генерація матриці *T* здійснюється у три етапи:

1. Створення масиву *T* розміру п'ять на п'ять, що складається із нулів.

2. Заповнення першого рядка одиницями.

3. Заповнення частини останнього рядка мінус одиницями до останнього елемента

Відповідні команди MATLAB наведені далі.

```
» T(1:5, 1:5) = 0
T=
     0   0   0   0   0
     0   0   0   0   0
     0   0   0   0   0
     0   0   0   0   0
     0   0   0   0   0
» T(1, :) = 1
T=
     1   1   1   1   1
     0   0   0   0   0
     0   0   0   0   0
     0   0   0   0   0
     0   0   0   0   0
» T(end, 3:end) = -1
T=
     1   1   1   1   1
     0   0   0   0   0
     0   0   0   0   0
     0   0   0   0   0
     0   0  -1  -1  -1
```

Створення деяких спеціальних матриць MATLAB здійснюється за допомогою вбудованих функцій.

3.1.9. Створення матриць спеціального виду

Заповнення прямокутної матриці нулями здійснюється вбудованою функцією *zeros*, аргументами якої є число рядків та стовпців матриці:

```
» A = zeros(2, 6)
A =
     0 0 0 0 0 0
     0 0 0 0 0 0
```

Один аргумент функції *zeros* призводить до утворення квадратної матриці заданого розміру:

```
» A = zeros(3)
A =
     0 0 0
     0 0 0
     0 0 0
```

Одинична матриця ініціалізується за допомогою функції *eye*:

```
» I = eye(3)
I=
     1 0 0
     0 1 0
     0 0 1
```

Функція *eye* з двома аргументами створює прямокутну матрицю, у якій на головній діагоналі стоять одиниці, інші елементи дорівнюють нулю:

```
» I = eye(4, 8)
I = 1 0 0 0 0 0 0 0
     0 1 0 0 0 0 0 0
     0 0 1 0 0 0 0 0
     0 0 0 1 0 0 0 0
```

Матриця, що складається з одиниць, утворюється внаслідок виклику функції *ones*:

```
» E = ones(3, 7)
E =
     1  1  1  1  1  1  1
     1  1  1  1  1  1  1
     1  1  1  1  1  1  1
```

Використання одного аргументу *ones* призводить до створення квадратної матриці, що складається з одиниць.

MATLAB надає можливість заповнення матриць випадковими елементами. Результатом звернення до функції *rand* є матриця чисел, розподілених випадково між нулем і одиницею, а функції *randn* – матриця чисел, розподілених за нормальним законом:

```
» R = rand(3, 5)
R =
     0.9501 0.4860 0.4565 0.4447 0.9218
     0.2311 0.8913 0.0185 0.6154 0.7382
     0.6068 0.7621 0.8214 0.7919 0.1763
```

Один аргумент функцій *rand* и *randn* призводить до формування квадратних матриць.

Нерідко виникає необхідність створення діагональних матриць, тобто матриць, у яких усі недіагональні елементи дорівнюють нулю. Функція *diag* формує діагональну матрицю з вектор-стовпця або вектор-рядка, розташовуючи їх елементи по діагоналі матриці:

```
» d = [1; 2; 3; 4];
» D = diag(d)
D =
     1  0  0  0
     0  2  0  0
     0  0  3  0
     0  0  0  4
```

Функція *diag* служить також для вилучання діагоналі матриці у вектор, наприклад:

```
» A = [10 1 2; 1 20 3; 2 3 30];
» d = diag(A)
d = 10
    20
    30
```

3.1.10. Поелементні операції з матрицями

Оскільки вектори та матриці зберігаються у двомірних масивах, то застосування математичних функцій до матриць та поелементні операції проводяться так само, як для векторів.

Введіть дві матриці

$$A=\begin{pmatrix}2 & 5 & -1\\ 3 & 4 & 9\end{pmatrix},\ B=\begin{pmatrix}-1 & 2 & 8\\ 7 & -3 & -5\end{pmatrix}.$$

Множення кожного елемента однієї матриці на відповідний елемент іншої здійснюється за допомогою оператора .*:

```
» C = A.*B
C =
    -2  10 -8
    21 -12 -45
```

Для поділу елементів першої матриці на відповідні елементи другої використовується оператор ./, а для поділу елементів другої матриці на відповідні елементи першої служить .\:

```
» R1 = A./B
R1 =
    -2.0000  2.5000 -0.1250
     0.4286 -1.3333 -1.8000
```

```
» R2 = A.\B
R2 =
    -0.5000   0.4000 -8.0000
     2.3333 -0.7500 -0.5556
```

Поелементне зведення в ступінь здійснюється за допомогою оператора .^ . Показник ступеня може бути числом або матрицею того ж розміру, що й матриця, що зводиться до ступеня. У другому випадку елементи першої матриці зводяться в ступінь, який дорівнює елементам другої матриці.

3.1.11. Візуалізація матриць

Матриці з досить великою кількістю нулів називаються розрідженими. Нерідко потрібно знати, де розташовані ненульові елементи, тобто одержати так званий шаблон матриці. Для цього у MATLAB служить функція *spy*. Подивимося шаблон матриці *G*

$$G = \begin{pmatrix} 2 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 2 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 2 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 2 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 2 \end{pmatrix}$$

```
» spy(G)
```

Після виконання команди *spy* на екрані з'являється вікно *Figure No. 1*. На вертикальній та горизонтальній осях відкладено номери рядків та стовпців. Ненульові елементи позначені маркерами, у нижній частині графічного вікна вказано число ненульових елементів (*nz* = 19).

Наочну інформацію співвідношення величин елементів матриці дає функція *imagesc*, яка інтерпретує матрицю як прямокутне зображення. Кожен елемент матриці представляється як квадратик, колір якого відповідає величині елемента. Щоб дізнатися відповідність кольору і величини елемента, слід використовувати

команду colorbar, що виводить поруч із зображенням матриці шкалу кольору (*Insert (*у графічному вікні *Figure No. 1), colorbar*). Нарешті, для друку на монохромному принтері зручно отримати зображення у відтінках сірого кольору, використовуючи команду *colormap(gray)* (*Edit* (у графічному вікні *Figure No.* 1) *Colormap, Colormap Editor, Tools, gray*). Ми будемо працювати з матрицею *G*. Набирайте команди, вказані нижче, і слідкуйте за станом графічного вікна:

```
» imagesc(G)
» colorbar
» colormap(gray)
```

В результаті виходить наочне уявлення матриці.

3.2. Індивідуальні завдання

1. Створіть журнал виконання лабораторної роботи.

2. Введіть дві матриці *A* та *B* розмірністю три на три.

3. Виконайте над матрицями операції складання, віднімання та множення.

4. Виконайте транспонування матриць *A* та *B*.

5. Створіть з матриць *A* та *B* матрицю *C* з комплексними числами, причому елементи матриці *A* мають стати дійсними частинами комплексних чисел, а елементи матриці *B* – комплексними частинами, тобто між елементами матриць має виконуватися співвідношення:

$$c_{kj} = a_{kj} + b_{kj}i, \qquad \text{де } k, j = 1, 2, 3.$$

6. Визначте матрицю, комплексно-сполучену матриці *C*.

7. Зведіть матрицю *A* у квадрат із використанням оператора ^. Порівняйте отриманий результат, помноживши матрицю *A* саму себе.

8. Обчисліть добуток першого рядка матриці *A* та матриці *B*, а також добуток матриці B та третього стовпця матриці *A*.

9. Розв'яжіть систему лінійних рівнянь

$$a_{11}x + a_{12}y + a_{13}z = d_1,$$
$$a_{21}x + a_{22}y + a_{23}z = d_2,$$
$$a_{21}x + a_{32}y + a_{33}z = d_3,$$

де коефіцієнти системи рівнянь визначаються номером зі списку журналі групи з табл. 3.1.

10. Створіть з матриць A і B та транспонованих матриць A^T и B^T блочную матрицю $K = \begin{pmatrix} A & B \\ B^T & A^T \end{pmatrix}$.

11. Видаліть другий стовпець і третій рядок із матриці K.

12. Заповніть прямокутну матрицю розмірами щонайменше 5×8 нулями за допомогою функції *zeros*.

13. Ініціалізуйте за допомогою функції *eye* прямокутну та квадратну одиничні матриці, що містять не менше п'яти одиниць.

14. Створіть за допомогою функції *ones* прямокутну та квадратну матриці з одиниць, що містять не менше чотирьох рядків.

15. Створіть за допомогою функції *rand* матрицю 4×5 з чисел, розподілених випадково в діапазоні від 0 до N.

16. З елементів першого рядка блокової матриці за допомогою функції *diag* сформуйте діагональну матрицю.

17. Виконайте відомі поелементні операції над матрицями A і B.

18. Виконайте візуалізацію блокової матриці, використовуючи команди *spy*, *imagesc*, *colorbar*, *colormap*(*gray*).

19. Наведіть із журналу виконання лабораторної роботи кілька перших та останніх рядків.

20. Оформіть звіт з лабораторної роботи.

Таблиця 3.1 – Коефіцієнти системи рівнянь

№	a_{11}	a_{12}	a_{13}	a_{21}	a_{22}	a_{23}	a_{31}	a_{32}	a_{33}	d_1	d_2	d_3
1	1,2	0,3	0,2	0,5	2,1	1,3	–0,9	0,7	5,6	1,32	3,91	5,4
2	1,2	0,3	–0,2	0,5	2,1	1,3	–0,9	0,7	5,6	1,32	3,91	5,4
3	–1,2	0,3	–0,2	0,5	2,1	1,3	–0,9	0,7	5,6	1,32	3,91	5,4
4	1,2	–0,3	0,2	0,5	2,1	1,3	–0,9	0,7	5,6	1,32	3,91	5,4
5	–1,2	–0,3	0,2	0,5	2,1	1,3	–0,9	0,7	5,6	1,32	3,91	5,4
6	–1,2	–0,3	–0,2	0,5	2,1	1,3	–0,9	0,7	5,6	1,32	3,91	5,4
7	–1,2	0,3	0,2	0,5	2,1	1,3	–0,9	0,7	5,6	1,32	3,91	5,4
8	1,2	0,3	0,2	0,5	2,1	1,3	–0,9	0,7	5,6	1,60	3,91	5,4
9	1,2	0,3	0,2	0,5	2,1	1,3	–0,9	0,7	5,6	1,60	4,90	5,4
10	1,2	0,3	0,2	0,5	2,1	1,3	–0,9	0,7	5,6	1,81	4,53	5,4
11	1,2	0,3	0,2	0,5	2,1	1,3	–0,9	0,7	5,6	1,81	4,53	5,8
12	1,2	0,3	0,2	0,5	2,1	1,3	–0,9	0,7	5,6	2,83	4,53	5,8
13	1,2	0,3	0,2	0,5	2,1	1,3	–0,9	0,7	5,6	3,83	4,53	5,8
14	1,2	0,3	0,2	0,5	2,1	1,3	–0,9	0,7	5,6	3,32	4,53	5,8
15	1,2	0,3	0,2	0,5	2,1	1,3	–0,9	0,7	5,6	0,32	0,53	5,8
16	1,2	0,3	0,2	0,5	2,1	1,3	–0,9	0,7	–5,6	0,32	0,45	5,8
17	1,2	0,3	0,2	0,5	2,1	1,3	–0,9	0,7	–5,6	–0,32	0,45	5,8
18	1,2	0,3	0,2	0,5	2,1	1,3	–0,9	0,7	–5,6	–0,32	–0,45	5,8
19	1,2	0,3	0,2	0,5	2,1	1,3	–0,9	0,7	–5,6	–0,32	–0,35	5,8
20	1,2	0,3	0,2	0,5	2,1	1,3	–0,9	0,7	–5,6	–0,32	–0,25	–5,8
21	1,2	0,3	0,2	0,5	2,1	1,3	–0,9	0,7	–5,6	–0,32	–0,18	5,0
22	1,2	0,3	0,2	0,5	2,1	1,3	–0,9	0,7	–5,6	–0,32	–0,15	4,0
23	1,2	0,3	0,2	0,5	2,1	1,3	–0,9	0,7	–5,6	–0,32	–0,1	2,0
24	1,2	0,3	0,2	0,5	2,1	1,3	–0,9	0,7	–5,6	–0,10	–0,1	1,0
25	1,2	0,3	0,2	0,5	2,1	1,3	–0,9	0,7	–5,6	0	–0,1	0,5
26	1,2	0,3	0,2	0,5	2,1	1,3	–0,9	0,7	–5,6	0	–0,01	0
27	1,2	0,3	0,2	0,5	2,1	1,3	–0,9	0,7	–5,6	–0,32	–0,18	5,0
28	1,2	0,3	0,2	0,5	2,1	1,3	–0,9	0,7	5,6	1,81	4,53	5,8
29	1,2	0,3	0,2	0,5	2,1	1,3	–0,9	0,7	5,6	2,83	4,53	5,8
30	1,2	0,3	0,2	0,5	2,1	1,3	–0,9	0,7	5,6	3,83	4,53	5,8

Лабораторна робота 4

ПОБУДОВА БАГАТОВИМІРНИХ МАТРИЦЬ У ПАКЕТІ MATLAB

Мета лабораторної роботи: отримання та закріплення знань, формування практичних навичок роботи з пакетом MATLAB під час побудови багатовимірних матриць.

4.1. Короткі відомості з теорії

У матеріалі, який ми розглядали раніше, розмірність матриць не перевищувала значення 2. Наприклад, розмірність векторів дорівнює одиниці або 1*D* (*one-dimensional array*), розмірність прямокутних матриць дорівнює двом або 2*D* (*two-dimensional array*). На відміну від терміну «розмір» (*size*), який визначає кількість клітинок, термін "розмірність" визначає необхідну кількість індексів (або координат) для однозначного звернення до однієї клітинки матриці.

Система MATLAB дає змогу працювати з матрицями, розмірність яких може перевищувати 2*D*. Такі матриці називають багатовимірними матрицями (*imuiti-dimensionai array*). Створення таких матриць синтаксично подібне до створення 2*D*-матриць.

4.1.1. Приклад створення матриць 3D

Як приклад покажемо створення матриці 3*D* як кубика з розмірами дві комірки в кожній розмірності:

```
» C(2,2,2) = 222
» C(:,:,1)  = 0   0
               0   0
» C(:,:,2)  = 0   0
               0 222
```

Результат цієї операції потребує пояснень. Уявімо задану матрицю у вигляді рис. 4.1.

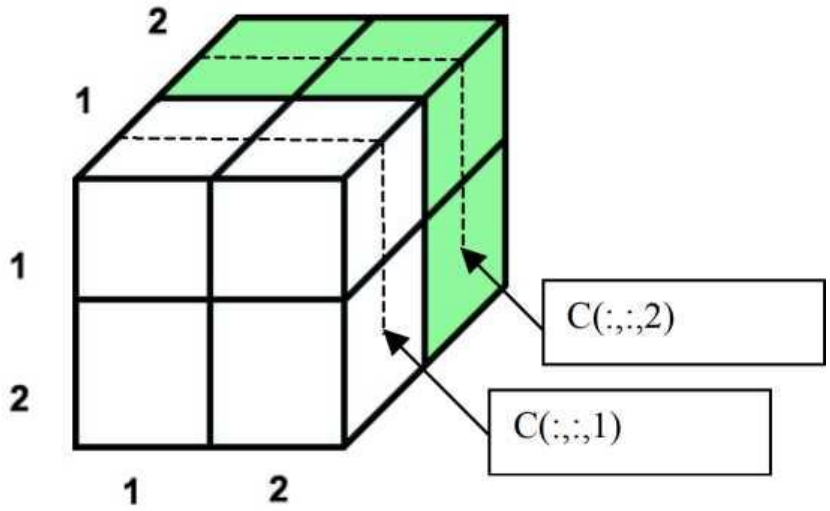


Рис. 4.1. Матриця 3*D* з розмірами дві комірки в кожній розмірності

Тоді результат, позначений як *C*(:,:,1), відобразить нам усі значення комірок її першого шару, а результат, позначений як *C*(:,:,2), усі значення комірок її другого шару.

Такий спосіб створення можна назвати як створення матриць шляхом зазначення кінцевих індексів на головній діагоналі.

4.1.2. Приклад маніпуляції значеннями та розмірністю матриць 3D

У 3*D*-матриці можна маніпулювати не тільки значеннями конкретних клітинок або векторів як у 2*D*-матриці, а й також можна змінювати цілі шари. Для цього конкретний шар необхідно зафіксувати в одній із розмірності, після чого замінити значення шару.

Приклад 4.1. Маніпуляція значеннями матриці, в якій шари фіксуються за третьою розмірністю:

```
% Сформуємо початкову/вхідну матрицю
» C(2,2,2)=222;
```

```
% Підготуємо значення, що замінюють значення шару
» B=[1,2; 3,4];
% Замінимо матрицею "B" перший шар матриці «C»
» C(:,:,1)=B
C(:,:,1) =   1 2
             3 4
C(:,:,2) =   0 0
             0 222
% Замінимо перший рядок (вектор) у другому шарі
» C(1,:,2)=[5, 6]
C(:,:,1) =   1 2
             3 4
C(:,:,2) =   5 6
             0 222
% На завершення видалимо другий шар повністю
» C(:,:,2)=[]
C =   1 2
      3 4
```

При цьому операція в цьому прикладі змінює не тільки значення матриці, а й її розмірність.

Приклад 4.2. Маніпуляція значеннями матриці, в якій шари фіксуються за першою розмірністю (рис. 4.2).

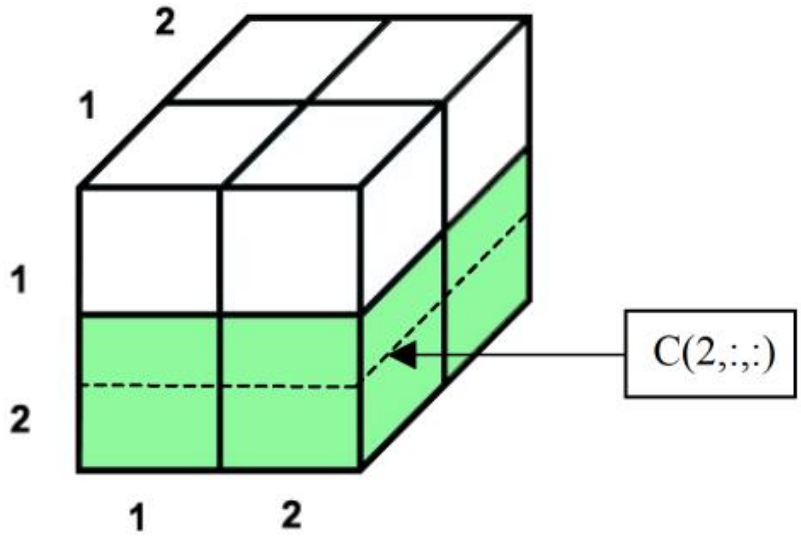


Рис. 4.2. Матриця 3*D*, у якій шари фіксуються за першою розмірністю

```
% Сформуємо вхідну матрицю
>> C(2,2,2)=222;
% Замінимо 2D-матрицею зафіксований шар матриці «C»
>> C(2,:,:)=[[1,2]; [3,4]]
C(:,:,1) =   0 0
             1 3
C(:,:,2) =   0 0
             2 4
```

Зверніть увагу, незважаючи на те, що операцію здійснювали в шарі з фіксованим рядком, результат відображається шарами (перетинами) з фіксацією за третьою розмірністю.

4.1.3. Матриці розмірністю 4D і вище

Індекси четвертої розмірності дають змогу створювати і вибірково працювати з безліччю 3*D*-матриць. Таке правило працює і для більш високих розмірностей. При цьому, з підвищенням розмірності відображення результату стає все менш і менш читабельним. У цьому легко переконається, виконавши створення, наприклад, 8*D*-матриці:

```
>> A(2,2,2,2,2,2,2,2)=0
```

Відображення результату такої операції являє собою 64 перерізи у вигляді 2*D*-матриці кожен. Це створює помітні труднощі під час планування опрацювання значень, особливо під час роботи із середніми (у сенсі розмірностей) структурами всередині таких матриць. Одним із виходів у такій ситуації можна вважати перехід до програмних засобів обробки, які ми розглянемо надалі.

4.1.4. Сервісні функції MATLAB для роботи з матрицями

Більшість функцій MATLAB мають багато варіантів списку їхніх параметрів, а відповідно і сенсу кожного з них. Детальний опис таких параметрів для кожної функції є в підсистемі ***help***.

Група функцій, яка дає змогу отримати розміри матриць, що динамічно змінюються (табл. 4.1).

Таблиця 4.1. – Функції для отримання розміру матриці.

Ім'я функції	Опис
length	Повертає довжину вектора в комірках
size	Повертає для кожної або обраної розмірності матриці довжину цієї розмірності в комірках

Група функцій для виконання типових трансформацій у матрицях (табл. 4.2).

Таблиця 4.2. – Функції для виконання трансформацій матриць.

Ім'я функції	Опис
fliplr	Виконує для векторів і 2*D*-матриць дзеркальне відображення відносно вертикалі.
flipud	Виконує для векторів і 2*D*-матриць дзеркальне відображення відносно горизонталі.
rot90	Виконує для квадратних 2*D*-матриць обертання значень матриці на кути кратні кутам 90^{o}.

Група функцій, яка дає змогу створювати та ініціалізувати типові варіанти матриць (табл. 4.3).

Таблиця 4.3. – Функції для створення матриць.

Ім'я функції	Опис
zeros	Створює матрицю заданої розмірності, комірки якої заповнені нулями.
ones	Створює матрицю заданої розмірності, комірки якої заповнені одиницями.
eye	Створює матрицю заданої розмірності, комірки головної діагоналі якої заповнені одиницями, а інші клітинки заповнені нулями.
rand	Створює матрицю заданої розмірності, комірки якої заповнені випадковими значеннями в діапазоні (0-1) відповідно до рівномірного закону розподілу.
randn	Створює матрицю заданої розмірності, комірки якої заповнені випадковими значеннями в діапазоні (0-1) відповідно до нормального закону розподілу.

У наведених таблицях 4.1. – 4.3. представлено лише найчастіше використовувані функції, умовно розділені на три категорії. Насправді, таких

функцій, особливо в категоріях 2 і 3, значно більше. Підсистема *help* за допомогою вкладки *search* і гіперпосилань дає змогу підібрати необхідну Вам функцію з безлічі функцій MATLAB.

4.2. Операції з матрицями будь-якої розмірності в MATLAB

Операції MATLAB над матрицями будь-якої розмірності підрозділяються на два класи операцій - це поелементні операції та матричні операції. Поелементні операції застосовуються до матриць будь-якої розмірності. Головне при двомісних поелементних операціях – це використання однакових розмірів у матрицях операндах. Матричні операції, як правило, застосовуються для розв'язання задач лінійної алгебри.

4.2.1. Поелементні операції для арифметичних матриць будь-якої розмірності

Узагальнене правило для не терміналу <Вираз>:

```
<Вираз> ::= < Простий вираз>
     |< Одномісна операція> <Простий вираз>
     |<Вираз> < Двомісна операція>
     |<Простий вираз> ( <Вираз> )
<Простий вираз> ::= <Операнд>
     | <Одномісна операція><Операнд>
     | <Операнд><'>
     | <Операнд><Двомісна операція><Операнд>
<Операнд> ::= <Константа>
     | <Матриця>
     | < Виклик функції>
< Виклик функції> ::= < Ім'я функції>
     | < Ім'я функції>()
     | < Ім'я функції> ( <Список параметрів> )
< Ім'я функції> ::= < Ідентифікатор>
<Список параметрів> ::= <Параметр>
     | <Параметр> , <Список параметрів>
```

Наведена формалізація не розкриває визначення не терміналів <Одномісна операція> і <Двомісна операція>. Крім того, операції в MATLAB мають вельми високий поліморфізм, тобто, сенс їхнього виконання і навіть склад суттєво залежить від типу операндів. У рамках цієї теми ми приділимо основну увагу поелементним арифметичним операціям, операндами яких є матриці різної розмірності.

Поелементними називаються операції, які виконуються над скалярними елементами матриць з однаковими індексами. Результатом є скалярні елементи матриці з індексами аналогічними індексам матриць операндів. Розміри матриць операндів мають бути рівні.

4.2.2. Одномісні поелементні операції для арифметичних матриць будь-якої розмірності

Для арифметичних матриць будь-якої розмірності визначається єдина одномісна операція (мінус):

```
<Одномісна операція для nD матриць> ::= -
```

Практично, це означає зміну знаку на протилежний для всіх значень у комірках матриці.

Приклад 4.3. Зміна знаку

```
% Нехай:
>>A
A(:,:,1) =   2 2
             2 2
A(:,:,2) =   2 2
             2 2
% Тоді:
>> -A
ans(:,:,1) = -2 -2
             -2 -2
ans(:,:,2) = -2 -2
             -2 -2
```

4.2.3. Двомісні поелементні операції для арифметичних nD-матриць і простого арифметичного операнда (скаляра).

Нагадаємо, що простий арифметичний операнд ми вже розглядали у вигляді:

```
<Простий арифметичний операнд> ::= <Арифметична константа>
    | <Ім'я змінної (розмірності 1x1)>
    | <Виклик функції (з результатом розмірності 1x1)>
```

При цьому під розмірністю 1×1 ми мали на увазі матрицю з єдиною коміркою. Слід також зазначити, що простий арифметичний операнд часто називають скаляром.

Для такого поєднання операндів двомісна арифметична операція визначається правилом:

```
<двомісна арифметична операція для nD-матриці та скаляра
> ::= - | + | * | / | ^ | ./ | .* | .^
```

Звернімо увагу, що крім символів традиційних операцій множення, ділення і взведення у ступінь (* | / | ^), використовуються їх двосимвольні позначення з лідируючою крапкою. Такі двосимвольні позначення підкреслюють необхідність виконувати операцію поелементно, тобто, виконувати відповідно для кожного осередку *nD*-матриці та осередку простого арифметичного операнда (скаляра). Необхідно підкреслити, що для операцій між скаляром і скаляром, а також матрицею і скаляром операції (* | / | ^) і (.* | ./ | .^) виконуються абсолютно однаково. Очевидно, що кінцевим результатом такої операції буде або скаляр, або *nD*-матриця.

Приклад 4.4. Нехай є вихідна кубічна матриця *A*. Для початку виконаємо її створення:

```
>> A = ones(3,3,3)
>> A * 4
ans(:,:,1) = 4  4
             4  4
             4  4
```

```
ans(:,:,2) = 4  4
             4  4
             4  4
ans(:,:,3) = 4  4
             4  4
             4  4
```

Приклад 4.5. На базі отриманих у прикладі 4.4 значень матриці *A* наведемо ще один приклад, який ілюструє можливі позиції операндів у розглянутій операції:

```
>> 4 / A
ans(:,:,1) = 1 1      % Результат
             1 1
             1 1
ans(:,:,2) = 1 1
             1 1
             1 1
ans(:,:,3) = 1 1
             1 1
             1 1
```

або (нагадаємо що *sqrt*(4) = 2)

```
>> (A.*2)./sqrt(4)
ans(:,:,1) = 1 1      % Результат
             1 1
             1 1
ans(:,:,2) = 1 1
             1 1
             1 1
ans(:,:,3) = 1 1
             1 1
             1 1
```

Приклад 4.6. Ще один важливий приклад показує, що комплексне число інтерпретується в MATLAB як арифметичний вираз, що складається з суми, дійсної та уявної частин:

```
>> A = ones(3,3,3).*2
>> A .^ (2 + 2i)
```

```
% Результат:
ans(:,:,1) = 0.7338 + 3.9321i 0.7338 + 3.9321i
             0.7338 + 3.9321i 0.7338 + 3.9321i
             0.7338 + 3.9321i 0.7338 + 3.9321i
ans(:,:,2) = 0.7338 + 3.9321i 0.7338 + 3.9321i
             0.7338 + 3.9321i 0.7338 + 3.9321i
             0.7338 + 3.9321i 0.7338 + 3.9321i
ans(:,:,3) = 0.7338 + 3.9321i 0.7338 + 3.9321i
             0.7338 + 3.9321i 0.7338 + 3.9321i
             0.7338 + 3.9321i 0.7338 + 3.9321i
```

Однак, враховуючи старшинство операцій при виконанні наступних рядків, ми вже отримаємо такий результат:

```
>> A = ones(3,3,3).*2
>> A .^ 2 + 2i
ans(:,:,1) = 4.0000 + 2.0000i 4.0000 + 2.0000i
             4.0000 + 2.0000i 4.0000 + 2.0000i
             4.0000 + 2.0000i 4.0000 + 2.0000i
ans(:,:,2) = 4.0000 + 2.0000i 4.0000 + 2.0000i
             4.0000 + 2.0000i 4.0000 + 2.0000i
             4.0000 + 2.0000i 4.0000 + 2.0000i
ans(:,:,3) = 4.0000 + 2.0000i 4.0000 + 2.0000i
             4.0000 + 2.0000i 4.0000 + 2.0000i
             4.0000 + 2.0000i 4.0000 + 2.0000i
```

4.2.4. Двомісні поелементні операції для арифметичних nD-матриць однакової розмірності та розміру.

Якщо операндами двомісної операції є *nD*-матриці однакової розмірності та розміру, то результатом операції є *nD*-матриця аналогічної розмірності та розміру, а значення її комірок обчислюватимуться як результат застосування операції до значень комірок *nD*-матриць операндів з однаковими індексами.

Математично це можна записати наступним чином:

Для кожного повністю визначеного варіанта значень індексів *nD*-матриць *inx*1, *inx*2, ..., *inxN*, поелементно виконати: *Y(inx*1, ..., *inxN) = X*1 *(inx*1, ..., *inxN) < операція > X*2 *(inx*1, ..., *inxN)*

Назвемо таку операцію *nD*- двомісною арифметичною операцією.

```
<nD- двомісна арифметична операція > ::=
- | + | .* | ./ | .^
```

Як видно з наведеного правила, операції * і / виключені зі списку можливих. Це пов'язано з тим, що виключені символи множення і ділення використовуються в MATLAB не тільки для поелементних, а й для матричних операцій, які визначаються в лінійній алгебрі. Наведемо кілька прикладів двомісних операцій для двох арифметичних *nD*-матриць однакової розмірності та розміру.

Приклад 4.7. Нехай є дві вихідні кубічні матриці *A* і *B* однакової розмірності. Для початку виконаємо їхнє створення:

```
>> A = ones(3,3,3);
>> B = ones(3,3,3).*2;
% Приклад поелементної операції ділення:
>> C = A ./ B
C(:,:,1) =   0.5000 0.5000    % Результат
             0.5000 0.5000
             0.5000 0.5000
C(:,:,2) =   0.5000 0.5000
             0.5000 0.5000
             0.5000 0.5000
C(:,:,3) =   0.5000 0.5000
             0.5000 0.5000
             0.5000 0.5000
```

Приклад 4.8. Приклад, у якому значення матриці *C* (див. приклад 4.7.) використовуються як набір показників ступеня для кожного елемента матриці B:

```
>> B .^ C
% Результат:
ans(:,:,1) = 1.4142 1.4142
             1.4142 1.4142
             1.4142 1.4142
ans(:,:,2) = 1.4142 1.4142
             1.4142 1.4142
             1.4142 1.4142
```

```
ans(:,:,3) = 1.4142 1.4142
             1.4142 1.4142
             1.4142 1.4142
```

4.3. Індивідуальні завдання

1. Створіть журнал виконання лабораторної роботи.

2. Введіть дві не нульові 3*D*-матриці *A* і *B* з розмірами дві комірки в кожній розмірності.

3. Виконайте над матрицями операції додавання, віднімання та множення.

4. Виконайте транспонування матриць *A* і *B*.

5. Зведіть 3*D*-матрицю *A* у квадрат з використанням оператора ^. Порівняйте отриманий результат, помноживши матрицю *A* саму на себе.

6. Обчисліть добуток першого шару матриці *A* і матриці *B*, а також добуток шару матриці *B* і третього шару матриці *A*.

7. Видаліть другий і третій шар матриці.

8. Заповніть 4*D*-матрицю довільного розміру нулями за допомогою функції *zeros*.

9. Створіть за допомогою функції *ones* 4*D*-матрицю з одиниць, що містять щонайменше чотири шари.

10. Створіть за допомогою функції *rand* 5*D*-матрицю, з чисел розподілених випадково в діапазоні від 0 до *N*.

11. Наведіть із журналу виконання лабораторної роботи.

12. Оформіть звіт з лабораторної роботи.

Лабораторна робота 5

УМОВНІ ВИРАЗИ, ОПЕРАТОРИ РОЗГАЛУЖЕННЯ ТА ЦИКЛИ В ПАКЕТІ MATLAB

Мета лабораторної роботи: отримання та закріплення знань, формування практичних навичок роботи з пакетом MATLAB при використанні умовних виразів, операторів розгалуження та циклів.

5.1. Короткі відомості з теорії

5.1.1. Умовні вирази (conditional expression)

Умовні вирази призначені для відповідей на запитання, задані за допомогою операцій (інструкцій) відношень і логічних операцій (інструкцій)).

Оператори відношень (*Relational operations*).

Оператори (інструкції) відношень завжди застосовуються до двох операндів, які подано у вигляді арифметичних значень або окремих текстових символів. Результатом операції є булівське значення *true* або *false*.

Оператори відношень можуть також застосовуватися до матриць однакового розміру та розмірності, елементами (комірками) яких є арифметичні значення або окремі текстові символи. У цьому випадку результатом операції буде матриця, отримана поелементним обчисленням зі значеннями *true* або *false* у відповідних елементах.

Застосування всіх операторів відношень для комплексних чисел можливе тільки щодо їхніх реальних або уявних частин. Для цього використовуються функції виділення реальної *real*(*x*) або уявної *imag*(*x*) частини таких чисел.

Безпосереднє порівняння двох комплексних чисел допускається тільки для операторів == (дорівнює) та ~= (не дорівнює).

Синтаксис і семантика операторів відношень для двох елементів (комірок) представлені в табл. 5.1.

Таблиця 5.1 – Синтаксис і семантика операторів відношень

Операція	**Опис результату операції**
A < *B*	Істина (1 або *true*), якщо лівий операнд *A* ***строго менший*** за правий операнд *B*, інакше хибність (0 або *false*).
A > *B*	Істина (1 або *true*), якщо лівий операнд *A* ***строго більший*** за правий операнд *B*, інакше хибність (0 або *false*).
A <= *B*	Істина (1 або *true*), якщо лівий операнд *A* ***менше або дорівнює*** правому операнду *B*, інакше хибність (0 або *false*).
A >= *B*	Істина (1 або *true*), якщо лівий операнд *A* ***більше або дорівнює*** правому операнду *B*, інакше хибність (0 або *false*).
A == *B*	Істина (1 або *true*), якщо лівий операнд *A* ***строго дорівнює*** правому операнду *B*, інакше хибність (0 або *false*).
A ~= *B*	Істина (1 або *true*), якщо лівий операнд *A* ***строго не дорівнює*** правому операнду *B*, інакше хибність (0 або *false*).

Приклади застосування операторів відношень

Приклад 5.1. Порівняння попередньо обчислених арифметичних виразів *X*1 і *X*2

```
X1 = 2.34;
X2 = 2;
(X1 > X2)
% Результат:
ans = 1
(X1 < X2)
% Результат:
ans = 0
```

Приклад 5.2. Порівняння попередньо обчислених комплексних виразів *X*1 і *X*2 за їхньою реальною та уявною частиною.

```
X1 = 1 + 2.34i;
X2 = 1 + 2i;
real(X1) == real(X2)
```

```
% Результат:
ans = 1
imag(X1) == imag(X2)
% Результат:
ans = 0
```

Приклад 5.3. Безпосереднє порівняння. Порівняння попередньо обчислених комплексних виразів *X*1 та *X*2

```
X1 = 1 + 2.34i;
X2 = 1 + 2i;
X1 == X2
% Результат:
ans = 0
X1 ~= X2
% Результат:
ans = 1
```

Приклад 5.4. Порівняння двох символів

```
X1='a';
X2='A';
X1 == X2
% Результат:
ans = 0
X1 < X2
% Результат:
ans = 0
X1 > X2
% Результат:
ans = 1
```

Приклад 5.5. Порівняння двох арифметичних масивів

```
X1=[1 2 3; 4 5 6];
X2=[1 2 8; 4 6 6];
X1 == X2
ans =
1   1   0
1   0   1
X1 < X2
```

```
ans =    0    0    1
         0    1    0
```

Приклад 5.6. Поелементне порівняння двох текстових рядків

```
X1='qwerty';
X2='QwErTy';
X1 == X2
ans =
    0   1   0   1   0   1
```

Приклад 5.7. Повне порівняння двох текстових рядків

```
X1='qwerty';
X2='QwErTy';
strcmp(X1,X2)
ans = 0
X1='qwerty';
X2='qwerty';
strcmp(X1,X2)
ans = 1
```

Приклад 5.8. Повне порівняння двох текстових рядків або матриць однакового розміру та розмірності за допомогою функції *isequal*

```
X1='qwerty';
X2='QwErTy';
isequal(X1,X2)
ans = 0
X1='qwerty';
X2='qwerty';
isequal(X1,X2)
ans = 1
X1=[1 2 3; 4 5 6];
X2=[1 2 8; 4 6 6];
isequal(X1,X2)
ans = 0
X1=[1 2 3; 4 5 6];
X2=[1 2 3; 4 5 6];
isequal(X1,X2)
ans = 1
```

Приклади обчислення відношень алгоритмічним шляхом.

Приклад 5.9. Повне алгоритмічне порівняння двох текстових рядків

```
X1='qwerty';
X2='QwErTy';
Y1=(X1 == X2);
% Обчислити добуток усіх елементів Y1
% за допомогою явного перерахування індексу в циклі
k=1;
for n = 1 : length(Y1)
k=k * Y1(n);
end;
% Відобразити результат порівняння
if k == 0
disp('Рядки X1 та X2 НЕ дорівнюють');
end;
```

Приклад 5.10. Повне алгоритмічне порівняння двох текстових рядків

```
X1='qwerty';
X2='QwErTy';
Y1=(X1 == X2);
% Обчислити добуток усіх елементів Y1
% за допомогою непрямого перерахування індексу в циклі
n = 1;
for k = Y1
n = n * k;
end;
% Відобразити результат порівняння
if n == 0
disp('Рядки X1 та X2 НЕ дорівнюють');
end;
```

Приклад 5.11. Повне алгоритмічне порівняння двох текстових рядків з використанням функцій *prod* (перемножити елементи масиву) і *double* (перетворити елементи масиву до арифметичного типу).

```
X1='qwerty';
X2='QwErTy';
Y1=(X1 == X2);
```

```
% Обчислити добуток усіх елементів Y1
n = prod(double(Y1));
% Відобразити результат порівняння
if n == 0
disp('Рядки X1 та X2 НЕ дорівнюють');
end;
```

5.1.2. Логічні оператори та операції (Logical operations)

Логічні операції «І», «АБО», «НЕ» оперують з обчисленими відношеннями і (або) логічними змінними, які приймають значення ІСТИНА (*true* або 1) або ХИБНІСТЬ (*false* або 0). Результатом логічної операції також є булівське значення *true* або *false*.

Логічна операція «І».

Логічна операція І виконує логічне множення двох змінних, кожна з яких може містити тільки булевське значення *true* або *false*. Варіанти результатів операції "І" подано в такій таблиці:

Y	**0**	**1**
0	0	0
1	0	1

Іншими словами результат операції І дорівнює *true* тільки коли обидва операнди *X* і *Y* дорівнюють *true*.

В MATLAB операція І позначається символом & для матриць (поелементне І) або && для скалярних величин.

Приклад 5.12. Явне обчислення операції І

```
true & false % матрична форма
% Результат:
ans = 1
true && false % скалярная форма
% Результат:
ans = 1
true & true
```

```
% Результат:
ans = 1
false & false
% Результат:
ans = 0
```

Приклад 5.13. Операція І для двох логічних матриць *Y*1 та *Y*2

```
X1=[1 2 3; 4 5 6];
X2=[1 2 8; 4 6 6];
Y1=(X1==X2)
% Результат:
Y1 = 1 1 0
     1 0 1
Y2=(X1~=X2)
% Результат:
Y2 = 0 0 1
     0 1 0
Z1=(Y1 & Y2)
% Результат:
Z1 = 0 0 0
     0 0 0
```

Логічна операція «АБО».

Логічна операція АБО виконує логічне додавання двох змінних, кожна з яких може містити тільки булевське значення *true* або *false*. Варіанти результатів операції «АБО» представлені в такій таблиці:

Y	**0**	**1**
0	0	1
1	1	1

Іншими словами результат операції АБО дорівнює *false* тільки коли обидва операнди *X* і *Y* дорівнюють *false*.

В MATLAB операція АБО позначається символом | для матриць (поелементне АБО) або || для скалярниых величин.

Приклад 5.14. Явне обчислення операції АБО

```
true | false % матрична форма
% Результат:
ans = 1
true || false % скалярная форма
% Результат:
ans = 1
true | true
% Результат:
ans = 1
false | false
% Результат:
ans = 0
```

Приклад 5.15. Операція АБО для двох логічних матриць *Y*1 та *Y*2

```
X1=[1 2 3; 4 5 6];
X2=[1 2 8; 4 6 6];
Y1=(X1==X2)
% Результат:
Y1 =
     1 1 0
     1 0 1
Y2=(X1~=X2)
% Результат:
Y2 =
     0 0 1
     0 1 0
Z1=(Y1 | Y2)
% Результат:
Z1 =
     1 1 1
     1 1 1
```

Логічна операція «НІ».

Логічна операція «НІ» виконує інвертування значення змінної, яка може містити тільки булевське значення *true* або *false*. Варіанти результатів операції «НІ» подано в такій таблиці:

X	НІ(X)
0	1
1	0

Іншими словами, результат операції «НІ» дорівнює *false* тільки тоді, коли операнд *X* дорівнює *true*.

В MATLAB операція «НІ» позначається символом ~ як для матриць (поелементне «НІ») так і для скалярних величин.

Приклад 5.16. Явне обчислення операції «НІ»

```
~true
% Результат:
ans = 0
~false
% Результат:
ans = 1
```

Приклад 5.17. Операція «НІ для логічної матриці *Y*1

```
X1=[1 2 3; 4 5 6];
X2=[1 2 8; 4 6 6];
Y1=(X1==X2)
% Результат:
Y1 =1 1 0
    1 0 1
Z1=~Y1
% Результат:
Z1 =0 0 1
    0 1 0
```

Логічна операція «ВИКЛЮЧНЕ АБО».

Логічна операція «ВИКЛЮЧНЕ АБО» виконує двійкове додавання без перенесення для двох змінних, кожна з яких може містити тільки булевське значення *true* або *false*. Варіанти результатів операції «ВИКЛЮЧНЕ АБО» подано в такій таблиці:

X / Y	**0**	**1**
0	0	1
1	1	0

Іншими словами, результат операції «Виключне АБО» дорівнює *false* тільки коли обидва операнди *X* і *Y* дорівнюють *false* або *true*.

В MATLAB операція «Виключне АБО» виконується тільки за допомогою функції *Z = xor*(*X*, *Y*) як для матриць (поелементне «Виключне АБО»), так і для скалярних величин.

Приклад 5.18. Явне обчислення операції «Виключне АБО»

```
xor(true,false)
% Результат:
ans = 1
xor(true,true)
% Результат:
ans = 0
xor(false,false)
% Результат:
ans = 0
```

Приклад 5.19. Операція «Виключне АБО» для двох логічних матриць *Y*1 і *Y*2

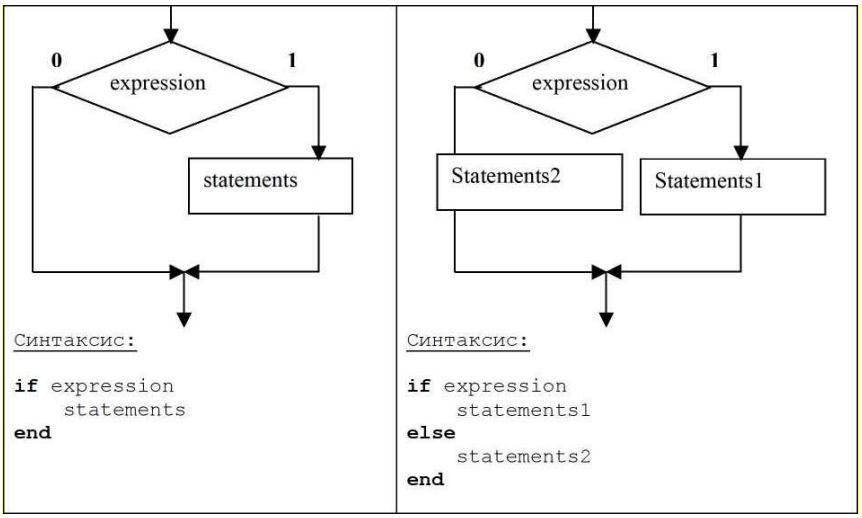
```
X1=[1 2 3; 4 5 6];
X2=[1 2 8; 4 6 6];
Y1=(X1==X2)
% Результат:
Y1 =1 1 0
    1 0 1
Y2=(X1~=X2)
% Результат:
Y2 =0 0 1
    0 1 0
Z1=xor(Y1, Y2)
Z1 =1 1 1
    1 1 1
```

5.1.3. Умовні оператори (інструкції)

5.1.3.1. Простий умовний оператор if

Приклад 5.20. Умовний оператор з однією гілкою по *true*

```
x=pi/7;
if sin(x) < cos(x)
disp('Для заданого x правильно sin(x) < cos(x)');
end;
```



Приклад 5.21. Умовний оператор з гілками *true* і *false*

```
x=pi/7;
if sin(x) >= cos(x)
disp('Для заданого x правильно sin(x) >= cos(x)');
else
disp('Для заданого x правильно sin(x) < cos(x)');
end;
```

Приклад 5.22. Перевірка на потрапляння заданого числа *X* у діапазони *X*1 … *X*2 і *X*3 ... *X*4

```
clc;
clear all;
x1=1; x2=3; x3=5; x4=7;
x=input('Введіть деяке число>>');
y1=(x>=x1)&(x<=x2);
y2=(x>=x3)&(x<=x4);
```

```
y= y1|y2;
if y
disp('число в одному з діапазонів');
else
disp('число поза діапазонами');
end;
```

5.1.3.2. Селектор за різними умовами if..elseif

Умовний оператор *if* у загальному випадку записується таким чином:

```
if Умова
    Інструкції_1
elseif Умова
    Інструкції_2
else
    Інструкції_3
end
```

Ця конструкція допускає кілька приватних варіантів. У найпростішому, типу *if_end*

```
if  Умова Інструкція end
```

доки Умова повертає логічне значення 1 (тобто «істина»), виконуються Інструкції, що становлять тіло структури *if_end*. При цьому оператор *end* вказує на кінець переліку Інструкцій. Інструкції у списку розділяються оператором "," (кома) або ";" (точка з комою). Якщо Умова не виконується (дає логічне значення 0, "не правда"), то Інструкції також не виконуються.

Ще одна конструкція

```
if  Умова Інструкції_1, else Інструкції _2, end
```

виконує `Інструкції_1` якщо виконується `Умова`, або `Інструкції_2` в іншому випадку.

Умови записуються у вигляді:

```
Вираз_1 Оператор_відношення Вираз_2
```

Причому в якості `Операторів_відношення` використовуються такі оператори: ==, <, >, <=, >= або ~=. Усі ці оператори являють собою пари символів без пробілів між ними.

Приклад 5.23. Простий селектор за різними умовами

```
x1='A';
x2='B';
if (x1 == 'A') & (x2 == 'B')
disp('Задані символи A і B');
elseif (x1 ~= 'A') & (x2 == 'B')
disp('Задано тільки символ A');
elseif (x1 == 'A') & (x2 ~= 'B')
disp('Задано тільки символ B');
end;
```

Приклад 5.24. Селектор за різними умовами з блоком *else*

```
x1='C';
x2='D';
if (x1 == 'A') & (x2 == 'B')
disp('Задані символи A і B');
elseif (x1 ~= 'A') & (x2 == 'B')
disp('Задано тільки символ A');
elseif (x1 == 'A') & (x2 ~= 'B')
disp('Задано тільки символ B');
else
disp('Символи A і B НЕ задані');
end;
```

5.1.3.3. Селектор за різними значеннями

```
switch switch_expr
  case case_expr
    statement,...,statement
  case {case_expr1,case_expr2,case_expr3,...}
    statement,...,statement
...
  otherwise
    statement,...,statement
end
```

Приклад 5.25. Селектор за різними значеннями рядків

```
str1 = 'Bilinear';
switch lower(str1)
case {'linear','bilinear'}
disp('Заявлено ім'я linear методів');
case 'cubic'
disp('Заявлено ім'я cubic методу');
case 'nearest'
disp('Заявлено ім'я nearest методу');
otherwise
disp(' Заявлено ім'я Unknown методу.');
end;
```

Приклад 5.26. Селектор за різними значеннями діапазонів

```
x1 = 12;
switch x1
case 1
disp(' Виявлено x1 = 1');
case {2,3,4,5,6,7,8}
disp(' Виявлено x1 в диапазоне [2:1:8]');
case {9,10,11,12,13,14,15,16}
disp(' Виявлено x1 в диапазоне [9:1:16]');
otherwise
disp(' x1 поза областями виявлень');
end;
```

5.1.4. Цикли

5.1.4.1. Цикл FOR. Синтаксис.

Цикл призначений повторювати інструкції (*statements*) певну кількість разів. Для цього, в заголовку циклу, використовується конструкція, яка в сучасних мовах програмування узагальнена терміном - ітератор: *variable = expression*.

```
for variable = expression statements
end
```

В MATLAB ця конструкція визначає змінну (*variable*), яка може використовуватися в тілі циклу, а також визначає собою кількість повторень тіла циклу.

Стовпці виразу (*expression*), навіть якщо вони складаються лише з одного елемента, зберігаються по одному у змінній (*variable*) і доступні для кожної інструкції (*statements*) у тілі циклу. Зауважимо, що на практиці вираз (*expression*) майже завжди має вигляд *scalar*, і в цьому випадку його стовпці є просто скалярами.

Область дії оператора *for* завжди закінчується відповідним *end*.

Приклад 5.27. Вкладені цикли з одиничним кроком. (Матриця Гільберта):

```
k = 8;
a = zeros(k,k); % матриця попереднього виділення
for m = 1:k
for n = 1:k
a(m,n) = 1/(m+n -1);
end;
end;
plot(a);
```

Приклад 5.28. Цикл із заданим діапазоном 1.0 ... 0.0 і кроком - 0.1

```
a=[];
ind = 1;
for S = 1.0: -0.1: 0.0
a(ind) = S;
ind = ind + 1;
end
plot(a);
```

Приклад 5.29. Послідовна вибірка в тіло циклу елементів із вектор-рядка

```
B = [3, 4, 5];
for S = B
S
end;
```

Приклад 5.30. Вибірка в тіло циклу вектор- стовпця (повністю)

```
B = [3; 4; 5];
for S = B
S
end;
```

Приклад 5.31. Вибірка в тіло циклу чергового вектор-стовпчика з двовимірної матриці

```
A=[[1,2,3]; [4,5,6]; [7,8,9]]
for S = A
S
end;
% Еквівалентна форма прикладу (4.3.1.5)
A=[[1,2,3]; [4,5,6]; [7,8,9]]
[d1,d2] = size(A)
for k = 1:d2
V = A(:,k)
end
% Еквівалентна форма прикладу (5)
A=[[1,2,3]; [4,5,6]; [7,8,9]]
[d1,d2] = size(A)
for k = 1:d2, V = A(:,k)
V
end;
```

Приклад 5.32. Послідовне встановлення значень *sqrt*(*m*) замість одиниць в одиничних *n*-векторах:

```
n=5;
m = 2;
for B = eye (n)
B.*sqrt(m)
end;
```

Приклад 5.33. Публікація стовпців у шарах 3*D* матриці

```
A=rand(2,2,2)
for S = A
S
end;
```

Приклад 5.34. Створення та ініціалізація вектор- рядка

```
% Варіант 1
for m=1:10
```

```
x(m)=m;
end
% Варіант 2
for m=1:10
x(1,m)=m;
end
```

Приклад 5.35. Створення та ініціалізація вектор- стовпця

```
for m=1:10
y(m,1)=m;
end
```

Приклад 5.36. Ініціалізація вектор- рядка

```
x=[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0];
for m=1:length(x)
x(m)=m;
end
```

Приклад 5.37. Додавання елементів вектор-рядка

```
x=rand(1,5);
sm = 0;
for m=1:length(x)
sm=sm+x(m);
end
```

Приклад 5.38. Додавання елементів вектор-стовпця

```
x=rand(1,5); % Вектор рядок
y=x' % Вектор стовпчик
sm = 0;
for m=1:length(y)
sm=sm+y(m);
end
```

Приклад 5.39. Створення та ініціалізація двовимірного масиву в режимі діалогу

```
r = input('Введіть число рядків або просто Enter >> ');
```

```
if isempty(r)
r = 10; % Значення за замовчуванням
end; ...
c = input('Введіть число стовпців або за замовчуванням
просто Enter >> ');
if isempty(c)
c = 10;
end;
for n=1:r
for m=1:c
M1(n,m) = 0;
end
end
```

Приклад 5.40. Пошук максимального елемента в масиві спочатку виконаємо копію масиву *Y* у масив *WM*

```
WM=input('Input 2D-array name (example: Y) >>');
mx =WM(1,1); ir=1; ic=1;
for r=1:size(WM,1)
for c=1:size(WM,2)
if WM(r,c) > mx
mx =WM(r,c); ir=r; ic=c;
end
end
end;
r, c, mx % Відобразити результати
```

Приклад 5.41. Пошук максимального елемента в масиві з вхідним контролем спочатку виконаємо копію масиву *Y* в масив *WM*

```
WM=input('Input 2D-array name (example: Y) >>');
if isempty(WM) || (size(WM,2)< 1)
disp('< array is incorrect> : end')
else
mx=WM(1,1); ir=1; ic=1;
for r=1:size(WM,1)
for c=1:size(WM,2)
if WM(r,c) > mx
mx =WM(r,c); ir=r; ic=c;
```

```
end
end
end;
r, c, mx % Відобразити результати
end;
```

5.1.4.2. Цикл While. Синтаксис.

Виконує тіло циклу поки умовний вираз у його заголовку (*expression*) обчислює істинне значення

```
while expression statements
end
```

Цикл *while* повторює інструкції (*statements*) свого тіла невизначену кількість разів. Інструкції виконуються, якщо дійсна частина умовного виразу в його заголовку (*expression*) має всі ненульові елементи, тобто, *true* (істина).

Прості умовні вирази в його заголовку (*expression*) зазвичай мають вигляд:

```
(умовний вираз) rel_op (умовний вираз)
```

де <*rel_op*> ::= = | ==| <|>| <=|> =| ~ =

Сфера дії оператора *while* завжди закінчується відповідним *end*.

***Expression* (умовний вираз)**

expression - як умовний вираз MATLAB, що зазвичай складається зі змінних або менших виразів, з'єднаних реляційними операторами (наприклад, *count* <*limit*) або логічними функціями (наприклад, *isreal* (*A*)).

Прості вирази можуть бути об'єднані логічними операторами (&, |, ~) в складі виразу. MATLAB оцінює складові вирази зліва направо, дотримуючись правил пріоритету оператора.

(*count* < *limit*) & ((*height* - *offset*) >= 0)

Statements. *Statements* - це одна або кілька інструкцій MATLAB, які мають виконуватися тільки в тому разі, якщо умовний вираз у його заголовку (*expression*) істинний або відмінний від нуля.

Зауваження. Якщо обчислений умовний вираз у його заголовку (*expression*) дає нескалярне значення, то кожний елемент цього значення має бути істинним або

ненульовим, щоб увесь вираз вважався істинним. Наприклад, оператор (*A*<*B*) є істинним, тільки якщо кожен елемент матриці *A* менший за його відповідний елемент у матриці *B*.

Проста інструкція

Змінна *eps* - це толерантність, яка використовується для визначення таких речей, як близькість сингулярності та рангу. Його обчислене значення є *epsilon* машини, або відстанню від 1.0 до наступного за величиною числа з плаваючою комою на вашому комп'ютері.

```
eps = 1;
while (1+eps) > 1
       eps = eps/2;
end
eps = eps*2
```

Приклад 5.42. Обчислення суми елементів вектора рядка

```
vs = rand(1,8)
k=1;
y=0;
while k <= length(vs)
y=y+vs(k);
k=k+1;
end; y
```

Приклад 5.43. Обчислення інтеграла функції *f*, заданої з рівномірним кроком *h* по осі *X* методом трапецій

```
f = [0 1 2 3 4 5]
h = 1
k=1;
y=0;
while k <= length(f)-1
y=y+(f(k)+f(k+1))/2;
k=k+1;
end;
y=y*h
```

Приклад 5.44. Обчислення основи натурального логарифма *e* як результату функції *exp*(1), що обчислюється рядом Маклорена.

```
Series_max = 32; % Максимальна довжина рядка
x=1; % Аргумент функції exp(x)
y=0; % Поточний результат
delta = 1; % Початкова абсолютна похибка
k=1; % Лічильник спроб
p=1; % Поточний ступінь
f=1; % Поточний факторіал
while (k < 32) % Обчислимо ряд для функції exp(1)
delta = p/f;
y=y + delta;
f=f*k;
p=p*x;
k=k+1;
if (abs(delta) < 1e-10)
break;
end;
end;
y   % Виведення результату і значення помилки
abserror=y-exp(1)
```

5.2. Індивідуальне завдання

1. Виконайте всі приклади з теоретичної частини лабораторної роботи.

2. Відповідно до номера *N* за списком у журналі групи, записаному у вигляді $N = CM$, де C – старша цифра, M – молодша цифра, визначити один із 10 алгоритмів, які треба реалізувати (табл. 5.2.), використовуючи техніку діалогового введення вихідних даних.

3. Оформіть звіт з лабораторної роботи.

Таблиця 5.2. – Перелік алгоритмів для реалізації

M	**Алгоритм**
1	Середнє значення елементів табличної функції
2	Обчислення інтеграла від табличної функції квадратурами прямокутників, трапецій і квадратурами Сімпсона
3	Добуток елементів табличної функції
4	Обчислення цілочисельного ступеня числа
5	Обчислення факторіала числа
6	Обчислення степеневих поліномів
7	Знаходження дійсного кореня алгебраїчного рівняння методом бісекції (дихотомії)
8	Обчислення елементарних функцій степеневими рядами (рядами Тейлора, Маклорена)
9	Пошук мінімального та максимального значень табличної функції
0	Найпростіше (бульбашкове сортування) табличних функцій

Лабораторна робота 6

М-ФАЙЛИ ТА ОСНОВИ ПРОГРАМУВАННЯ В MATLAB

Мета лабораторної роботи: отримання та закріплення знань, формування практичних навичок роботи з пакетом MATLAB при використанні М-файлів та виконання програм з нелінійною структурою.

6.1. Короткі відомості з теорії

6.1.1. Робота в редакторі М-файлів

Робота з командного рядка MATLAB ускладнена, якщо вимагається вводити багато команд і часто їх змінювати. Ведення щоденника за допомогою команди *diary* і збереження робочого середовища лише трохи полегшує роботу. Найзручнішим способом виконання команд MATLAB є використання М-файлів, в яких можна набирати команди, виконувати їх всі одразу або частинами, зберігати у файлі і використати надалі. Для роботи з М-файлами призначений редактор М-файлів. За допомогою цього редактора можна створювати власні функції і викликати їх, у тому числі і з командного рядка.

Розкрийте меню *File* основного вікна MATLAB і в пункті *New* виберіть підпункт *M-file*. Новий файл відкривається у вікні редактора М-файлів.

Наберіть в редакторі команди, що призводять до побудови двох графіків в одному графічному вікні:

```
» x = [0:0.1:7];
» f = exp(-x);
» subplot(1, 2, 1)
» plot(x, f)
» g = sin(x);
» subplot(1, 2, 2)
» plot(x, g)
```

Збережіть тепер файл з ім'ям *mydemo.m* в підкаталозі work основного каталогу MATLAB, вибравши пункт *Save as* меню *File* редактора. Для запуску на виконання усіх команд, що містяться у файлі, слід вибрати пункт *Run* в меню *Debug*. На екрані з'явиться графічне вікно *Figure No.1,* що містить графіки функцій. Якщо Ви вирішили побудувати графік косинуса замість синуса, то просто змініть рядок *g = sin*(*x*) в М-файлі на *g* = cos(*x*) і запустіть усі команди знову.

Зауваження 1

Якщо при наборі зроблено помилку і MATLAB не може розпізнати команду, відбувається виконання команд до неправильно введеної, після чого виводиться повідомлення про помилку в командне вікно.

Дуже зручною можливістю, що надається редактором М-файлів, є виконання частини команд. Закрийте вікно *Figure No.1*. Виділіть за допомогою миші, утримуючи ліву кнопку, або клавішами зі стрілками при натиснутій клавіші <*Shift*> перші чотири команди програми і виконайте їх з пункту *Evaluate Selection* меню *Text*. Зверніть увагу, що у графічне вікно виведено лише один графік, що відповідає виконаним командам. Запам'ятайте, що для виконання частини команд їх слід виділити та натиснути <*F*9>. Виконайте три команди програми і простежте за станом графічного вікна. Потренуйтеся самостійно, наберіть приклади з попередніх лабораторних робіт у редакторі М-файлів і запустіть їх.

Окремі блоки М-файлу можна супроводжувати коментарями, які пропускаються при виконанні, але зручні при роботі з М-файлами. Коментарі в MATLAB починаються зі знаку відсотка та автоматично виділяються зеленим кольором, наприклад: `%побудова графіку sin(x) в окремому вікні.`

У редакторі М-файлів можна одночасно відкрити кілька файлів. Перехід між файлами здійснюється за допомогою закладок з іменами файлів, що розташовані внизу вікна редактора.

Відкриття існуючого М-файлу здійснюється за допомогою пункту *Open* меню *File* робочого середовища або редактора М-файлів. Відкрити файл у редакторі

можна і командою MATLAB *edit* із командного рядка, вказавши як аргумент ім'я файлу, наприклад:

```
» edit mydemo
```

Команда *edit* без аргументу призводить до створення нового файлу.

Всі приклади, які зустрічаються в цій та наступних лабораторних роботах, найкраще набирати та зберігати в М-файлах, доповнюючи їх коментарями, та виконувати з редактора М-файлів. Застосування чисельних методів та програмування в MATLAB вимагає створення М-файлів.

6.1.2. Типи М-файлів

М-файли в MATLAB бувають двох типів: файл-програми (*Script M-Files*), що містять послідовність команд, та файл-функції (*Function M-Files*), в яких описуються функції, що визначаються користувачем.

Файл-програму (файл-процедуру) Ви створили під час прочитання попереднього підрозділу. Усі змінні, оголошені у файл-програмі, стають доступними у робочому середовищі після виконання. Виконайте в редакторі М-файлів файл-програму з підрозділа 6.1.1 і наберіть команду *whos* у командному рядку, щоб переглянути вміст робочого середовища. У командному вікні з'явиться опис змінних:

```
» whos
       Name      Size  Bytes  Class
       f         1x71  568    double array
       g         1x71  568    double array
       x         1x71  568    double array
Grand total  is 213 elements using 1704 bytes
```

Змінні, визначені в одному файлі-програмі, можна використовувати в інших файл-програмах і командах, що виконуються з командного рядка. Виконання команд, що містяться у файл-програмі, здійснюється двома способами:

1) із редактора М-файлів так, як описано вище;

2) з командного рядка або іншої файл-програми, при цьому ім'я М-файлу використовується як команда.

Застосування другого способу набагато зручніше, особливо якщо створена файл-програма буде неодноразово використовуватись згодом. Фактично створений М-файл стає командою, яку розуміє MATLAB. Закрийте всі графічні вікна та наберіть у командному рядку *mydemo,* з'являється графічне вікно, що відповідає командам файл-програми *mydemo.m*. Після введення команди *mydemo* MATLAB проводить наступні дії:

– перевіряє, чи є введена команда іменем будь-якої зі змінних, визначених у робочому середовищі. Якщо введено змінну, то виводиться її значення;

– якщо введено не змінну, то MATLAB шукає введену команду серед вбудованих функцій. Якщо команда виявляється вбудованою функцією, то відбувається її виконання.

Якщо введено не змінну і не вбудовану функцію, то MATLAB починає пошук М-файлу з назвою команди та розширенням *m*. Пошук починається з поточного каталогу *Current Directory*. Якщо М-файл у ньому не знайдено, то MATLAB переглядає каталоги, встановлені на шляху пошуку (*Path*). Знайдений М-файл виконується в MATLAB.

Якщо жодна з перерахованих вище дій не призвела до успіху, то виводиться повідомлення в командне вікно, наприклад:

```
» mydem
??? Undefined function or variable 'mydem'.
```

Як правило, М-файли зберігаються в каталозі користувача. Для того, щоб система MATLAB могла знайти їх, слід встановити шляхи, що вказують розташування М-файлів.

Зауваження 2

Зберігати власні М-файли поза основним каталогом MATLAB слід з двох причин. По-перше, при переустановленні MATLAB файли, які містяться в підкаталогах основного каталогу MATLAB, можуть бути знищені. По-друге, при запуску MATLAB усі файли підкаталогу *toolbox* розміщуються в пам'яті комп'ютера певним оптимальним чином так, щоб збільшити продуктивність роботи. Якщо ви записали М-файл у цей каталог, то скористатися ним можна буде лише після перезапуску MATLAB.

6.1.3. Встановлення шляхів

Починаючи з 6-ї версії MATLAB є можливість визначення поточного каталогу та шляхів пошуку. Встановлення цих властивостей здійснюється за допомогою відповідних діалогових вікон або командами з командного рядка.

Поточний каталог визначається в діалоговому вікні *Current Directory* робочого середовища. Вікно присутнє у робочому середовищі, якщо вибрано пункт *Current Directory* меню *View* робочого середовища.

Поточний каталог вибирається зі списку. Якщо його немає в списку, його можна додати з діалогового вікна *Browse for Folder*, викликаного натисканням на кнопку, розташовану праворуч від списку. Вміст поточного каталогу відображається у таблиці файлів.

Визначення шляхів пошуку здійснюється у діалоговому вікні *Set Path* навігатора шляхів, доступ до якого здійснюється з пункту *Set Path* меню *File* робочого середовища.

Для додавання каталогу натисніть кнопку *Add Folder* і в діалоговому вікні *Browse for Path* виберіть потрібний каталог. Додавання каталогу з його підкаталогами здійснюється при натисканні на кнопку *Add with Subfolders*. Шлях до доданого каталогу з'являється у полі MATLAB *search path*. Порядок пошуку відповідає розташуванню шляхів у цьому полі, першим проглядається каталог, шлях до якого розміщено вгорі списку. Порядок пошуку можна змінити або взагалі

видалити шлях до якогось каталогу, для чого виділіть каталог у полі MATLAB *search path* та визначте його положення за допомогою наступних кнопок:

Move to Top – помістити вгору списку;

Move Up – перемістити вгору на одну позицію;

Remove – видалити зі списку;

Move Down – перемістити вниз на одну позицію;

Move to Bottom – помістити вниз списку.

Після внесення змін слід зберегти інформацію про шляхи пошуку, натиснувши кнопку *Save*. За допомогою кнопки *Default* можна відновити стандартні установки, а *Revert* призначена для повернення до збережених.

Команди для встановлення шляхів

Дії встановлення шляхів в MATLAB дублюються командами. Поточний каталог встановлюється командою *cd*, наприклад, *cd c:\users\igor*. Команда *CD,* викликана без аргументу, виводить шлях до поточного каталогу. Для встановлення шляхів служить команда *path*, що викликається з двома аргументами:

path (*path*, '*c:\users\igor*') – добавляє каталог *c:\users\igor* з нижчим пріоритетом пошуку;

path ('*c: \users\igor*',*path*) – добавляє каталог *c:\users\igor* з вищим пріоритетом пошуку

Використання команди *path* без аргументів призводить до відображення на екрані списку шляхів пошуку. Видалити шлях зі списку можна за допомогою команди *rmpath*:

rmpath (*'c:\users\igor'*) видаляє шлях до каталогу *c:\users\igor* зі списку шляхів.

Зауваження 3

Не видаляйте без необхідності шляхи до каталогів, особливо до тих, у яких ви не впевнені. Видалення може призвести до того, що частина функцій, визначених у MATLAB, стане недоступною.

Приклад 6.1. Створіть у кореневому каталозі диска *D* (або будь-якому іншому диску чи каталозі, де студентам дозволено створювати свої каталоги) каталог зі своїм прізвищем, наприклад, *WORK_IVANOV* і запишіть туди М-файл *mydemo.m* під ім'ям *mydemo3.m*. Встановіть шляхи до файлу та продемонструйте доступність файлу з командного рядка. Результати наведіть у звіті з лабораторної роботи.

Варіант рішення:

1. У кореневому каталозі диска *D* створюється каталог *WORK_IVANOV*.

2. В каталог *WORK_IVANOV* записується М-файл *mydemo.m* під ім'ям *mydemo3.m.*

3. Відкривається діалогове вікно *Set Path* меню *File* робочого середовища MATLAB.

4. Натискається кнопка *Add Folder* і в діалоговому вікні *Browse for Path*, що з'явилося, вибирається каталог *WORK_IVANOV*.

5. Додавання каталогу з його підкаталогами здійснюється при натисканні на кнопку *Add with Subfolders*. Шлях до доданого каталогу з'являється у полі MATLAB *search path*.

6. Для запам'ятовування шляху натискається клавіша *Save* діалогового вікна *Set Path*.

7. Виконується перевірка правильності всіх дій шляхом набору команди *mydemo3* з командного рядка. На екрані з'явиться графічне вікно *Figure No.1*, що містить графіки функцій.

Розглянуті вище файл-програми є послідовністю команд MATLAB, вони не мають вхідних і вихідних аргументів. Для використання чисельних методів та при програмуванні власних додатків у MATLAB необхідно вміти складати файл-функції, які роблять необхідні дії з вхідними аргументами та повертають результат у вихідних аргументах. У цьому підрозділі розібрано кілька простих прикладів, що

дозволяють зрозуміти роботу з файл-функціями. Файл-функції, як і файл-процедури, створюються редакторі М-файлов.

6.1.4. Файл-функції

Файл-функції з одним вхідним аргументом

Припустимо, що в обчисленнях часто необхідно використовувати функцію

$$e^{-x}\sqrt{\frac{x^2+1}{x^4+0{,}1}}.$$

Доцільно один раз написати файл-функцію, а потім викликати її усюди, де необхідно обчислення цієї функції. Відкрийте у редакторі М-файлів новий файл та наберіть текст лістингу:

```
function f = myfun(x)
f= exp(-x)*sqrt((x^2+1)/(x^4+0.1));
```

Слово *function* у першому рядку визначає, що файл містить файл-функцію. Перший рядок є заголовком функції, в якій розміщується ім'я функції та списки вхідних та вихідних аргументів. У прикладі, наведеному в лістингу, ім'я функції *myfun*, один вхідний аргумент *x* і один вихідний – *f*. Після заголовка слідує тіло функції (воно, в даному прикладі, складається з одного рядка), де обчислюється її значення. Обчислене значення записується у *f*. Крапку з комою поставлено для запобігання виведенню зайвої інформації на екран.

Тепер збережіть файл у робочому каталозі. Зверніть увагу, що вибір пункту *Save* або *Save as* меню *File* призводить до появи діалогового вікна збереження файлу, у полі *File name* якого міститься назва *myfun*. Не змінюйте його, збережіть файл-функцію у файлі із запропонованим ім'ям.

Тепер створену функцію можна використовувати так само, як і вбудовані *sin*, *cos* та інші, наприклад, з командного рядка:

```
» y = myfun(1.3)
y =  0.2600
```

Виклик власних функцій може здійснюватися із файл-програми та з іншої файл-функції.

Попередження 1

Каталог, в якому містяться файл-функції, повинен бути поточним, або шлях до нього повинен бути доданий у шляху пошуку, інакше MATLAB просто не знайде функцію або викличе замість неї іншу з тим самим ім'ям (якщо вона знаходиться в каталогах, доступних для пошуку).

Файл-функція, наведена в лістингу, має один істотний недолік. Спроба обчислення значень функції від масиву призводить до помилки, а не масиву значень, як це відбувається при обчисленні вбудованих функцій

```
» x = [1.3 7.2];
» y = myfun(x)
??? Error using ==> ^
Matrix must be square.
Error in ==> C:\MATLABRl1\work\myfun.m
On line 2 ==> f = exp(-x)*sqrt((x^2+1)/(x^4+1));
```

Якщо ви вивчили роботу з масивами, то усунення цього недоліку не спричинить труднощів. Необхідно просто під час обчислення значення функції використовувати поелементні операції.

Змініть тіло функції, як зазначено в наступному лістингу (не забудьте зберегти зміни у файлі *myfun.m*)

```
function f = myfun(x)
f = exp(-x).*sqrt((x.^2+1)./(x.^4+0.1));
```

Тепер аргументом функції *myfun* може бути як число, так і вектор або матриця значень, наприклад:

```
» x = [1.3 7.2];
» y = myfun(x)
У =  0.2600   0.0001
```

Змінна *y*, в яку записується результат виклику функції *myfun*, автоматично стає вектором потрібного розміру.

Побудуйте графік функції *myfun* на відрізку [0, 4] з командного рядка або за допомогою файл-програми:

```
x = [0:0.5:4];
y = myfun(x);
plot(x, y)
```

MATLAB надає ще одну можливість роботи з файл-функціями - використання їх як аргументи деяких команд. Наприклад, для побудови графіка є спеціальна функція *fplot*, що замінює послідовність команд, наведену вище. При виклику *fplot* ім'я функції, графік якої потрібно побудувати, укладається в апострофи, межі побудови вказуються у векторному рядку з двох елементів *fplot*('*myfun*', [0, 4]).

Побудуйте графіки *myfun* за допомогою *plot* та *fplot* на одних осях, за допомогою *hold on*. Зверніть увагу, що графік, побудований за допомогою *fplot*, більш точно відображає поведінку функції, т. я. *fplot* сама підбирає крок аргументу, зменшуючи його на ділянках швидкої зміни функції, що відображається. Результати наведіть у звіті з лабораторної роботи.

Файл-функції з кількома вхідними аргументами

Написання файлів з кількома вхідними аргументами практично не відрізняється від випадку з одним аргументом. Усі вхідні аргументи розміщуються у списку через кому. Наприклад, наступний лістинг містить файл-функцію, яка обчислює довжину радіус-вектору точки тривимірного простору $\sqrt{x^2 + y^2 + z^2}$.

Лістинг файл-функції з кількома аргументами

```
function r = radius3(x, y, z)
r = sqrt(x.^2 + y.^2 + z.^2);
```

Для обчислення довжини радіус-вектору тепер можна використовувати функцію radius3, наприклад:

```
» R = radius3(1, 1, 1)
R =
    1.732
```

Крім функцій із кількома вхідними аргументами, MATLAB дозволяє створювати функції, які повертають кілька значень, тобто мають кілька вихідних аргументів.

Файл-функції з кількома вихідними аргументами

Файл-функції з кількома вихідними аргументами зручні при обчисленні функцій, що повертають кілька значень (в математиці вони називаються векторними функціями). Вихідні аргументи додаються через кому до списку вихідних аргументів, а сам список полягає у квадратних дужках. Хорошим прикладом є функція, що переводить час, заданий у секундах в години, хвилини та секунди. Ця файл-функція наведена в наступному лістингу.

Лістинг функції переведення секунд у години, хвилини та секунди

```
function [hour, minute, second] = hms(sec)
hour = floor(sec/3600);
minute = floor((sec-hour*3600)/60);
second = sec-hour*3600-minute*60;
```

При виклику файл-функцій з кількома вихідними аргументами результат слід записувати у вектор відповідної довжини

```
» [H, M, S] = hms(10000)
H =
    2
M =
    46
S =
    40
```

6.1.5. Основи програмування в MATLAB

Файл-функції та файл-програми, що використовуються у попередніх підрозділах, є найпростішими прикладами програм. Усі команди MATLAB, що містяться в них, виконуються послідовно. Для вирішення багатьох серйозніших завдань потрібно писати програми, в яких дії виконуються циклічно або, в залежності від деяких умов, виконуються різні частини програм. Розглянемо основні оператори, які визначають послідовності виконання команд MATLAB. Оператори можна використовувати як у файл-процедурах, так і в функціях, що дозволяє створювати програми зі складною розгалуженою структурою.

Оператор циклу for

Оператор призначений для виконання заданого числа дій, що повторюються. Найпростіше використання оператора *for* здійснюється наступним чином:

```
for count = start:step:final
% Далі йде текст програми із команд MATLAB
end
```

Тут *count* – змінна циклу, *start* – її початкове значення, *final* – кінцеве значення, а *step* – крок, на який збільшується *count* при кожному наступному заході в цикл. Цикл закінчується, як тільки значення *count* стає більшим за *final*. Змінна циклу може набувати як цілі, так й речові значення будь-якого знака. Розберемо застосування оператора циклу для деяких характерних прикладів.

Нехай потрібно вивести сімейство кривих для $x \in [0, 2\pi]$, яке задано функцією, яка залежить від параметра $y(x, a) = e^{-ax} \sin x$, для значень параметра від –0,1 до 0,1.

Наберіть текст файл-процедури в редакторі М-файлів, збережіть у файлі *FORdem1.m* і запустіть його на виконання (з редактора М-файлів або з командного рядка, набравши в ньому команду *FORdem1* та натиснувши <*Enter*>):

```
% файл-програма для побудови сімейства кривих
x = [0:pi/30:2*pi];
for a = -0.1:0.02:0.1
    y = exp(-a*x).*sin(x);
    hold on
    plot(x, y)
end
```

В результаті виконання *FORdem1* з'явиться графічне вікно (рис. 6.1), яке містить потрібне сімейство кривих.

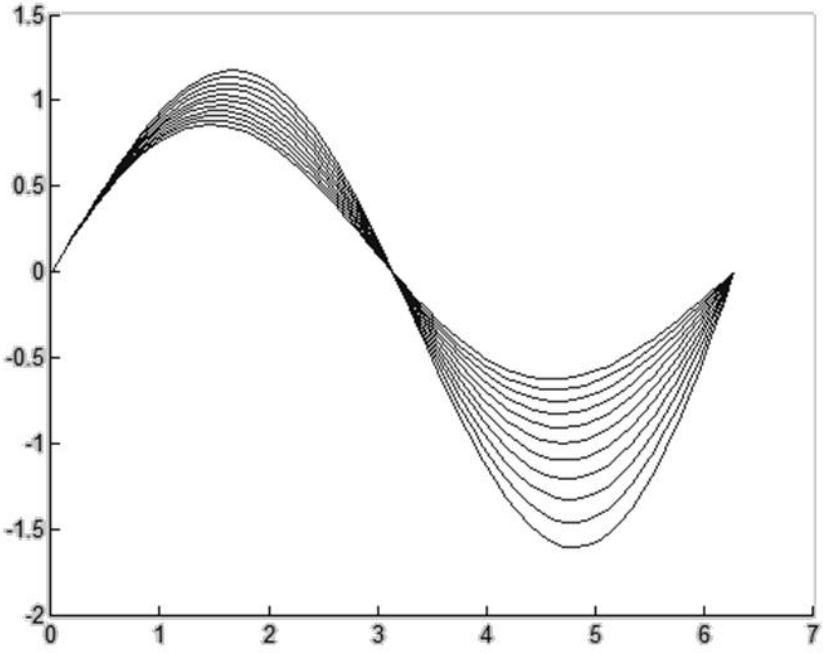


Рис. 6.1. Результат виконання *FORdem1*

Зауваження 4

Редактор М-файлів автоматично пропонує розташувати оператори всередині циклу з відступом від лівого краю. Використовуйте цю можливість для зручності роботи з текстом програми.

Напишіть файл-програму для обчислення суми

$$S = \sum_{k=1}^{10} \frac{1}{k!}.$$

Алгоритм обчислення суми використовує накопичення результату, тобто. спочатку сума дорівнює нулю (S=0), потім в змінну k заноситься одиниця, обчислюється 1/k!, додається до S і результат знову заноситься в S. Далі k збільшується на одиницю, і процес триває, поки останнім доданком не стане 1/10!. Файл-програма *Fordem2*, наведена в наступному лістингу, обчислює суму, яку шукає.

Лістинг файл-програми *Fordem2* для обчислення суми

```
% ФАЙЛ-ПРОГРАМА ДЛЯ ОБЧИСЛЕННЯ СУМИ
% 1/1!+1/2!+ … +1/10!
% Обнулення S для накопичення суми
S = 0;
% накопичення суми у циклі
for k = 1:10
    S = S + 1/factorial(k);
end
% виведення результату у командне вікно
S
```

Наберіть файл-програму в редакторі М-файлів, збережіть її у поточному каталозі у файлі *Fordem2.m* та виконайте. Результат з'явиться у командному вікні, т.я. в останньому рядку файл-програми S міститься без крапки з комою для виведення значення змінної S

```
S =
    1.7183
```

Зверніть увагу, що решта рядків файл-програми, які могли б спричинити виведення на екран проміжних значень, завершуються крапкою з комою для пригнічення виведення у командне вікно.

Перші рядки з коментарями не випадково відокремлені порожнім рядком від решти тексту програми. Саме вони виводяться на екран, коли користувач за

допомогою команди *help* з командного рядка отримує інформацію про те, що робить *Fordem2*

```
» help Fordem2
 ФАЙЛ-ПРОГРАММА ДЛЯ ОБЧИСЛЕННЯ СУМИ
 1/1!+1/2!+ … +1/10!
```

Під час написання файл-програм та файл-функцій не нехтуйте коментарями!

Усі змінні, що використовуються у файл-програмі, стають доступними у робочому середовищі. Вони є так званими глобальними змінними. З іншого боку, у файл-програмі можуть бути використані всі змінні, введені в робочому середовищі.

Розглянемо задачу обчислення суми, схожу на попередню, але яка залежить від змінної *x*

$$S = \sum_{k=1}^{10} \frac{x^k}{k!}.$$

Для обчислення цієї суми у файл-програмі *Fordem2* потрібно змінити рядок усередині циклу *for* на

```
S = S + x.^k/factorial(k);
```

Перед запуском програми слід визначити змінну *x* у командному рядку за допомогою наступних команд:

```
» x = 1.5;
» Fordem2
S =
    3.4817
```

У якості *x* може бути вектор або матриця, оскільки у файл-програмі *Fordem2* при накопиченні суми використовувалися поелементні операції.

Перед запуском *Fordem2* потрібно обов'язково привласнити змінній *x* деяке значення, а для обчислення суми, наприклад із п'ятнадцяти доданків, доведеться внести зміни до тексту файл-програми. Набагато краще написати універсальну файл-функцію, у якій вхідними аргументами будуть значення *x* і верхньої межі суми, а вихідним - значення суми *S(x)*. Файл-функція *sumN* наведена у наступному лістингу.

Лістинг файл-функції для обчислення суми

```
function S = sumN(x, N)
% файл-функція для обчислення суми
% x/1!+x^2/2!+ … +x^N/N!
% використання: S = sumN(x, N)

% обнулення S для накопичення суми
S = 0;
% накопичення суми в циклі
for m = 1:1:N
    S = S + x.^m/factorial(m);
end
```

Про використання функції *sumN* можна дізнатися, набравши в командному рядку *help sumN*. У командне вікно виведуться перші три рядки з коментарями, відокремлені від тексту файл-функції порожнім рядком.

Зверніть увагу, що змінні файл-функції не є глобальними (*m* у файл-функції s*umN*). Спроба перегляду значення змінної *m* із командного рядка призводить до повідомлення про те, що *m* не визначено. Якщо в робочому середовищі є глобальна змінна з тим же ім'ям, визначена з командного рядка або у файл-програмі, вона ніяк не пов'язана з локальною змінною у файл-функції. Як правило, краще оформляти власні алгоритми у вигляді файл-функцій для того, щоб змінні, що використовуються в алгоритмі, не змінювали значення однойменних глобальних змінних робочого середовища.

Цикли *for* можуть бути вкладені один в одного, причому змінні вкладених циклів повинні бути різними.

Цикл *for* виявляється корисним при виконанні схожих дій, що повторюються, у тому випадку, коли їх число заздалегідь визначено. Обійти це обмеження дозволяє більш гнучкий цикл *while*.

Оператор циклу while

Розглянемо приклад на обчислення суми, схожий приклад із попереднього пункту. Потрібно знайти суму ряду для заданого *x* (розкладання в ряд *sin x*):

$$S(x) = \sum_{k=0}^{\infty} (-1)^k \frac{x^{2k+1}}{(2k+1)!}.$$

Суму можна накопичувати до тих пір, поки доданки є не надто маленькими, скажімо більші по модулю за 10^{-10}. Циклом *for* тут не обійтися, оскільки заздалегідь невідома кількість доданків. Вихід полягає у застосуванні циклу *while*, який працює, поки виконується умова циклу:

while умова циклу

команди MATLAB

end

У цьому прикладі умова циклу передбачає, що поточний доданок $x^k / k!$ більше 10^{-10}. Для запису цієї умови використовується знак більше (>). Текст файл-функції *mysin*, що обчислює суму ряду, наведено у наступному лістингу.

Лістинг файл-функції *mysin*, що обчислює синус розкладанням у ряд

```
function S = mysin(x)
% Обчислення синуса розкладанням у ряд
% Використання: y = mysin(x), -pi<x<pi
S = 0;k = 0;
while abs(x.^(2*k+1)/factorial(2*k+1))>1.0e-10
    S = S + (-1)^k*x.^(2*k+1)/factorial(2*k+1);
    k = k + 1;
end
```

Зверніть увагу, що цикл *while*, на відміну від *for*, не має змінної циклу, тому довелося до початку циклу *k* присвоїти нуль, а всередині циклу збільшувати *k* на одиницю.

Умова циклу *while* може містити не тільки знак >. Для завдання умови виконання циклу допустимі інші операції відношення, наведені в табл. 6.1.

Таблиця 6.1 – Операції відношення

Позначення	Операція відношення
==	Рівність
<	Менше
>	Більше
<=	Менше або дорівнює
>=	Більше або дорівнює
~=	Не дорівнює

Завдання складніших умов проводиться із застосуванням логічних операторів. Наприклад, умова $-1 \le x < 2$ полягає в одночасному виконанні двох нерівностей $x \ge -1$ і $x < 2$ та записується за допомогою логічного оператора *and*

```
and(x >= -1, x < 2)
```

або еквівалентним чином із символом &

```
(x >= -1) & (x < 2)
```

Логічні оператори та приклади їх використання наведені в табл. 6.2.

Таблиця 6.2 – Логічні оператори

Оператор	Умова	Запис в MATLAB	Еквівалентний запис
Логічне "І"	$x < 3$ і $k = 4$	*and*(x < 3, k == 4)	(x < 3) & (k == 4)
Логічне "АБО"	$x = 1,\ 2$	*or*(x == 1,x == 2)	(x == 1) | (x == 2)
Заперечення "НІ"	$a \ne 1.9$	*not*(a == 1.9)	~(a == 1.9)

При обчисленні суми нескінченного ряду є сенс обмежити кількість доданків. Якщо ряд розходиться через те, що його члени не прагнуть нуля, то умова на мале значення поточного доданку може ніколи не виконатися і програма зациклиться. Виконайте підсумовування, додавши в умову циклу *while* файл-функції *mysin* обмеження на кількість доданків:

```
while(abs(x.^(2*k+1)/factorial(2*k+1))>1.0e-10)&
(k<=10000))
```

або в еквівалентній формі

```
while and(abs(x.^(2*k+1)/factorial(2*k+1))>1.0e-10),
k<=10000)
```

Організація повторюваних дій у вигляді циклів робить програму простою і зрозумілою, проте часто потрібно виконати той чи інший блок команд залежно від умов, тобто, використовувати розгалуження алгоритму.

Умовний оператор if

Умовний оператор *if* дозволяє створити розгалужений алгоритм виконання команд, в якому при виконанні певних умов працює відповідний блок операторів або команд MATLAB.

Оператор *if* може застосовуватися в простому вигляді для виконання блоку команд при задоволенні деякої умови або в конструкції *if-elseif-else* для написання алгоритмів, що розгалужуються.

Нехай потрібно вирахувати вираз $\sqrt{x^2-1}$. Припустимо, що обчислення виконуються в ділянці дійсних чисел і потрібно вивести попередження про те, що результат є комплексним числом. Перед обчисленням функції слід перевірити значення аргументу *x* і вивести у командне вікно попередження, якщо модуль *x* не перевищує одиниці. Тут необхідно застосування умовного оператора *if*, застосування якого у найпростішому випадку виглядає так:

if умова

команди MATLAB

end

Якщо умова виконується, то реалізуються команди MATLAB, розміщені між *if* та *end*, а якщо умова не виконується, відбувається перехід до команд, розташованим після *end*. При записі умови використовуються операції, наведені в табл. 6.1.

Файл-функція, яка перевіряє значення аргументу, наведена у наступному лістингу. Команда *warning* служить для виведення попередження у командне вікно.

Лістинг файл-функції *Rfun*, що перевіряє значення аргументу

```
function f = Rfun(x)
% обчислює sqrt(x^2-1)
% виводить попередження, якщо результат комплексний
% використання y = Rfun(x)
% перевірка аргументу
if abs(x)<1
    warning('результат комплексний')
end
% обчислення функції
f = sqrt(x^2-1);
```

Тепер виклик *Rfun* від аргументу, меншого одиниці, призведе до виведення у командне вікно попередження:

```
» y = Rfun(0.2)
y =
    0 + 0.97979589711327i     %результат комплексний
```

Файл-функція *Rfun* тільки попереджає, що її значення комплексне, проте обчислення з нею продовжуються. Якщо ж комплексний результат означає помилку обчислень, слід припинити виконання функції, використовуючи команду *error* замість *warning*.

Оператор розгалуження if-elseif-else

У випадку застосування оператора розгалуження *if-elseif-else* виглядає так:

if умова 1
    команди MATLAB
elseif умова 2
    команди MATLAB
……
elseif умова *N*
    команди MATLAB
else
    команди MATLAB
end

Залежно від виконання тієї чи іншої *N* умов працює відповідна гілка програми, якщо не виконується жодна з *N* умов, то реалізуються команди MATLAB, розміщені після *else*. Після виконання будь-якої з гілок відбувається вихід із оператора. Гілок може бути скільки завгодно або лише дві. У разі двох гілок використовується завершальне *else*, а *elseif* пропускається. Оператор розгалуження повинен завжди закінчуватись оператором *end*.

Приклад використання оператора *if-elseif-else*, в якому аналізується значення змінної *a* та виводиться повідомлення про величину *a*, наведено у наступному лістингу

```
function ifdem(a)
% приклад використання оператора if-elseif-else
if (a == 0)
    warning('a дорівнює нулю')
elseif a == 1
    warning('a дорівнює одиниці')
elseif a == 2
    warning('a дорівнює двом')
elseif a >= 3
    warning('a, більше або дорівнює трьом ')
else
    warning('a менше трьох, і не дорівнює нулю, одиниці,
двом')
end
```

Оператор розгалуження switch

Для здійснення множинного вибору або розгалуження може застосовуватись оператор *switch*. Він є альтернативою оператору *if-elseif-else*. Загалом застосування оператора розгалуження switch виглядає наступним чином:

switch вираз

case значення 1
команди MATLAB
case значення 2
команди MATLAB
……
case значення *N*
команди MATLAB
case {значення *N*+1, значення *N*+2, …}
команди MATLAB
……
case {значення *NM*+1, значення *NM*+2,…}
otherwise
команди MATLAB
end

У цьому операторі спочатку обчислюється значення виразу (це може бути скалярне числове значення чи рядок символів). Потім це значення порівнюється зі значеннями: значення 1, значення 2, …, значення *N*, значення *N*+1, значення *N*+2, …, значення *NM*+1, значення *NM*+2,… (які можуть бути числовими чи рядковими) . Якщо знайдено збіг, виконуються команди MATLAB, які стоять після відповідного ключового слова case. В іншому випадку виконуються команди MATLAB, розташовані між ключовими словами *otherwise* та *end*.

Рядків із ключовим словом *case* може бути скільки завгодно, але рядок із ключовим словом *otherwise* має бути один.

Після виконання будь-якої з гілок відбувається вихід із *switch*, при цьому значення, задані в інших *case*, не перевіряються.

Застосування *switch* пояснює наступний приклад:

```
function demswitch(x)
a = 10/5 + x
switch a
     case -1
          warning('a = -1')
     case 0
          warning('a = 0')
     case 1
          warning('a = 1')
     case {2, 3, 4}
          warning('a дорівнює 2 або 3 або 4')
otherwise
          warning('a не дорівнює -1, 0, 1, 2, 3, 4')
end
» x = -4
demswitch(x)
a = -2
» x = 1
demswitch(x)
a = 3
Warning:a дорівнює 2 або 3 або 4
```

Оператор переривання циклу break

При організації циклічних обчислень слід дбати про те, щоб усередині циклу не виникло помилок. Наприклад, нехай задано масив *x*, що складається з цілих чисел, і потрібно сформувати новий масив *y* за правилом $y(i) = x(i + 1) / x(i)$. Очевидно, що завдання може бути вирішено за допомогою циклу *for*. Але, якщо один із елементів вихідного масиву дорівнює нулю, то при розподілі вийде *inf* і наступні обчислення можуть виявитися марними. Запобігти цій ситуації можна виходом із циклу, якщо поточне значення *x(i)* дорівнює нулю. Наступний фрагмент програми демонструє використання оператора *break* для переривання циклу:

```
for x = 1:20
z = x-8;
     if z==0
          break
     end
     y = x/z
end
```

Як тільки змінна *z* набуває значення 0, цикл переривається.

Оператор *break* дозволяє достроково перервати виконання циклів *for* та *while*. Поза цими циклами оператор *break* не працює.

Якщо оператор *break* застосовується у вкладеному циклі, він здійснює вихід лише з внутрішнього циклу.

6.2. Індивідуальні завдання

1. Створіть М-файл *mydemo.m*.

2. Створіть у кореневому каталозі диска *D* (або будь-якому іншому диску чи каталозі, де студентам дозволено створювати свої каталоги) каталог зі своїм прізвищем, наприклад, *WORK_IVANOV* і запишіть туди М-файл *mydemo.m* під ім'ям *mydemo3.m*. Встановіть шляхи до файлу та продемонструйте доступність файлу з командного рядка.

3. Побудуйте графіки файл-функції *myfun* за допомогою команд *plot* та *fplot* на одних осях (за допомогою *hold on*).

4. Напишіть файл-функцію *root2*, яка знаходить лише дійсні корні квадратного рівняння, а за наявності комплексного кореню видає повідомлення про помилку. У демонстраційних прикладах другий коефіцієнт квадратного рівняння повинен дорівнювати Вашому номеру за списком у журналі групи.

5. Напишіть файл-функцію, яка знаходить найбільший спільний дільник (НОД) *z* двох натуральних чисел *x* та *y* за допомогою алгоритму Евкліда. В одному з демонстраційних прикладів найбільший спільний дільник повинен дорівнювати *3N+1*, де *N* - Ваш номер за списком у журналі групи.

Довідкова інформація. Ідея алгоритму Евкліда заснована на тому, що якщо *z*=НОД (*x, y),* то при рівності чисел *x* та *y* НОД *z* збігається з *x* та *y*, а у разі нерівності чисел *x* та *y* їх різниця між більшим і меншим разом з меншим числом має той самий

найбільший спільний дільник. Алгоритм визначення НОД Евкліда можна записати так:

Крок 1. Якщо $x > y$, то перейти на крок 4.

Крок 2. Якщо $x < y$, то перейти на крок 5, інакше перейти на крок 3.

Крок 3. $z = x$. Кінець.

Крок 4. Від x відняти y і вважати, що ця різниця тепер дорівнює значенню x. Перейти на крок 1.

Крок 5. Від y відняти x і вважати, що ця різниця тепер дорівнює значенню y. Перейти на крок 1.

6. Напишіть файл-функцію, яка знаходить прості числа, що не перевищують $150+10N$, де N – номер за списком у журналі групи.

Довідкова інформація. Простими називаються цілі позитивні числа, більше одиниці, які без залишку діляться тільки на самих себе та одиницю. Одним з найпростіших алгоритмів отримання простих чисел, що не перевершують n, є алгоритм Ератосфена, який отримав назву «решето Ератосфена». Він складається з наступних кроків:

Крок 1. Виписати послідовно всі цілі числа, починаючи з двох та закінчуючи n.

Крок 2. Задатись числом $p = 2$.

Крок 3. Якщо $p^2 \le n$, то перейти на крок 4, інакше перейти на крок 6.

Крок 4. У послідовності чисел, починаючи з числа $p + 1$, закреслити всі числа, що кратні p, не зважаючи на те, що частина чисел могла бути вже закреслена.

Крок 5. Перше після числа p незакреслене число послідовності вважати новим значенням p. Повернутися на крок 3 алгоритму.

Крок 6. Процес закінчено. Усі незакреслені числа послідовності є простими.

7. Обчисліть суму простих чисел, знайдених у завданні 6.

8. Оформіть звіт з лабораторної роботи.

Лабораторна робота 7

ПОБУДОВА ТАБЛИЦЬ ЗНАЧЕНЬ І ГРАФІКІВ ФУНКЦІЙ В ПАКЕТІ MATLAB

Мета лабораторної роботи: отримання і закріплення знань, формування практичних навичок роботи з пакетом MATLAB при побудові таблиць значень і графіків функцій.

7.1. Короткі відомості з теорії

7.1.1. Побудова таблиць значень функції однієї змінної в пакеті MATLAB

Відображення функції у вигляді таблиці зручно, якщо є порівняно невелике число значень функції. Нехай потрібно вивести в командне вікно таблицю значень функції

$$y(x) = \frac{\sin^2 x}{1+\cos x} + e^{-x}\ln x$$

в точках 0,2; 0,3; 0,5; 0,8; 1,3; 1,7; 2,5.

Задача вирішується у два етапи.

1. Створюється вектор-рядок *x*, що містить координати заданих точок.

2. Обчислюються значення функції *y*(*x*) від кожного елемента вектору *x* і записуються отримані значення в вектор-рядок *y*.

Значення функції необхідно знайти для кожного з елементів вектор-рядка *x*, тому операції у виразі для функції повинні виконуватися поелементно.

```
» x = [0.2 0.3 0.5 0.8 1.3 1.7 2.5]
x =
0.2000 0.3000 0.5000 0.8000 1.3000 1.7000 2.5000
» Y = sin(x).^2./(1+cos(x))+exp(-x).*log(x)
```

```
Y =
-1.2978   -0.8473   -0.2980   0.2030   0.8040   1.2258 1.8764
```

Зверніть увагу, що при спробі використання операцій зведення в ступінь ^, ділення / і множення * (які не належать до поелементних) виводиться повідомлення про помилку вже при зведенні sin(x) в квадрат:

```
» Y = sin(x)^2/(1+cos(x))+exp(-x)*log(x)
???  Error  using  ==>  ^
Matrix must be square.
```

У MATLAB операції * та ^ застосовуються для перемноження матриць відповідних розмірів і зведення квадратної матриці в ступінь.

Таблиці можна надати більш зручний для читання вид, розташувавши значення функції безпосередньо під значеннями аргументу:

```
» x
x =
0.2000 0.3000 0.5000 0.8000 1.3000 1.7000 2.5000
» y
y =
-1.2978   -0.8473   -0.2980   0.2030   0.8040   1.2258 1.8764
```

Часто потрібно вивести значення функції в точках відрізка, віддалених один від одного на рівну відстань (крок). Припустимо, що необхідно вивести таблицю значень функції $y(x)$ на відрізку [1, 2] з кроком 0.2. Можна, звичайно, ввести вектор-рядок значень аргументу x = [1, 1.2, 1.4, 1.6, 1.8, 2.0] з командного рядка і обчислити всі значення функції так, як описано вище. Однак якщо крок буде не 0.2, а, наприклад, 0.01, то доведеться велика робота по введенню вектора x.

У MATLAB передбачено просте створення векторів, кожен елемент яких відрізняється від попереднього на постійну величину, тобто на крок. Для введення таких векторів служить двокрапка (не плутайте з індексацією за допомогою

двокрапки). Наступні два оператора призводять до формування однакових вектор-рядків. Умовно можна записати:

```
» x = [1, 1.2, 1.4, 1.6, 1.8, 2.0]
x =
    1.0000 1.2000 1.4000 1.6000 1.8000 2.0000
» x = [1:0.2:2]
x =
    1.0000 1.2000 1.4000 1.6000 1.8000 2.0000
```

де *x* = [початкове значення : крок : кінцеве значення].

Необов'язково піклуватися про те, щоб сума передостаннього значення кроку дорівнювала б кінцевому значенню, наприклад, при виконанні наступного оператора присвоювання:

```
» x = [1:0.2:1.9]
x =
    1.0000 1.2000 1.4000 1.6000 1.8000
```

Вектор-рядок заповниться до елемента, що не перевершує визначене нами кінцеве значення. Крок може бути і негативним:

```
» x = [1.9:-0.2:1]
x =
    1.9000 1.7000 1.5000 1.3000 1.1000
```

У разі негативного кроку для отримання непорожнього вектор-рядка початкове значення має бути більше кінцевого.

Для заповнення вектор-стовпця елементами, що починаються з нуля і закінчуються 0.5 з кроком 0.1, слід заповнити вектор-рядок, а потім використовувати операцію транспонування:

```
» x = [0:0.1:0.5]'
```

```
x =
0
0.1000
0.2000
0.3000
0.4000
0.5000
```

Зверніть увагу, що елементи вектору, які заповнюються за допомогою двокрапки можуть бути тільки дійсними, тому для транспонування можна використовувати апостроф замість точки з апострофом.

Крок, що дорівнює одиниці, допускається не вказувати при автоматичному заповненні:

```
» x = [1:5]
x =
1 2 3 4 5
```

Нехай потрібно вивести таблицю значень функції:

$$y(x) = e^{-x}\sin(10x)$$

на відрізку [0, 1] з кроком 0,05.

Для виконання цього завдання необхідно провести наступні дії:

1. Сформувати вектор-рядок x за допомогою двокрапки.

2. Обчислити значення *y* (*x*) від елементів *x*.

3. Записати результат в вектор-рядок *y*.

4. Вивести *x* і *y*.

```
» x = [0:0.05:1];
» y = exp(-x).*sin(10*x);
» x
x =
Columns 1 through 7
0 0.0500 0.1000 0.1500 0.2000 0.2500 0.3000
Columns 8 through 14
0.3500 0.4000 0.4500 0.5000 0.5500 0.6000 0.6500
```

```
Columns 15 through 21
0.7000 0.7500 0.8000 0.8500 0.9000 0.9500 1.0000
» y
Y = Columns 1 through 7
0 0.4560 0.7614 0.8586 0.7445 0.4661 0.1045
Columns 8 through 14
-0.2472 -0.5073 -0.6233 -0.5816 -0.4071 -0.1533 0.1123
Columns 15 through 21
0.3262 0.4431 0.4445  0.3413 0.1676  -0.0291  -0.2001
```

Вектор-рядки *x* і *y* складаються з двадцяти одного елемента і не поміщаються на екрані в один рядок, тому виводяться по частинах. Оскільки *x* і *y* зберігаються в двомірних масивах розмірністю один на двадцять один, то виводяться по стовпцях, кожен з яких складається з одного елемента. Спочатку виводяться стовпці з першого по сьомий (`columns 1 through 7`), потім з восьмого по чотирнадцятий (`columns 8 through 14`) і, нарешті, з п'ятнадцятого по двадцять перший (`columns 15 through 21`). Більш наочним і зручним є графічне представлення функції.

7.1.2.Побудова графіків функції однієї змінної

Графіки функцій в лінійному масштабі

MATLAB володіє добре розвиненими графічними можливостями для візуалізації даних. Розглянемо спочатку побудову найпростішого графіка функції однієї змінної на прикладі функції

$$y(x) = e^{-x}\sin(10x),$$

визначеної на відрізку $[0, 1]$. Виведення функції у вигляді графіка складається з наступних етапів:

1. Завдання вектору значень аргументу *x*.

2. Обчислення вектору *y* значень функції *y*(*x*).

3. Виклик команди *plot* для побудови графіка.

Команди для завдання вектору *x* та обчислення функції краще завершувати крапкою з комою для запобігання виведення в командне вікно

їх значень (після команди *plot* крапку з комою ставити необов'язково, тому що вона нічого не виводить у командне вікно)

```
» x = [0:0.05:1];
» y = exp(-x).*sin(10*x);
» plot(x, y)
```

Після виконання команд на екрані з'являється вікно *Figure No.1* з графіком функції. Вікно містить меню, панель інструментів і область графіка. Надалі будуть описані команди, спеціально призначені для оформлення графіка. Зараз нас цікавить сам принцип побудови графіків і деякі найпростіші можливості візуалізації функцій.

Для побудови графіка функції в робочому середовищі MATLAB повинні бути визначені два вектори однакової розмірності, наприклад *x* і *y*. Відповідний масив *x* містить значення аргументів, а *y* – значення функції від цих аргументів. Команда *plot* з'єднує точки з координатами (*x*(*i*), *y*(*i*)) прямими лініями, автоматично масштабуючи осі для оптимального розташування графіка у вікні. При побудові графіків зручно розташувати на екрані основне вікно MATLAB і вікно з графіком поруч так, щоб вони не перекривалися.

Побудований графік функції має злами. Для більш точної побудови графіка функцію необхідно обчислити *y*(*x*) в більшій кількості точок на відрізку [0, 1], тобто задати менший крок при введенні вектору *x:*

```
» x = [0:0.01:1];
» y = exp(-x).*sin(10*x);
» plot(x, y)
```

В результаті виходить графік функції у вигляді більш плавної кривої.

Порівняння декількох функцій зручно робити, відобразивши їх графіки на одних осях. Наприклад, побудуємо на відрізку [-1, -0.3] графіки функцій

$$f(x)=\sin\left(\frac{1}{x^2}\right),\quad f(x)=\sin\left(\frac{1{,}2}{x^2}\right)$$

за допомогою наступною послідовністю команд:

```
» x1 = [–1:0.005:–0.3];
» f = sin(x1.^-2);
» x2 = [–1:0.005:0.3];
» g = sin(1.2*x2.^-2);
» plot(x1, f, x2, g)
```

Аналогічним чином за допомогою завдання в *plot* через кому пар аргументів (вектор абсцис, вектор ординат), здійснюється побудова графіків довільного числа функцій.

Зауваження 1

Використання *plot* з одним аргументом - вектором призводить до побудови "графіка вектора", тобто залежності значень елементів вектору від їх номерів. Аргументом *plot* може бути і матриця, в цьому випадку на одні координатні осі виводяться графіки стовпців.

Іноді потрібно порівняти поведінку двох функцій, значення яких сильно відрізняються один від одного. Графік функції з невеликими значеннями практично зливається з оссю абсцис, і встановити його вид не вдається. У цій ситуації допомагає функція *plotyy*, яка виводить графіки у вікно з двома вертикальними осями, що мають відповідний масштаб.

Порівняйте, наприклад, дві функції: $f(x)=x^{-3}$ та $F(x)=1000\cdot(x+0{,}5)^{-4}$:

```
» x = [0.5:0.01:3];
» f = x.^-3;
» F = 1000*(x+0.5).^-4;
» plotyy(x, f, x, F)
```

При виконанні цього прикладу зверніть увагу, що колір графіка збігається з кольором відповідної йому осі ординат.

Функція *plot* використовує лінійний масштаб по обох координатних осях. Однак MATLAB надає користувачеві можливість будувати графіки функцій однієї змінної в логарифмічному або напівлогарифмічному масштабі.

Графіки функцій в логарифмічних масштабах

Для побудови графіків в логарифмічному і напівлогарифмічному масштабах служать наступні функції:

– *loglog* (логарифмічний масштаб по обох осях);

– *semilogx* (логарифмічний масштаб тільки по осі абсцис);

– *semilogy* (логарифмічний масштаб тільки по осі абсцис ординат).

Аргументи функцій *loglog, semilogx і semilogy* задаються у вигляді пари векторів значень абсцис і ординат так само, як для функції *plot*, описаної в попередньому пункті. Побудуємо, наприклад, графіки функцій $f(x) = ln(0,5x)$ та $g(x) = sin(ln(x))$ на відрізку [0.1, 10] в логарифмічному масштабі по осі *x*:

```
» x = [0.1:0.01:10];
» f = log(0.5*x);
» g = sin(log(x));
» semilogx(x, f, x ,g)
```

Завдання властивостей ліній на графіках функцій

Побудовані графіки функцій повинні бути максимально зручними для сприйняття. Часто потрібно нанести маркери, змінити колір ліній, а при підготовці до монохромної друку – задати тип лінії (суцільна, пунктирна, штрих-пунктирна і т.п.). MATLAB надає можливість керувати видом графіків, побудованих за допомогою *plot, loglog, semilogx* і *semilogy*, для чого служить додатковий аргумент, що поміщається за кожною парою векторів. Цей аргумент укладається в апострофи і складається з трьох символів, які визначають: колір, тип маркера і тип лінії.

Використовується одна, дві або три позиції, в залежності від необхідних змін. У таблиці 7.1 наведені можливі значення даного аргументу із зазначенням результату.

Якщо, наприклад, необхідно побудувати перший графік червоними точковими маркерами без лінії, а другий графік - чорною пунктирною лінією, то слід використовувати команду *plot* (*x, f, 'r.', X, g, 'k:'*).

Таблиця 7.1 – Можливі значення аргументу

Колір		Тип маркеру		Тип лінії	
y	жовтий	.	точка	-	суцільна
m	рожевий	o	кружечок	:	пунктирна
c	блакитний	*x*	хрестик	-.	штрих-пунктирна
r	червоний	+	знак "плюс"	--	штрихова
g	зелений	*	зірочка		
b	синій	*s*	квадрат		
w	білий	*d*	ромб		
k	чорний	*v*	трикутник вершиною вниз		
		^	трикутник вершиною вгору		
		<	трикутник вершиною вліво		
		>	трикутник вершиною вправо		
		p	п'ятикутна зірка		
		h	шестикутна зірка		

Оформлення графіків функцій

Зручність використання графіків багато в чому залежить від додаткових елементів оформлення: координатної сітки, підписів до осей, заголовка та легенди. Сітка наноситься командою *grid on*, підписи до осей розміщуються за допомогою *xlabel, ylabel*, заголовок дається командою *title*. Наявність декількох графіків на одних осях вимагає розміщення легенди командою *legend* з інформацією про лінії. Всі перераховані команди застосовні до графіків як в лінійному, так і в логарифмічному і напівлогарифмічному масштабах.

Наступні команди виводять графіки зміни добової температури, які забезпечені всією необхідною інформацією

```
» time = [0 4 7 9 10 11 12 13 13.5 14 14.5 15 16 17 18 20
22];
» temp1 = [14 15 14 16 18 17 20 22 24 28 25 20 16 13 13 14
13];
» temp2 = [12 13 13 14 16 18 20 20 23 25 25 20 16 12 12 11
10];
» plot(time, temp1, 'ro-', time, temp2, 'go-')
» grid on
» title('Добові температури')
» xlabel('Час (год.)')
» ylabel('Температура (С)')
» legend('10 травня', 11 травня')
```

При додаванні легенди слід врахувати, що порядок і кількість аргументів команди *legend* повинні відповідати лініям на графіку. Останнім додатковим аргументом може бути положення легенди в графічному вікні:

• – 1 - поза графіком в правому верхньому куті графічного вікна;

• 0 - вибирається краще становище в межах графіка так, щоб якомога менше перекривати самі графіки;

• 1 - у верхньому правому куті графіка (це положення використовується за замовчуванням);

• 2 - в верхньому лівому кутку графіка;

• 3 - в нижньому лівому кутку графіка;

• 4 - в нижньому правому куті графіка.

У заголовку графіка, легендою і підписах осей допускається додавання формул і зміна стилів шрифту за допомогою формату *TeX*. MATLAB виводить графіки різним кольором. Монохромний принтер надрукує графіки різними відтінками сірого кольору, що не завжди зручно. Команда *plot* дозволяє легко задати стиль і колір ліній, наприклад:

```
» plot(x,f,'k-',x,g,'k:')
```

здійснює побудову першого графіка суцільний чорною лінією, а другого - чорною пунктирною. Аргументи 'k-' і 'k:' задають стиль і колір першої і другої ліній. Тут *k* означає чорний колір, а дефіс або двокрапка - суцільну або пунктирну лінію. Вікно з графіком можна закрити, натиснувши на кнопку з хрестиком в правому верхньому куті.

7.1.3. Побудова графіків функцій двох змінних

Побудова графіка функції двох змінних в MATLAB на прямокутної області визначення змінних включає два попередніх етапи:

1. Розбиття області визначення прямокутною сіткою.

2. Обчислення значень функції в точках перетину ліній сітки і запис їх в матрицю.

Побудуємо графік функції $z\ (x, y) = x^2 + y^2$ на області визначення у вигляді квадрата $x \in [\ 0, 1\ ]$, $y \in [\ 0, 1\ ]$. Необхідно розбити квадрат рівномірною сіткою (наприклад, з кроком 0.2) і обчислити значення функцій у вузлах, позначених точками.

Зручно використовувати два двомірних масиви *x* і *y*, розмірністю шість на шість, для зберігання інформації про координати вузлів. Масив *x* складається з однакових рядків, в яких записані координати *x*1, *x*2, ..., *x*6, а масив *y* містить однакові стовпці з *y*1, *y*2, ..., *y*6. Значення функції в вузлах сітки запишемо в масив *z* такої ж розмірності (6 х 6), причому для обчислення матриці *Z* використовуємо вираз для функції, але з поелементними матричними операціями. Тоді, наприклад, *z* (3, 4) якраз дорівнюватиме значенню функції *z* (*x*, *y*) в точці (*x*3, *y*4).

Для генерації масивів сітки *x* та *y* за координатами вузлів в MATLAB передбачена функція *meshgrid*, для побудови графіка у вигляді каркасної поверхні - функція *mesh*. Наступні оператори призводять до появи на екрані вікна з графіком функції (точка з комою в кінці операторів не ставиться, щоб проконтролювати генерацію масивів):

```
» [X,Y] = meshgrid(0:0.2:1,0:0.2:1)
X=
     0         0.2000    0.4000    0.6000    0.8000    1.0000
     0         0.2000    0.4000    0.6000    0.8000    1.0000
     0         0.2000    0.4000    0.6000    0.8000    1.0000
     0         0.2000    0.4000    0.6000    0.8000    1.0000
     0         0.2000    0.4000    0.6000    0.8000    1.0000
     0         0.2000    0.4000    0.6000    0.8000    1.0000
y =
     0         0         0         0         0         0
     0.2000    0.2000    0.2000    0.2000    0.2000    0.2000
     0.4000    0.4000    0.4000    0.4000    0.4000    0.4000
     0.6000    0.6000    0.6000    0.6000    0.6000    0.6000
     0.8000    0.8000    0.8000    0.8000    0.8000    0.8000
     1.0000    1.0000    1.0000    1.0000    1.0000    1.0000
» Z = X.^2+Y.^2
Z =
     0         0.0400    0.1600    0.3600    0.6400    1.0000
     0.0400    0.0800    0.2000    0.4000    0.6800    1.0400
     0.1600    0.2000    0.3200    0.5200    0.8000    1.1600
     0.3600    0.4000    0.5200    0.7200    1.0000    1.3600
     0.6400    0.6800    0.8000    1.0000    1.2800    1.6400
     1.0000    1.0400    1.1600    1.3600    1.6400    2.0000
» mesh(X,Y,Z)
```

MATLAB дозволяє наносити на графік додаткову інформацію, зокрема, відповідність кольорів значенням функції. Сітка генерується за допомогою команди *meshgrid*, викликаної з двома аргументами. Аргументами є вектори, елементи яких відповідають сітці на прямокутної області побудови функції. Можна використовувати один аргумент, якщо область побудови функції - квадрат. Для обчислення функції слід використовувати поелементні операції.

Розглянемо основні можливості, що надаються MATLAB для візуалізації функцій двох змінних, на прикладі побудови графіка функції

$$z(x,y) = 4\sin(2\pi x)\cdot\cos(1{,}5\pi y)\cdot(1-x^2)\cdot y\cdot(1-y)$$

на прямокутної області визначення $x \in [-1, 1]$, $y \in [0, 1]$.

Підготуємо матриці з координатами вузлів сітки і значеннями функції:

```
» [X, Y] = meshgrid(-1:0.05:1, 0:0.05:1);
» Z=4*sin(2*pi*X).*cos(1.5*pi*Y).*(1-X.^2).*Y.*(1-Y);
```

Для побудови каркасної поверхні використовується функція *mesh*, що викликається з трьома аргументами:

```
» mesh(X,Y,Z)
```

Колір ліній поверхні відповідає значенням функції. MATLAB малює тільки видиму частину поверхні.

За допомогою команди *hidden off* можна зробити каркасну поверхню "прозорою", додавши приховану частину. Команда *hidden on* прибирає невидиму частину поверхні, повертаючи графіку колишній вигляд.

Функція *surf* будує каркасну поверхню графіка функції і заливає кожну клітину поверхні певним кольором, що залежать від значень функції в точках, відповідних кутів клітини. В межах кожної клітини колір постійний. Подивіться результати виконання команди

```
» surf(X,Y,Z)
```

Команда *shading flat* дозволяє прибирати каркасні лінії. Для отримання поверхні, плавно залитою кольором, що залежить від значень функції, призначена команда *shading interp.*

За допомогою *shading faceted* можна повернутися до поверхні з каркасними лініями.

Тривимірні графіки, одержувані за допомогою описаних вище команд, зручні для отримання уявлення про форму поверхні, проте по ним важко судити про значення функції. У MATLAB визначена команда *colorbar*, яка виводить поруч з графіком стовпець, який встановлює відповідність між кольором і значенням функції. Побудуйте за допомогою *surf* графік поверхні і доповніть його інформацією про колір

```
» surf(X,Y,Z)
» colorbar
```

Команду *colorbar* можна застосовувати в поєднанні з усіма функціями, що будують тривимірні об'єкти.

Користуючись кольоровою поверхнею, важко зробити висновок про значення функції в тій чи іншій точці площині *xy*. Команди *meshc* або *surfc* дозволяють отримати більш точне уявлення про поведінку функції. Ці команди будують каркасну поверхню або залиту кольором каркасну поверхню і розміщують на площині *xy* лінії рівня функції (лінії сталості значень функції):

```
» surfc(X,Y,Z)
» colorbar
```

MATLAB дозволяє побудувати поверхню, що складається з ліній рівня, за допомогою функції *contour3*. Цю функцію можна використовувати так само, як і описані вище *mesh, surf, meshc* і *surfc* з трьома аргументами. При цьому число ліній рівня вибирається автоматично. Є можливість задати четвертим аргументом в *contour3* або число ліній рівня, або вектор, елементи якого рівні значенням функції, які відображаються у вигляді ліній рівня. Завдання вектору (четвертого аргументу *levels*) зручно, коли потрібно досліджувати поведінку функції в деякій області її значень (зріз функції). Побудуйте, наприклад, поверхню, що складається з ліній рівня, відповідних значень функції від 0 до 0,5 з кроком 0,01:

```
» levels = [0:0.01:0.5];
» contour3(X, Y, Z, levels)
» colorbar
```

7.1.4. Побудова контурних графіків функцій двох змінних

MATLAB надає можливість отримувати різні типи контурних графіків за допомогою функцій *contour* і *contourf*. Розглянемо їх можливості на прикладі функції

$$z(x,y) = 4\sin(2\pi x)\cdot\cos(1{,}5\pi y)\cdot(1-x^2)\cdot y\cdot(1-y).$$

Використання *contour* з трьома аргументами *contour (X, Y, Z)* призводить до побудови графіку, на якому показані лінії рівня на площині *xy*, але без вказівки числових значень на них. Такий графік є малоінформативним, він не дозволяє дізнатися значення функції на кожній з ліній рівня. Використання команди *colorbar* також не дозволить точно визначити значення функції. Кожній лінії рівня можна привласнити значення, яке приймає на ній досліджувана функція, за допомогою функції *clabel*. Функція *clabel* викликається з двома аргументами: матрицею, що містить інформацію про лінії рівня і покажчиком на графік, на якому слід нанести розмітку. Користувачеві не потрібно самому створювати аргументи *clabel*. Функція *contour*, викликана з двома вихідними параметрами, не тільки будує лінії рівня, але і знаходить необхідні для *clabel* параметри. Використовуйте *contour* з вихідними аргументами *CMatr* і *h* (в масиві *CMatr* міститься інформація про лінії рівня, а в масиві *h* - покажчики). Завершіть рядок з функцією *contour* точкою з комою для запобігання виведення на екран значень вихідних параметрів і нанесіть на графік сітку:

```
» [CMatr, h] = contour(X, Y, Z);
» clabel(CMatr, h)
» grid on
```

Додатковим аргументом функції *contour* (так само, як і *contour3*, описаної вище) може бути або число ліній рівня, або вектор, що містить значення функції, для яких потрібно побудувати лінії рівня.

Наочну інформацію про зміну функції дає заливка прямокутника на площині *xy* кольором, що залежить від значення функції в точках площині. Для побудови таких графіків призначена функція *contourf*, використання якої не відрізняється від застосування *contour*. У наступному прикладі виводиться графік, який складається з двадцяти ліній рівня, а проміжки між ними заповнені кольорами, відповідними значенням досліджуваної функції:

```
» contourf(X, Y, Z, 20)
» colorbar
```

7.1.5. Оформлення графіків функцій

Простим і ефективним способом зміни колірного оформлення графіка є установка колірної палітри за допомогою функції *colormap*. Наступний приклад демонструє підготовку графіка функції для друку на монохромному принтері, використовуючи палітру *gray*

```
» surfc(X, Y, Z)
» colorbar
» colormap(gray)
» title('Графік функції z(x,y)')
» xlabel('x')
» ylabel('y')
» zlabel('z')
```

Зверніть увагу, що команда *colormap* (*gray*) змінює палітру графічного вікна, тобто наступні графіки будуть виводитися в цьому вікні також в сірих тонах. Для відновлення початкового значення палітри слід застосувати команду *colormap* (*'default'*). Кольорові палітри, доступні в MATLAB, наведені в табл. 7.2.

7.1.6. Виведення кількох графіків на одні осі

Для відображення декількох графіків функцій однієї змінної на одних осях використовуються можливості функцій *plot*, *plotyy*, *semilogx, semilogy, loglog*. Вони дозволяють виводити графіки декількох функцій, задаючи відповідні векторні аргументи парами, наприклад *plot (x, f, x, g)*. Однак для об'єднання тривимірних графіків їх використовувати не можна.

Для об'єднання таких графіків призначена команда *hold on*, яку потрібно задати перед побудовою графіка. У наступному прикладі об'єднання двох графіків (площині і конуса) призводить до їх перетину. Конус задається параметрично наступними залежностями:

$$x(u,v) = 0{,}3\cdot u\cdot\cos v;\ \ y(u,v) = 0{,}3\cdot u\cdot\sin v;\ z(u,v) = 0{,}6\cdot u;\ \ u,v\in[-2\pi,\ 2\pi].$$

Таблиця 7.2 – Кольорові палітри

Палітра	Зміна кольору
autumn	Плавне зміна червоний - помаранчевий - жовтий
bone	Схожа на палітру *gray*, але з легким відтінком синього кольору
colorcube	Кожен колір змінюється від темного до яскравого
cool	Відтінки блакитного та пурпурного кольорів
copper	Відтінки мідного кольору
flag	Циклічна зміна червоний - білий - синій - чорний
gray	Відтінки сірого
hot	Плавне зміна чорний - красний - помаранчевий - жовтий - білий
hsv	Плавна зміна, як кольорів веселки
jet	Плавне зміна синій - блакитний – червоний - зелений - жовтий - червоний
pink	Схожа на палітру *gray*, але з легким відтінком коричневого кольору
prism	Циклічна зміна червоний - помаранчевий - жовтий - зелений - синій - фіолетовий
spring	Відтінки пурпурного та жовтого
summer	Відтінки зеленого та жовтого
vga	Палітра Windows із шістнадцяти кольорів
white	Один білий колір
winter	Відтінок синього та зеленого

Для графічного відображення конуса спочатку необхідно згенерувати за допомогою двокрапки вектор-стовпець і вектор-рядок, що містять значення параметрів на заданому інтервалі (важливо, що *u* - вектор-стовпець, а *v* - вектор-рядок):

```
» u = [-2*pi:0.1*pi:2*pi]';
» v = [-2*pi:0.1*pi:2*pi];
```

Наступним кроком формуються матриці *X, Y*, що містять значення функцій $x(u, v)$ і $y(u, v)$ в точках, відповідних значень параметрів.

Формування матриць виконується за допомогою зовнішнього множення векторів.

Сформуємо матриці *X, Y*, необхідні для графічного відображення конуса:

```
» X = 0.3*u*cos(v);
» Y = 0.3*u*sin(v);
```

Матриця *Z* повинна бути того ж розміру, що і матриці *X* та *Y*. Крім того, вона повинна містити значення, що відповідають значенням параметрів. Якби в функцію *z(u, v)* входило добуток *u* та *v*, то матрицю *Z* можна було заповнити аналогічно матрицям *X* та *Y* за допомогою зовнішнього добутку. З іншого боку, функцію *z(u, v)* можна представити у вигляді $z(u, v) = 0{,}6 \cdot u \cdot g(v)$, де $g(v) \equiv 1$. Тому для обчислення *Z* можна застосувати зовнішній добуток векторів *u* та *g(v)*, де вектор-рядок *g(v)* має ту ж розмірність, що *v*, але складається з одиниць:

```
» Z = 0.6*u*ones(size(v));
```

Усі необхідні матриці для відображення конуса створені. Задати площину можна таким чином:

```
» [X,Y] = meshgrid(-2:0.1:2);
» Z = 0.5*X+0.4*Y;
```

Тепер не складно записати повну послідовність команд для побудови пересічних конуса і площині:

```
» u = [-2*pi:0.1*pi:2*pi]';
» v = [-2*pi:0.1*pi:2*pi];
» X = 0.3*u*cos(v);
» Y  = 0.3*u*sin(v);
» Z = 0.6*u*ones(size(v));
» surf(X, Y, Z)
» [X,Y] = meshgrid(-2:0.1:2);
» Z = 0.5*X+0.4*Y;
» hold on
» mesh(X, Y, Z)
» hidden off
```

В результаті роботи програми отримаємо геометричну фігуру, наведену на рис. 7.1.

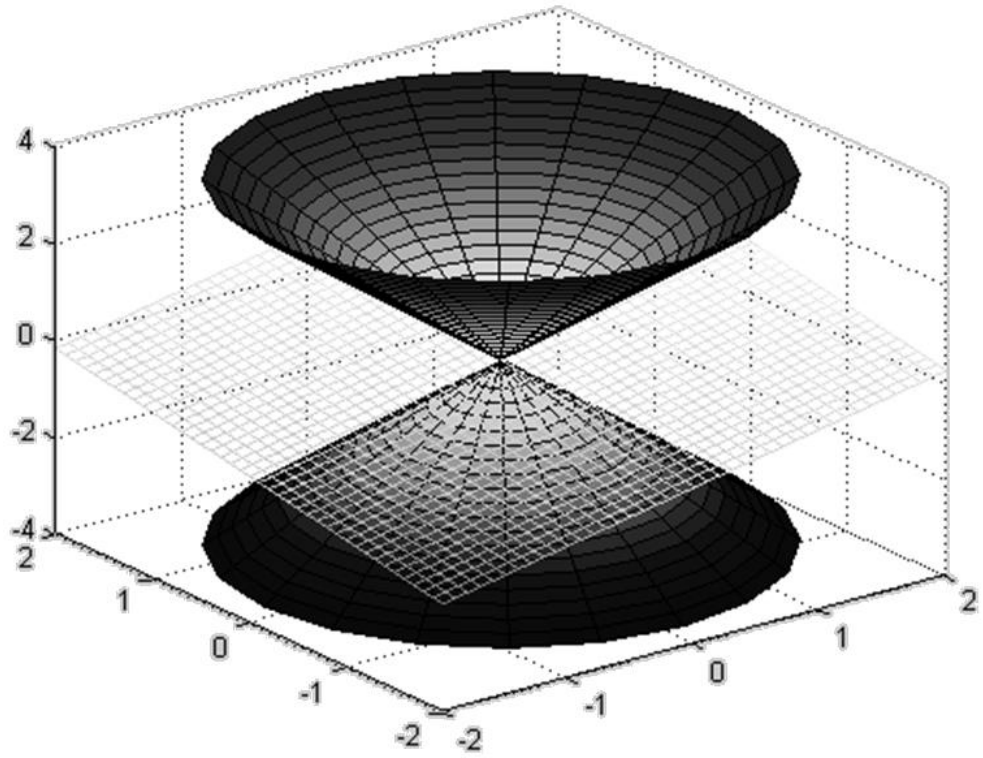


Рис. 7.1. Результат роботи програми

Команда *hidden off* застосована для того, щоб показати частину конуса, що знаходиться під площиною.

Зверніть увагу, що команда *hold on* поширюється на всі наступні висновки графіків в поточне вікно. Для розміщення графіків в нових вікнах слід виконати команду *hold off*. Команда *hold on* може застосовуватися і для розташування декількох графіків функцій однієї змінної, наприклад,

```
» plot(x,f,x,g)
```

еквівалентно послідовності

```
» plot(x,f)
» hold on
» plot(x,g)
```

7.2. Індивідуальні завдання

1. Відповідно номеру N за списком в журналі групи, записаному у вигляді $N = CM$, де C – старша цифра, M – молодша цифра, у цілочисленному інтервалі $[N, N + 5]$ розрахуйте таблицю значень для вираження, заданого за допомогою табл. 7.3 і табл. 7.4.

Таблиця 7.3 – Індивідуальне завдання по молодшій цифрі M

Молодша цифра M	0 або 5	1 або 6	2 або 7	3 або 8	4 або 9
Вираз	$\frac{A^{2/3}\cdot\sqrt{C^3}}{(B+D)^2}$	$\frac{ABC}{A^2+D}$	$\frac{A\cdot\sqrt{B}}{D^{1/3}/C^2}$	$\frac{C^{2/5}+\sqrt{D}}{B^{\frac{1}{3}}\cdot\sqrt[5]{A^2}}$	$\frac{AC\cdot\sqrt[3]{D}}{C^{2/3}+B^2}$

Таблиця 7.4 – Індивідуальне завдання по старшій цифрі C

Старша цифра C	A	B	C	D
0	$(\sin(N)\cos(N))^2$	$(\ln(N+2))/N$	$(\exp(N/N^2))^{-2}$	$A+B$
1	$1+(\cos(N+1))^2$	$(\ln(N^2))^{-2}$	$\exp(-N)+1$	$A+C$
2	$(\mathrm{tg}(N))^2+N^{-1}$	$(\log(N))^{-2}$	$1+N/\exp(N)$	B/C
3	$\sin(N)/(\mathrm{ctg}(N))^4$	$\log(N^{-3}+N^3)$	$N/\exp(-N)$	$C/(A+B)$

2. Виконайте завдання 1, використовуючи негативний крок –1.

3. Відобразіть в лінійному масштабі на одних осях два графіки функції $f(x)$ з завдання один з кроками 1.0 і 0.05 в інтервалі $[N, N + 5]$.

За допомогою функції *plotyy* побудуйте з кроком 0.1 в інтервалі $[N, N + 5]$ графіки функцій $f(x)$ та $f(x)\cdot sin(x)\cdot 10^{-2}$

4. Виконайте завдання 3, використовуючи:

- логарифмічний масштаб по обох осях;
- логарифмічний масштаб по осі абсцис;
- логарифмічний масштаб по осі ординат.

При виконанні цього пункту використовуйте шість типів маркерів, шість різних кольорів ліній і різні типи ліній.

5. Сформуйте матрицю і вектор розмірами відповідно не менше 5x6 та 1x7, першими елементами яких є Ваш номер за списком в журналі групи. Побудуйте графіки вектору і матриці.

6. Наведіть графік функції $z(x, y) = x^2 + y^2$ на області визначення у вигляді квадрата $x \in [0, 1]$, $y \in [0, 1]$ з кроком 0.2 і графік функції з меншим кроком сітки.

7. Побудуйте прозору і непрозору каркасну поверхню для функції $z(x, y) = 4sin(2\pi x) \cdot cos(1{,}5\pi y) \cdot (1 - x^2) \cdot y \cdot (1 - y)$ на прямокутній області визначення $x \in [-1, 1]$, $y \in [0, 1]$.

Змініть функцію $z(x, y)$ будь-яким чином, але таким, щоб у виразі функції фігурував Ваш номер за списком в журналі групи. Наведіть в звіті прозору і непрозору каркасну поверхню для вашої функції.

8. Побудуйте каркасну поверхню для функції $z(x, y)$ за допомогою команд *surf*(*X,Y,Z*), *shading flat*, *shading interp*. Результати наведіть у звіті.

9. Побудуйте каркасну поверхню для функції $z(x, y)$ за допомогою команд *surf*(*X,Y,Z*) та *colorbar*. Результати наведіть у звіті.

10. Побудуйте каркасну поверхню для функції $z(x, y)$ за допомогою команд *surfc*, *meshc* и *colorbar*. Результати наведіть у звіті.

11. Побудуйте поверхні функції $z(x, y)$, що складаються з ліній рівня, за допомогою функції *contour3* з трьома і чотирма аргументами. Результати приведіть у звіті.

12. Побудуйте контурні графіки функції $z(x, y)$, за допомогою функцій *contour*, *contourf*, *clabel*. Результати наведіть у звіті.

13. Виконайте три різних колірних оформлення графіка функції $z(x, y)$. Результати приведіть в звіті.

14. Виконайте побудову пересічних конуса і площині.

15. Виконайте перетин конуса двома різними площинами.

16. Оформити звіт по лабораторній роботі.

Лабораторна робота 8

ПОБУДОВА ГРАФІКІВ З ВИКОРИСТАННЯМ ФУНКЦІЙ AXIS, EZPLOT, EZPOLAR В ПАКЕТІ MATLAB

Мета лабораторної роботи: отримання та закріплення знань, формування практичних навичок роботи з пакетом MATLAB під час побудови графіків з використанням функцій *axis, ezplot, ezpolar*.

8.1. Короткі відомості з теорії

8.1.1. Функція axis. Синтаксис.

Функція *axis* призначена для масштабування та встановлення зовнішнього вигляду осей координат. Функція має велику кількість синтаксичних форм і параметрів. Багато застосувань є досить специфічними. З цієї причини обмежимося тільки тими синтаксичними формами, які найчастіше застосовуються на початкових етапах освоєння MATLAB.

- *axis* ([*Xmin Xmax Ymin Ymax*]) встановлює масштабування для осей *x* та *y* на поточній ділянці.
- *axis* ([*Xmin Xmax Ymin Ymax Zmin Zmax*]) встановлює масштабування для *x*-, *y*- і *z*-осі на поточному тривимірному графіку.
- *axis auto* повертає масштабування осі за замовчуванням, автоматично для кожного виміру обираються "хороші" ліміти в екстентах усіх ліній, поверхонь, патчів і зображень.

8.1.2. Функція axis. Приклади застосування.

Приклад 8.1. Застосування функції *axis*. Створити порожнє вікно із заданого математичного розміру.

```
help axis
close all;
axis([-4 4 -2 2]);
```

Приклад 8.2. Керування математичним розміром вікна перегляду у вікні графіка.

```
close all;
x = 0:.025:pi/2;
plot(x,tan(x),'-ro')
input('Для продовження натисніть Enter ...')
axis([0 pi/2 0 5]);
input('Для продовження натисніть Enter ...')
axis([0 pi/4 0 2]);
input('Для продовження натисніть Enter ...')
axis([0 pi/2 0 20]);
input('Для продовження натисніть Enter ...')
close all;
```

Приклад 8.3. Ручне керування вікном графіків і його масштабування.

```
close all;
axis manual;
axis([-4 4 -2 2]); % Встановити розмір вікна
hold on; % Заморозити вікно та його вертикальний масштаб
fplot('[3*sin(x)/x]', [-2*pi 2*pi], '-r');
fplot('[abs(sin(x))]',[-2*pi 2*pi]);
input('Для продовження натисніть Enter ...')
axis([-4 4 -2 6]);
input('Для продовження натисніть Enter ...')
hold off;
input('Для продовження натисніть Enter ...')
axis auto;
```

8.1.3. Функція ezplot. Синтаксис.

Функція *ezplot* використовується для автоматичної візуалізації функцій, використовуючи вбудовані об'єкти (області, лінії, текст тощо).

ezplot (*f*)

ezplot(*f*, [*min*,*max*])

ezplot(*f*, [*Xmin*, *Xmax*, *Ymin*, *Ymax*])

де *f*(*x*) - функція задана в аналітичному вигляді

ezplot(*x, y*)

ezplot(*x, y,* [*tmin*,*tmax*])

де *x, y* – функції задані в аналітичному вигляді

8.1.4. Функція ezplot. Опис.

ezplot (*f*) відображає вираз *f* = *f*(*x*) за замовчуванням: -2*pi* <*x* <2*pi*.

ezplot (*f*, [*min*, *max*]) графіки *f* = *f*(*x*) над областю аргументу: *min* < *x* < *max*.

Для неявно визначених функцій двох змінних *f* = *f* (*x*, *y*):

ezplot (*f*) відображає графіки *f*(*x*, *y*)=0 над областю аргументу за замовчуванням -2*pi* < *x* < 2*pi*, -2*pi* < *y* < 2*pi*.

ezplot (*f*, [*Xmin*, *Xmax*, *Ymin*, *Ymax*]) графіки *f* (*x*, *y*) = 0 над областю аргументу *xmin* <*x* <*xmax* і *ymin* <*y* <*ymax*.

ezplot (*f*, [*min*, *max*]) графіки *f* (*x*, *y*) = 0 по *min* <*x* <*max* і *min* <*y* <*max*.

Якщо *f* є функцією змінних *u* і *v* (а не *x* і *y*), то кінцеві точки для області аргументу *umin*, *umax*, *vmin* і *vmax* сортуються за алфавітом. Таким чином, ezplot ('*u* ^ 2 - *v* ^ 2 - 1', [- 3,2, -2,3]) відображає u2 - v2 - 1 = 0 над -3 <*u* <2, -2 <*v* <3.

ezplot (*x*, *y*) зображує параметрично визначену планарну криву, яка є параметрично визначеною *x* = *x* (*t*) і *y* = *y* (*t*) для області аргументу 0 <*t* <2.

ezplot (*x*, *y*, [*tmin*, *tmax*]) графіки *x* = *x* (*t*) і *y* = *y* (*t*) по *tmin* <*t* <*tmax*.

8.1.5. Функція ezplot. Приклади застосування

Приклад 8.4. Побудова графіка функції *sin (x)*

```
ezplot ('sin (x)')
```

Приклад 8.5. Побудова графіка функції *sin (x)* з визначеним діапазоном

```
ezplot ('sin (x)', [- 2 * pi 2 * pi])
```

Приклад 8.6. Відображення неявно визначеної функції $x^2 – y^4 = 0$ для області аргументу [-2, 2]:

```
ezplot ('x ^ 2-y ^ 4', [-2, 2])
```

Приклад 8.7. Відображення різних функцій

```
ezplot ('x ^ 2 - y ^ 2')
input('Для продовження натисніть Enter ...')
ezplot ('sin (x)', 'sin (y)', [- 2 * pi 2 * pi])
input('Для продовження натисніть Enter ...')
ezplot ('sin (x)', 'cos (y)', [- 2 * pi 2 * pi])
input('Для продовження натисніть Enter ...')
ezplot('sin(x)/x')
```

Приклад 8.8. Управління вікном відображення

```
close all;
ezplot('sin(x)/x',[-16*pi 16*pi])
axis([-16*pi 16*pi -0.4 1])
```

Приклад 8.9. Функції двох незалежних змінних *X* і *Y* з однаковим діапазоном, на прикладі фігури Ліссажу.

```
ezplot('sin(x)','sin(y)',[-2*pi 2*pi])
input('Для продовження натисніть Enter ...')
ezplot('sin(x)','cos(y)',[-2*pi 2*pi])
input('Для продовження натисніть Enter ...')
ezplot('sin(x)','cos(2*y)',[-2*pi 2*pi])
input('Для продовження натисніть Enter ...')
ezplot('sin(x)','cos(3*y)',[-2*pi 2*pi])
input('Для продовження натисніть Enter ...')
ezplot('sin(x+pi/4)','cos(3*y)',[-2*pi 2*pi])
input('Для продовження натисніть Enter ...')
ezplot('sin(x)','cos(4*y)',[-2*pi 2*pi])
```

Приклад 8.10. Функції двох змінних *A* і *B*, які обчислюються на основі незалежної змінної *t* із заданим діапазоном. За замовчуванням діапазон

визначається як [-2*pi, 2*pi]

```
ezplot('t*cos(t)','t*sin(t)',[0,4*pi])
input('Для продовження натисніть Enter ...')
ezplot('sin(3*t)*cos(t)','sin(3*t)*sin(t)',[0,pi])
```

Приклад 8.11. Функції змінних: *X* (незалежна) і змінної *Y*, яка пов'язана зі значеннями *X* рівнянням *Y*(*X*) = 0

```
ezplot('x^2 - y')
input('Для продовження натисніть Enter ...')
ezplot('x^2 - y',[-1 20])
input('Для продовження натисніть Enter ...')
ezplot('x^2 - y^2 - 1')
input('Для продовження натисніть Enter ...')
ezplot('x^3 + y^3 - 5*x*y + 1/5',[-3,3])
input('Для продовження натисніть Enter ...')
ezplot('1/y-log(y)+log(-1+y)+x - 1')
```

8.1.6. Функція ezpolar. Синтаксис.

Функція *ezpolar* використовується для створення графіків у полярних координатах, де точки визначаються за допомогою кута і відстані від початку координат.

ezpolar (*f*)

ezpolar (*f*, [*a*, *b*])

де *f* - функція, задана у вигляді рядка.

8.1.7. Функція ezpolar. Опис.

ezpolar (*f*) відображає полярну криву *rho = f* (*theta*) для області аргументу (за замовчуванням 0 <*theta* <2*pi*).

ezpolar (*f*, [*a*, *b*]) графіки *f* для *a* <*theta* <*b*.

8.1.8. Функція ezpolar. Приклади застосування

Приклад 8.12. Створення полярного графіка функції *1 + cos(t)* по області [0, 2 * *pi*]:

```
ezpolar ( '1 + cos (t)')
```

Приклад 8.13. Функція, яка відображається вектором кінцевої довжини, що обчислюється літералом і кутом повороту *t* у заданому діапазоні. За замовчуванням діапазон визначається як [0, 2**pi*]

```
ezpolar('sin(t)',[0, 2*pi])
input('Для продовження натисніть Enter ...')
ezpolar('1+cos(t)')
input('Для продовження натисніть Enter ...')
ezpolar('1+cos(t)',[0, pi])
input('Для продовження натисніть Enter ...')
ezpolar('sin(t)+cos(t)',[0, pi])
input('Для продовження натисніть Enter ...')
ezpolar('sin(t)/t', [-6*pi,6*pi])
input('Для продовження натисніть Enter ...')
ezpolar('sin(2*t)*cos(3*t)',[0,pi])
input('Для продовження натисніть Enter ...')
ezpolar('sin(tan(t))')
input('Для продовження натисніть Enter ...')
ezpolar('cos(5*t)')
input('Для продовження натисніть Enter ...')
ezpolar('cos(8*t)')
```

8.2. Індивідуальні завдання

1. Виконайте всі приклади з теоретичної частини лабораторної роботи та додайте отримані графіки у звіт.

2. Використовуючи у складі MATLAB підсистему *help*, самостійно опишіть синтаксис і опис функції *polar*.

3. Порівняйте функції *polar* і *plot*. Виконайте наступні два приклади:

```
clear all;
close all;
x = 0:.01:2*pi;
polar(x,sin(2*x),'--r')
```

```
% Проектування Функцій у полярних координатах
clear all;
close all;
% An M-file script to produce % Comment lines
% "flower petal" plots
theta = -pi:0.01:pi; % Computations
rho(1,:) = 2*sin(5*theta).^2;
rho(2,:) = cos(10*theta).^3;
rho(3,:) = sin(theta).^2;
rho(4,:) = 5*cos(3.5*theta).^3;
for k = 1:4
polar(theta,rho(k,:)) % Graphics output
pause
end;
```

4. Оформити звіт з лабораторної роботи.

Лабораторна робота 9

СИМВОЛЬНІ ОБЧИСЛЕННЯ В ПАКЕТІ MATLAB

Мета лабораторної роботи: отримання та закріплення знань, формування практичних навичок роботи з пакетом MATLAB під час обчислень у символьному вигляді.

9.1. Короткі відомості з теорії

9.1.1. Символьні змінні та функції

В склад MATLAB входить *ToolBox Symbolic Math*, призначений для обчислень у символьному вигляді. Перетворення виразів, визначення аналітичних розв'язків задач лінійної алгебри, диференціального та інтегрального числення, одержання чисельних результатів із будь-якою точністю - далеко не повний перелік можливостей, які надають *ToolBox*.

Символьні змінні та функції є об'єктами класу *sym object*, на відміну від числових змінних, які містяться в масивах *double array*. Символьний об'єкт створюється за допомогою функції *syms*. Команда

```
» syms x a b
```

створює три символьні змінні *x, a, b*. Конструювання символьних функцій від змінних класу *sym object* здійснюється за допомогою звичайних арифметичних операцій і позначень для вбудованих математичних функцій, наприклад:

```
» f=(sin(x)+a)^2*(cos(x)+b)^2/sqrt(abs(a+b))
f=
     (sin(x)+a)^2*(cos(x)+b)^2/abs(a+b)^(1/2)
```

Запис формули в один рядок не завжди зручний, більш природний вигляд дробу дає змогу отримати функція *pretty*:

```
» pretty(f)
                2              2
  (b + cos(x))   (a + sin(x))
  --------------------------
                  1/2
          |a + b|
```

Визначена функція *f* також є символьною, у чому не важко переконатися за допомогою команди *whos*.

Наявні символьні змінні та функції дають змогу утворювати нові символьні вирази:

```
» syms y
» g=(exp(-y)+1)/exp(y)
g =
    (exp(-y)+1)/exp(y)
» h=f*g
h =
    (sin(x)+a)^2*(cos(x)+b)^2/abs(a+b)^(1/2)*(exp(-y)+1)/
exp(y)
» pretty(h)
                2 /   1         \              2
  (b + cos(x))   | ------ + 1 |  (a + sin(x))
                 \ exp(y)      /
  ----------------------------------------
                              1/2
                exp(y)  |a + b|
```

Символьну функцію можна створити без попереднього оголошення змінних за допомогою функції *sym*, вхідним аргументом якої є рядок із виразом, поміщеним в апострофи:

```
» z=sym('c^2/(d+1)')
z =
    c^2/(d+1)
```

```
» pretty(z)
              2
             c
            -----
            d + 1
```

Примітка 1

Функція *sym* може бути використана для оголошення символьних змінних. Команда *syms a b c* еквівалентна послідовності команд *a* = *sym*('*a*'), *b* = *sym*('*b*'), *c* = *sym*('*c*'):

```
» syms a b c
» whos a b c
Name   Size    Bytes Class
 a     1x1     126 sym object
 b     1x1     126 sym object
 c     1x1     126 sym object
Grand total is 3 elements using 378 bytes
» a1=sym('a1'), b1=sym('b1'), c1=sym('c1')
a1 =
    a1
b1 =
    b1
c1 =
    c1
» whos a1 b1 c1
 Name   Size     Bytes Class
 a1     1x1     128 sym object
 b1     1x1     128 sym object
 c1     1x1     128 sym object
Grand total is 3 elements using 384 bytes
```

Під час роботи в області комплексних чисел слід зазначити, що визначувані змінні є в загальному випадку комплексними. Комплексні символьні змінні задаються командою *syms* з опцією *unreal*. Опція *real* означає, що змінні трактуються як дійсні.

Розглянемо приклад, де результат символьних обчислень залежить від того, які символьні змінні використовуються – дійсні чи комплексні. Оголосимо дві

дійсні змінні *a* і *b* та утворимо комплексне число, вважаючи, що *a* є дійсною частиною, а *b* – уявною, потім знайдемо спряжене до нього число за допомогою *conj*:

```
» syms a b real
» p=conj(a+i*b)
p = a-i*b
```

Зробимо аналогічні дії, попередньо оголосивши *a* і *b* комплексними символьними змінними

```
» syms a b unreal
» q=conj(a+i*b)
q = conj(a+i*b)
```

Таким чином, у загальному випадку $p \neq q$.

Символьні змінні можуть бути елементами векторів і матриць. Елементи рядків матриць при введенні відокремлюються пробілами або комами, а стовпців – крапкою з комою, так само як і при введенні звичайних матриць. У результаті утворюються символьні матриці та вектори, до яких застосовні матричні та поелементні операції, а також вбудовані функції. Розглянемо приклад введення і множення двох символьних матриць:

```
» syms a b c d e f g h
» A=[a b;c d]
A =
    [ a, b]
    [ c, d]
» B=[e f; g h]
B =
    [ e, f]
    [ g, h]
» C=A*B
C =
    [ a*e+b*g, a*f+b*h]
    [ c*e+d*g, c*f+d*h]
```

Функція *sym* дозволяє перетворювати значення числових змінних на символьні. Введіть числову матрицю *A* і перетворіть її на символічну матрицю *B*:

```
» A=[1.3 -2.1 4.9; 6.9 3.7 8.5]
A =
    1.3000  -2.1000  4.9000
    6.9000  3.7000  8.5000
» B=sym(A)
B =
    [ 13/10, -21/10, 49/10]
    [ 69/10, 37/10,  17/2]
```

При переході від числових виразів до символьних використовується запис числа у вигляді раціонального дробу. Використання раціональних дробів під час виконання символьних обчислень дає змогу завжди отримувати точний результат, який не містить похибки округлення. Переконатися в цьому можна на наступному простому прикладі. Встановіть формат *long* для відображення максимально можливого числа значущих цифр для значень числових змінних і знайдіть суму чисел 10^{10} и 10^{-10}

```
» format long
» 1.0e10+1.0e-10
ans =
    1.000000000000000e+010
```

Тепер перетворіть числа в символьні змінні і знову обчисліть суму:

```
» large=sym(1.0e10);
» small=sym(1.0e-10);
» s=large+small
s = 10000000000000000001/10000000000
```

Раціональний дріб є точним значенням суми. Зрозуміло, що символьні обчислення потребують більших часових витрат порівняно зі звичайними.

Обчислення символьних виразів проводиться за допомогою функції *vpa*, наприклад

```
» c=sym('sqrt(2)');
» cn=vpa(c)
cn =
    1.41421356237309504880168872420969
```

За замовчуванням зберігається тридцять дві значущі цифри. Другий вхідний параметр функції *vpa* дає змогу задавати обчислення із заданим числом розрядів, наприклад, із 45 розрядами

```
» cn=vpa(c,45)
cn = 1.41421356237309504880168872420969807856967187
```

Вихідне значення функції *vpa* є символьною змінною, але його можна використовувати у звичайних обчисленнях, наприклад

```
» cn=vpa(c,45)
cn =
    1.41421356237309504880168872420969807856967187
» cn+2
ans =
    3.41421356237309504880168872420969807856967187
» cn*2
ans =
    2.82842712474619009760337744841939615713934375
```

Результат арифметичних операцій у цих випадках виходить у символьних змінних. Для переведення символьних змінних у числові, тобто змінні типу *double array*, використовується функція *double*:

```
» cnd=double(cn)
cnd = 1.41421356237310
```

Візуалізація символьних функцій однієї змінної здійснюється за допомогою *ezplot*. Найпростіший варіант використання *ezplot* полягає у зазначенні символьної функції як єдиного вхідного аргументу, при цьому в графічне вікно виводиться графік функції на відрізку $[-2\pi, 2\pi]$. Наприклад:

```
» f=sym('x*sin(x^2)^3');
» ezplot(f)
```

Зверніть увагу (рис. 9.1), що автоматично створюється відповідний заголовок.

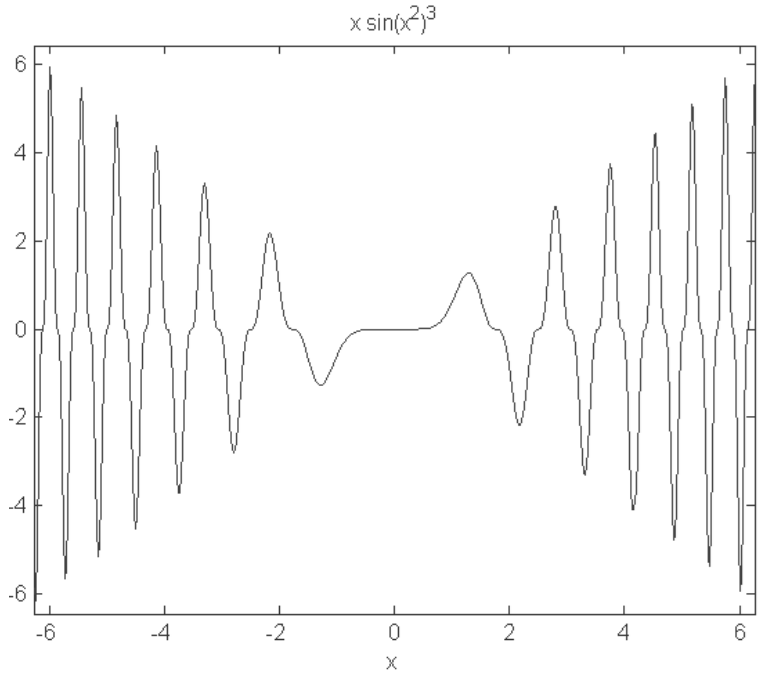


Рис. 9.1. Графік символьної функції

Другим аргументом може бути заданий вектор із межами відрізка, на якому потрібно побудувати графік функції:

```
» ezplot(f,[-9 7])
```

Функція *ezplot* має деякі відмінності від свого аналога – функції *fplot*, що застосовується до числових функцій. Зокрема, можлива вказівка символьної функції, що залежить від двох аргументів:

```
» z=sym('x^2+a^3');
» ezplot(z,[-2 1 -3 4])
```

Межі зміни аргументів визначаються їхніми назвами. Перші два числа відповідають першому за алфавітом аргументу, а останні – другому. У

розглянутому прикладі *a* змінюється від –2 до 1, а *x* змінюється від –3 до 4. У графічне вікно виводиться лінія, що задовольняє вираз `x^2+a^3=0`.

ToolBox Symbolic Math надає користувачеві цілий набір засобів для візуалізації символьних функцій.

9.1.2. Спрощення та перетворення виразів

Складні алгебраїчні та тригонометричні вирази часто можуть бути приведені до еквівалентних шляхом спрощення. *ToolBox Symbolic Math* має цілу низку сервісних функцій, призначених для різних перетворень символьних виразів. Операції з поліномами реалізують чотири функції: *collect, expand, factor, horner*.

Обчислення коефіцієнтів при степенях незалежної змінної проводиться за допомогою функції *collect*. Введіть поліном і відобразіть його в командному вікні за допомогою *pretty*:

```
» p=sym('(x+a)^4+(x-1)^3-(x-a)^2-a*x+x-3');
» pretty(p)
             3                  4           2
  x + (x - 1)  - a x + (a + x)  - (a - x)  - 3
```

Потім застосуйте до полінома функцію *collect*:

```
» pc=collect(p);
» pretty(pc)
   4              3        2        2
  x  + (4 a + 1) x  + (6 a  - 4) x  +
      3               4    2
+ (4 a  + a + 4) x + a  - a  - 4
```

За замовчуванням як незалежну змінну в поліномі обирають *x*, однак можна вважати, що *a* – незалежна змінна, а *x* входить у коефіцієнти полінома, який залежить від *a*. Другий аргумент функції *collect* призначений для вказівки змінної, при ступенях якої слід знайти коефіцієнти:

```
» pca=collect(p,'a');
» pretty(pca)
```

```
   4          3       2       2       3
  a  + (4 x) a  + (6 x  - 1) a  + (4 x  + x) a + x +
            3    2    4
+ (x - 1)  - x  + x  - 3
```

Функція *expand* являє поліном сумою одночленів:

```
» pe=expand(p);
» pretty(pe)
   4      3        2  2    2        3
  a  + 4 a  x + 6 a  x  - a  + 4 a x  +
          4    3      2
+ a x + x  + x  - 4 x  + 4 x - 4
```

Аргументом функції *expand* може бути не тільки поліном, а й символьний вираз, що містить тригонометричні, експоненціальні та логарифмічні функції:

```
» pk=sym('(sin(x)+cos(x))^3+(sin(x)+cos(x))^2+
(sin(x)+cos(x))')
pk =
     (sin(x)+cos(x))^3+(sin(x)+cos(x))^2+(sin(x)+cos(x))
» pk2=expand(pk);
» pretty(pk2)
      3           2                2                 2
cos(x)  + 3 cos(x)  sin(x) + cos(x)  + 3 cos(x) sin(x) +
                                   3         2
+ 2 cos(x) sin(x) + cos(x) + sin(x)  + sin(x) + sin(x)
```

Аргументами функцій *expand, collect* може бути не тільки окремо поліном або функція (тригонометрична, експоненціальна, логарифмічна), а й їхні поєднання, наприклад:

```
» p=sym('(x+a)^4+(x-1)^3+(sin(x)+cos(x))^4');
» pretty(p)
         3                      4           4
  (x - 1)  + (cos(x) + sin(x))  + (a + x)
» p1=collect(p)
```

```
p1 =
     x^4+(1+4*a)*x^3+(-3+6*a^2)*x^2+(3+4*a^3)*x+a^4-
-1+(sin(x)+cos(x))^4
» p2=expand(p1)
p2 =
     x^4+x^3+4*x^3*a-3*x^2+6*x^2*a^2+3*x+4*x*a^3+a^4-
1+sin(x)^4+4*sin(x)^3*cos(x)+6*sin(x)^2*cos(x)^2+4*sin(x)*c
os(x)^3+cos(x)^4
» pretty(p2)
   4      3        2  2        3    4    3      2
  a  + 4 a  x + 6 a  x  + 4 a x  + x  + x  - 3 x  + 3 x +
        4           3                  2       2
 + cos(x)  + 4 cos(x)  sin(x) + 6 cos(x)  sin(x)  + 4*
               3         4
 * cos(x) sin(x)  + sin(x)  - 1
```

Символьні поліноми розкладаються на множники функцією *factor*, якщо одержувані множники мають раціональні коефіцієнти:

```
» p=sym('7*x^5-56*x^4+105*x^3+140*x^2-532*x+336');
» p1=factor(p)
p1 =
     7*(x-2)*(x-3)*(x-4)*(x+2)*(x-1)
```

Функція *factor* може також представляти числа у вигляді добутку простих чисел

```
» syms a
» a=sym('2738470');
» a1=factor(a)
a1 = (2)*(5)*(7)*(19)*(29)*(71)
```

Зверніть увагу, що звернення

```
» a2=factor(2738470)
a2 = 2   5   7  19  29  71
```

виводить у командне вікно аналогічний результат, однак змінна *a*1 є символьною, а змінна *a*2 – дійсною, у чому нескладно переконатися за допомогою *whos*:

```
» whos a1 a2
   Name  Size    Bytes Class
    a1   1x1     176 sym object
    a2   1x6     48 double array
Grand total is 33 elements using 224 bytes
```

MATLAB є об'єктно-орієнтованим середовищем, числові змінні *double array* утворюють клас зі своєю функцією *factor*, а функція *factor* для символьних змінних реалізована в інший файл-функції. MATLAB визначає за типом аргументу відповідний клас, а потім і необхідну функцію.

Функція *horner* дає змогу представити поліном за схемою Горнера (*Horner polynomial representation*):

```
» horner(p1)
ans =
    336 +(-532+(140+(105+(-56+7*x)*x)*x)*x)*x
```

Спрощення виразів загального вигляду здійснюється за допомогою функцій *simple*, *simplify*, які засновані на різних підходах. Функція *simplify* реалізує потужний алгоритм спрощення виразів, які містять як тригонометричні, експоненціальну та логарифмічну функції, так і спеціальні функції. Крім того, функція *simplify* здатна перетворювати вирази, що містять символьне зведення до степеня, підсумовування та інтегрування. Алгоритм, закладений у *simple*, намагається отримати вираз, що представляється меншим числом символів, ніж вихідний, послідовно застосовуючи всі функції спрощення *ToolBox*. При спрощенні складних виразів бажано застосовувати обидві функції, оскільки будь-яка з них може дати кращий результат, ніж інша. Розглянемо такий приклад

```
» v=sym('(((2+sqrt(3))/(sqrt(2)+sqrt(2+sqrt(3))))+
+(2-sqrt(3))/(sqrt(2)-sqrt(2-sqrt(3))))^2')
v =
((3^(1/2) - 2)/(2^(1/2) - (2 - 3^(1/2))^(1/2)) -
-(3^(1/2) + 2)/((3^(1/2) + 2)^(1/2) + 2^(1/2)))^2
```

```
» pretty(v)
  /        1/2                    1/2           \2
  |       3    - 2               3    + 2       |
  | ------------------- - ------------------- |
  |  1/2         1/2 1/2     1/2     1/2    1/2 |
  \ 2    - (2 - 3   )       (3    + 2)    + 2    /
» v2=simplify(v)
» pretty(v2)
  /        1/2                    1/2           \2
  |       3    - 2               3    + 2       |
  | ------------------- - ------------------- |
  |  1/2         1/2 1/2     1/2     1/2    1/2 |
  \ 2    - (2 - 3   )       (3    + 2)    + 2    /
» v1=simple(v)
v1 = 2
```

Таким чином, у цьому прикладі найкращий результат отримано за допомогою функції *simple*.

Функція *subs* дозволяє здійснити підстановку одного виразу в інший. У загальному випадку функція викликається з трьома вхідними аргументами: ім'ям символічної функції, змінною, що підлягає заміні, і виразом, який слід підставити у вираз. Функція *subs*, зокрема, полегшує введення громіздких символьних виразів. Наприклад:

```
» f=sym('((N^2-x^2)/(N+x)^2)+(sin(2*x)/A)*(sqrt(B*C))
+(A/B)^2+(cos(x)/C)^2')

» pretty(f)
   2    2          2    2                  1/2
  N  - x     cos(x)    A    sin(2 x) (B C)
  -------- + ------- + -- + ----------------
         2       2       2           A
  (N + x)       C       B

» f=subs(f,'A','sin(x)');
» f=subs(f,'B','tan(x)');
» f=subs(f,'C','cot(x)');
```

```
» pretty(f)
   2    2          2         2
1/2
  N  - x     cos(x)    sin(x)    sin(2 x) (cot(x) tan(x))
  -------- + ------- + ------- + ------------------------
         2         2         2              sin(x)
  (N + x)    cot(x)    tan(x)
```

Спростимо отриманий вираз за допомогою функції *simple*:

```
» f2=simple(f)
f2 =
    (2*sin(x)*N+sin(2*x)*N+sin(2*x)*x)/(N+x)/sin(x)
» pretty(f2)
      2 sin(x) N + sin(2 x) N + sin(2 x) x
      -----------------------------------
           (N + x) sin(x)
```

Підстановка замість змінної її числового значення призводить до обчислення символьної функції від значення аргументу, наприклад

```
» f=sym('((N^2-x^2)/(N+x)^2)+(sin(2*x)/A)*(sqrt(B*C))
+(A/B)^2+(cos(x)/C)^2')
» pretty(f)
   2    2          2    2                  1/2
  N  - x     cos(x)    A    sin(2 x) (B C)
  -------- + ------- + -- + -----------------
         2       2        2           A
  (N + x)       C        B
» q=subs(f,'x',0)
q =
    1+A^2/B^2+1/C^2
» f3=sym('sin(x)+exp(x)+tan(x)');
» q1=subs(f3,'x',0)
q1 =
    1
```

9.1.3. Розкладання в ряд Тейлора та визначення символьних виразів для сум

Розкладання математичних функцій у ряд Тейлора дає змогу здійснювати функція *taylor*, наприклад:

```
» f=sym('sin(x)');
» tf=taylor(f);
» pretty(tf)
   5     3
  x     x
  --- - -- + x
  120   6
```

За замовчуванням виводиться шість членів ряду розкладання в околиці точки нуль. Число членів розкладання можна задати в другому додатковому параметрі функції *taylor*. Третій параметр вказує, за якою зі змінних слід проводити розкладання в тому випадку, коли символьна функція визначена від декількох змінних. Точка, в околиці якої проводиться розкладання, вказується в четвертому вхідному аргументі функції *taylor*, наприклад

```
» syms x;
» f=sym('sin(x)');
» tf=taylor(f, 5, x, pi/4);
» pretty(tf)
   1/2 / pi     \3    1/2 / pi     \2    1/2 / pi     \4
  2    | -- - x |    2    | -- - x |    2    | -- - x |
       \ 4      /         \ 4      /         \ 4      /
2           \ 4      /
  ---------------- - ---------------- + ---------------- +
         12                  4                  48

            1/2 / pi     \
    1/2   2    | -- - x |
  2           \ 4      /
+ ---- - --------------
  2           2
```

Знаходження символьних виразів для сум, зокрема й нескінченних, дозволяє здійснити функція *symsum*. Звернення до *symsum* у загальному випадку передбачає завдання чотирьох аргументів: доданка в символьній формі, що залежить від індексу, самого індексу, а також верхньої і нижньої межі суми. Якщо до доданків

входить факторіал, то слід застосувати до виразу функцію *sym*. Знайдіть значення нескінченної суми, що є розкладанням у ряд функції *sin(x)*

$$s = \sum_{k=0}^{\infty} (-1)^k \frac{x^{2k+1}}{(2k+1)!}$$

```
» syms k x
» s=symsum((-1)^(k)*x^(2*k+1)/sym('(2*k+1)!'),
k,0,inf)
s =
     sin(x)
```

9.1.4. Визначення меж, диференціювання та інтегрування

Функція *limit* знаходить межу функції в деякій точці, включно з плюс або мінус нескінченність. Першим вхідним аргументом *limit* є символьний вираз, другим – змінна, а третім – точка, в якій визначається межа. Нехай, наприклад, потрібно обчислити

$$\lim_{x\to\infty} (1+\frac{1}{x})^{ax}$$

```
» syms a x
» limit((1+1/x)^(x*a),x,Inf)
ans = exp(a)
```

Функція *limit* дає змогу знаходити односторонні межі. Для знаходження межі праворуч слід вказати четвертий додатковий аргумент *'right'*, а ліворуч – *'left'*. Знайдіть розв'язок наступних двох задач

$$\lim_{x\to 0+} (10+x)^{1/x}; \quad \lim_{x\to 0-} (10+x)^{1/x}.$$

```
» syms x
» limit((10+x)^(1/x),x,0, 'left')
ans = 0
» limit((10+x)^(1/x),x,0, 'right')
ans = inf
```

Зверніть увагу, що звичайна межа в точці нуль не існує:

```
» limit((10+x)^(1/x),x,0)
ans = NaN
```

Визначення похідної через межу дає змогу застосовувати *limit* для диференціювання функцій. Наприклад, знайдемо першу похідну функції *arctg(x)*, використовуючи рівність

$$\frac{d}{dx}\text{arctg}(x) = \lim_{h\to 0}\frac{\text{arctg}(x+h)-\text{arctg}(x)}{h}$$

```
» syms h x
» L=limit((atan(x+h)-atan(x))/h, h, 0);
» pretty(L)
       1
    ------
         2
    1 + x
```

Обчислення похідних будь-якого порядку простіше проводити за допомогою функції *diff*. Символьний запис функції вказується в першому вхідному аргументі, змінна, за якою проводиться диференціювання – у другому, а порядок похідної – у третьому. Застосуємо *diff* для обчислення першої та другої похідних функції $arctg(x)$:

```
» P=diff('atan(x)',x,1);
» pretty(P)
    1
  ------
   2
  x  + 1
» P=diff('atan(x)',x,2);
» pretty(P)
      2 x
  - ---------
      2     2
    (x  + 1)
```

Символьне інтегрування є значно складнішим завданням, ніж диференціювання. *ToolBox Symbolic Math* дає змогу працювати як з невизначеними інтегралами, так і з визначеними. Невизначені інтеграли від символьних функцій обчислюються за допомогою функції *int*. Як вхідні аргументи вказуються символьна функція і змінна, за якою проводиться інтегрування, наприклад, нехай необхідно обчислити невизначений інтеграл $f = \int \exp(2x)dx$, тоді отримаємо:

```
» syms x
» f=sym('exp(2*x)');
» I=int(f,x)
» pretty(I)
    1/2 exp(2 x)
```

Зрозуміло, що функція *int* не дає змоги отримати невизначений інтеграл від довільної функції.

Для знаходження визначеного інтеграла в символьному вигляді слід задати нижню і верхню межі інтегрування, відповідно, у третьому і четвертому аргументах *int*:

```
» syms x a b
» f=sym('exp(2*x)');
» I=int(f,x,a,b);
» pretty(I)
    1/2 exp(2 b) - 1/2 exp(2 a)
```

Подвійні інтеграли обчислюються дворазовим застосуванням функції int. Нехай, наприклад, необхідно обчислити інтеграл $\int_c^d \int_a^b y\sin(x)dxdy$, тоді для його визначення необхідно задати символьні змінні *a, b, c, d, x, y*, підінтегральну функцію *f* від *x* та *y* і проінтегрувати спочатку за однією змінною, а потім за іншою

```
» syms a b c d x y
» f=sym('y*sin(x)');
```

```
» Ix=int(f,x,a,b)
Ix =
    -y*cos(b)+y*cos(a)
» Iy=int(Ix,y,c,d)
Iy =
    1/2*(-cos(b)+cos(a))*(d^2-c^2)
» pretty(Iy)
      2    2
    (c  - d ) (cos(a) - cos(b))
  - --------------------------
                 2
```

Аналогічним чином обчислюються будь-які кратні інтеграли в символьному вигляді.

9.2. Індивідуальне завдання

1. Задайте за допомогою символьних змінних дві символьні матриці *A* і *B*, розмірами 3×3. Визначте добуток матриць *A* і *B*.

2. Задайте числову матрицю *D*, розмірами 3×3, що містить у якості елементів числа *N*, *N*+1, *N*–1, 0.9*N*, 2.4, 3.5 і т.д., де *N* – ваш номер за списком у журналі групи. Отримайте з числової матриці символьну.

3. За допомогою символьних обчислень визначте суму чисел $10^{10}+10^{-10}+10^{N}+10^{-N}$, де *N* – ваш номер за списком у журналі групи.

4. За допомогою символьних обчислень визначте корінь квадратний із числа *N.N* із сорока двома значущими цифрами.

5. Побудуйте графіки символьних функцій:

$$f = N^2\sin(Nx),\ -2\pi < x < 2\pi;$$
$$f_1 = (x-N)^2 + (y+N)^3,\ -30 \le x \le 30;\ -30 \le y \le 30,$$

де *N* – Ваш номер за списком у журналі групи.

6. Поліном

$$y = (x+N)^5 - (x+2N)^3 + (x+N)^2 + x - N, \qquad (9.1)$$

де N – ваш номер за списком у журналі групи, за допомогою команди *pretty* відобразіть у командному вікні, потім перетворіть його до вигляду, що містить ступені x із відповідними коефіцієнтами.

7. Визначте коефіцієнти полінома (9.1) при змінній N.

8. Подайте поліном (9.1) у вигляді суми одночленів.

9. Розкладіть поліном (9.1) на множники.

10. Застосуйте до полінома (9.1) функцію *horner*.

11. Представте число 1000 N+2 N у вигляді добутку простих чисел.

12. Спростіть вираз

$$y = \sin(a)\sin(b-c)\cos(b+c-a) + \sin(b)\sin(c-a)\cos(c+a-b) + \\ + \sin(c)\sin(a-b)\cos(a+b-c).$$

13. Отримайте сім членів ряду Тейлора в околиці точки нуль для функції $y = N(\sin(x) + \cos(x))$.

14. Визначте суму членів ряду $s = \sum_{k=0}^{\infty} N(-1)^k \frac{x^{2k}}{(2k)!}$.

15. Визначте межу $\lim_{x\to 0} N\frac{\sin(x)}{x}$, де N – ваш номер за списком у журналі групи.

16. Визначте межі $\lim_{x\to 0+} \operatorname{arctg}(\frac{1}{x})$ і $\lim_{x\to 0-} \operatorname{arctg}(\frac{1}{x})$.

17. Визначте перші три похідні від функції $y = N(\sin(x))^2 + N^2x^3 + e^{Nx}$, де N – ваш номер за списком у журналі групи.

18. Обчисліть інтеграл $\int_0^b \int_0^a (\frac{x^2}{2p} + \frac{y^2}{2q})dxdy$.

19. Оформіть звіт з лабораторної роботи.

Лабораторна робота 10

ДИСКРЕТНА АПРОКСИМАЦІЯ ТАБЛИЧНО ЗАДАНОЇ ФУНКЦІЇ В ПАКЕТІ MATLAB

Мета лабораторної роботи: отримання та закріплення знань, формування практичних навичок роботи з пакетом MATLAB під час апроксимації таблично заданої функції.

10.1. Короткі відомості з теорії

Нехай деяку функцію $F(x)$, представлену табличними значеннями, на аргументах (X_0, ... , X_m, ... , X_M) в області (Xb, Xe), необхідно якнайкраще (тобто якнайточніше і, за можливості, якнайкомпактніше) представити в аналітичному вигляді (рис. 10.1).

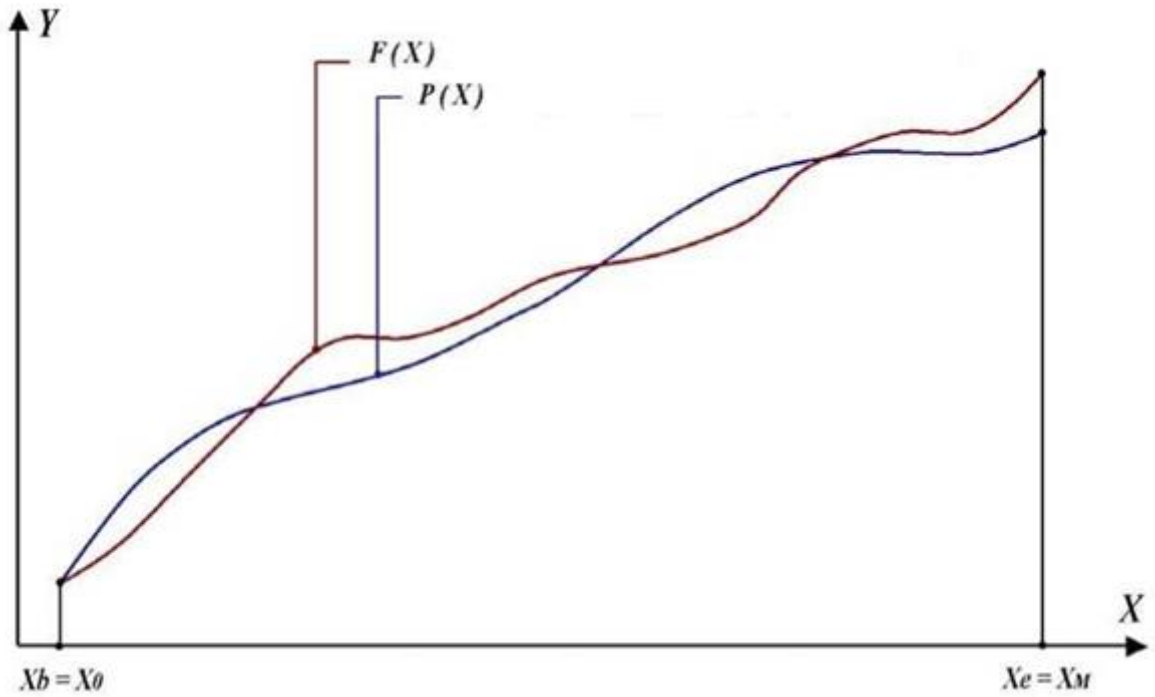


Рис. 10.1. Постановка задачі апроксимації таблично заданої функції.

Така задача отримала назву – "задача апроксимації таблично заданої функції". Цю задачу, як правило, розв'язують за допомогою апроксимувальної функції $P(x)$ шляхом визначення її коефіцієнтів (a_0 , ... , a_i , ... , a_n) таким чином, щоб досягти мінімальної відмінності між вихідною таблично заданою функцією $F(x)$ та її аналітичним еквівалентом, тобто:

$$F(x) \cong P(x) = \sum_{n=0}^{N} a_n v_n(x),$$

де $v_n(x)$ – є однією із заздалегідь обраних аналітичних функцій, які називають базисними функціями; a_n – є шуканим коефіцієнтом при цій функції.

Як ви вже здогадалися, для того, щоб знайти розв'язок поставленої задачі, як мінімум, необхідно математично визначити умови, за яких досягається рівність:

$$F(x) \cong P(x).$$

Слід також підкреслити, що успіх розв'язання задачі апроксимації залежить також від того, чи будуть такі умови конструктивними з погляду досягнення поставленої мети.

10.1.1. Завдання апроксимації в дискретній формі

Класичним прикладом умов, які дають змогу розв'язати задачу апроксимації, є умови, сформульовані в методі найменших квадратів.

10.1.1.1. Побудова умов для розв'язання задачі апроксимації в дискретній формі.

Для початку, виходячи з припущення про існування аналітичного вигляду функцій, графічно представимо функцію $F(x)$, а також деяку апроксимуючу функцію – $P(x)$, яка поки що не цілком задовольняє розв'язанню нашої задачі.

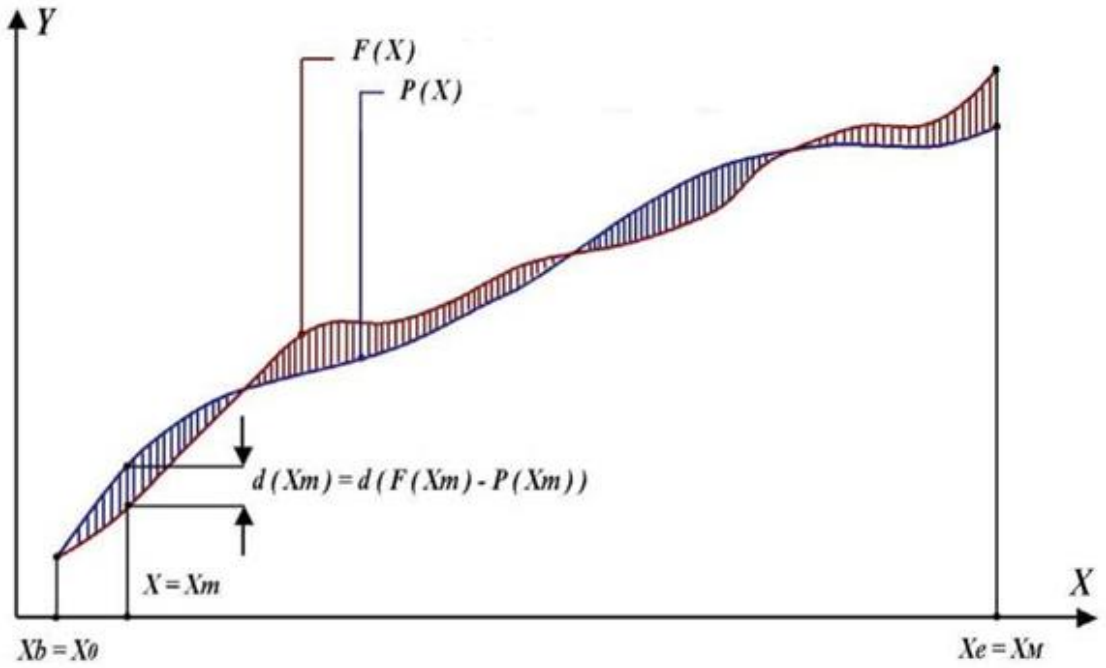


Рис.10.2. Відхилення між апроксимованою та апроксимованою функціями.

Крок 1. Дискретні відхилення та їх сума

Висновки, які безпосередньо доступні з графічного представлення функцій $F(x)$ і $P(x)$, можна сформулювати так:

1. Якби нам одразу вдалося досягти рівності функцій, то для кожного значення незалежної змінної x, відхилення d між відповідними значеннями цих функцій було б нульовим.

2. Сума всіх відхилень буде тим ближчою до нуля, чим ближчим до нуля буде кожне зі складових суми відхилень. Однак, дане твердження буде справедливим тільки в тому випадку, якщо всі відхилення d ми будемо розглядати без урахування їхнього знака. Якщо сума всіх відхилень (без урахування їхнього знака) прагне до нуля,

$$\sum_{m=0}^{M} d(x_m) \to 0,$$

то до нуля прагнутимуть усі відхилення, що входять до неї. Таким чином, якщо ми

побудуємо формальний метод, що обчислює коефіцієнти (a_0, ... , ai, ... , $aк$), при цьому значення цих коефіцієнтів дасть змогу якнайкраще наблизити таку суму до нульового значення, то це буде рівнозначно розв'язанню задачі апроксимації:

$$F(x) \cong P(x).$$

Крок 2. Відхилення у формі модуля

Для початку, визначимо відхилення $d(x)$ між функціями $F(x)$ і $P(x)$ як просту різницю між цими функціями:

$$d(x) = F(x) - P(x).$$

На жаль, прості різниці, отримані за різних значень $\boldsymbol{x}$, можуть мати як позитивні, так і негативні значення. Підсумовуючи всі позитивні та від'ємні відхилення між функціями $F(x)$ і $P(x)$, цілком можливо отримати нульовий результат за очевидної нерівності цих функцій. Для цього достатньо, щоб сума всіх позитивних відхилень випадковим чином дорівнювала сумі всіх негативних відхилень.

На перший погляд, цей недолік легко усунути – достатньо різницю між функціями записати у формі модуля:

$$d_{\mathrm{mod}}(x_m) = |F(x_m) - P(x_m)|.$$

Однак застосування модуля вносить у такий вираз особливі точки, тобто точки, в яких похідні, узяті зліва і справа від точки, не дорівнюють між собою. Поява таких особливих точок (а їхня кількість явно непередбачувана) одразу вносить суттєві складнощі в методи пошуку мінімумів, які нам далі доведеться використовувати.

Крок 3. Відхилення у квадратичній формі

Проблема особливих точок, виявлена нами на кроці 2, ставить перед нами просте запитання - що для нас важливіше, зручна і надійна оцінка рівності функцій $F(x)$ і $P(x)$ чи прив'язка конструйованої оцінки до відхилення у вигляді простої

різниці між цими функціями? Відповідь очевидна – важливо отримати однозначну, усюди позитивну, безперервну функцію оцінки, позбавлену особливих точок. Найпростіший спосіб отримати таку оціночну функцію, це представити відхилення як квадрат різниці функцій $F(x)$ і $P(x)$:

$$d_{sq}(x_m) = \big(F(x_m) - P(x_m)\big)^2.$$

У цьому випадку, сума всіх відхилень визначена як:

$$\sum_{m=0}^{M} d_{sq}(x_m) \to 0,$$

матиме в своєму розпорядженні весь, необхідний для нас, набір властивостей:

1. Сума буде гарантовано позитивна й обмежена знизу нульовим значенням, яке є індикатором рівності функцій $F(x)$ і $P(x)$
2. Операції, які використовувалися під час конструювання суми, не вноситимуть особливих точок, що перешкоджають застосуванню методів мінімізації.

Крок 4. Формальна умова для розв'язання задачі апроксимації

На підставі результатів кроку 3, остаточний вираз для формальної оцінки успішності розв'язування задачі апроксимації можна записати таким чином:

$$D = \sum_{m=0}^{M} \big(F(x_m) - P(x_m)\big)^2 \to 0$$

або

$$D = \sum_{m=0}^{M} \left(F(x_m) - \sum_{n=0}^{N} v_m(x_n)\right)^2 \to 0.$$

10.1.1.2. Розв'язання задачі апроксимації в дискретній формі

У рамках лабораторної роботи ми опустимо всі кроки, які послідовно приводять нас до побудови наступної системи лінійних рівнянь відносно невідомих a_n, розв'язання якої дає змогу знайти апроксимуючу функцію $P(x)$:

$$\begin{cases} \sum_{n=0}^{N} a_n C_{0,n} = B_0; \\ \dots \\ \sum_{n=0}^{N} a_n C_{N,n} = B_N, \end{cases}$$

де

$$B_k = \sum_{m=0}^{M} \big(F(x_m) \cdot v_k(x_m)\big);$$

$$C_{k,n} = \sum_{m=0}^{M} \big(v_k(x_m) \cdot v_n(x_m)\big).$$

Легко помітити, що в ці суми входять функції, які, або дані нам як табличні значення для всіх своїх аргументів x, або, з огляду на відомий аналітичний вигляд функцій $v(x)$, можуть бути обчислені як значення для такого ж набору аргументів x. Таким чином, розглянуті суми B_k і $C_{k,n}$, можна розглядати як константи, значення яких залежать тільки від індексів k і n.

10.2. Реалізації методів найменших квадратів

10.2.1. Реалізація методу найменших квадратів для базисних функцій x^n

```
% Least Squares_Method
% МЕТОД НАЙМЕНШИХ КВАДРАТІВ ДЛЯ БАЗИСНИХ ФУНКЦІЙ x^n
% Синтаксис виклику:
% [A]=LSM01(F,nV)
% F - Вихідна матриця таблична функція вигляду:
```

```
% x1 x2 x3 x4 .... xM
% y1 y2 y3 y4 .... yM
% nV - Довжина степеневого ряду, що апроксимує
% Наприклад (підготовка тестових даних):
% x=[0:pi/100:2*pi];
% y=eval('x+sin(x)-log(x+1)');
% F=[x;y];
% Виклик методу:
% [A]=LSM01(F,10)
% Результат:
% Максимальна абсолютна похибка: 0.00028535
% A =
% -0.0003
% 1.0068
% 0.4619
% -0.4028
% 0.1001
% -0.0232
% 0.0070
% -0.0013
% 0.0001
% -0.0000
function [A]=LSM01(F,nV)
if isempty(F)
disp('Матриця [F] не задана');
A=NaN;
return
end;
% Мінімальний вхідний контроль
if (size(F,1) ~= 2)
disp('Розміри матриці [F] задані неправильно');
A=NaN;
return
end;
if (size(F,2) < nV)
disp(' Недостатньо точок у [F] для визначення коефіцієнтів
для [V]');
A=NaN;
return
end;
% Підготовка робочих матриць
C=zeros(nV,nV);
```

```
B=zeros(nV,1);
% Побудуємо систему лінійних рівнянь для апроксимації
% Для всіх рядків матриці 'C'
for row=1:nV
% Для всіх 'x' матриці 'F'
B(row,1)=0;
for k=1:size(F,2)
x=F(1,k);
B(row,1)=B(row,1)+ (F(1,k)^(row-1))*F(2,k);
end;
% Для всіх колонок матриці 'C'
for col=1:nV
% Для всіх 'x' матриці 'F'
C(row,col)=0;
for k=1:size(F,2)
x=F(1,k);
C(row,col)=C(row,col)+(F(1,k)^(row-1))*(F(1,k)^(col-1));
end; end; end;
% Обчислимо коефіцієнти для апроксимуючого виразу
ab=cat(2,C,B);
A = fgauss01(ab);
% Нижче наведено варіанти, які іноді видають попередження
% A=inv(C)*B; % A=C\B;
% Обчислимо значення функції за апроксимуючим виразом
% Для всіх 'x' матриці 'F'
for k=1:size(F,2)
x=F(1,k);
y(1,k)=0;
% Для всіх рядків матриці 'V'
for row=1:nV
y(1,k)=y(1,k)+A(row,1)*(F(1,k)^(row-1));
end; end;
% Побудуємо графік вихідної табличної функції
% plot(F(1,:),F(2,:));
% Побудуємо графік за результатами апроксимації
% plot(F(1,:),y);
% Побудуємо графік абсолютної похибки
delta=F(2,:)-y;
maxdelta=max(abs(delta));
disp(strcat('  Максимальна   абсолютна   похибка   :   ',
num2str(maxdelta)));
plot(F(1,:),delta);
```

10.2.2. Тест методу найменших квадратів для базисних функцій x^n

```
% SCRIPT Test01_LSM
disp('----------------------------------------');
disp('ПІДГОТОВКА ТЕСТОВОГО ПОЛІНОМА ');
x=[0:0.1:2];
c=[0.5 2 -1 0.3 1.7]
disp(' Коефіцієнти перераховуються від старших ступенів до
молодших ');
y=polyval(c,x);
plot(x,y);
replay=input('Для продовження натисніть Enter...');
disp('----------------------------------------');
disp('АПРОКСИМАЦІЯ СТЕПЕНЕВИМ РЯДОМ');
F=[x;y];
[A1]=LSM01(F,5);
A1'
disp('Коефіцієнти перераховуються від молодших ступенів до
старших');
replay=input('Для продовження натисніть...');
disp('----------------------------------------');
disp('НАКЛАДЕННЯ ШУМУ З МАКСИМАЛЬНОЮ АМПЛІТУДОЮ');
y=polyval(c,x);
ns=rand(1,length(x));
maxns=max(abs(ns))
y=y+ns;
F=[x;y];
disp('АПРОКСИМАЦІЯ СТЕПЕНЕВИМ РЯДОМ');
[A2]=LSM01(F,5);
A2'
disp(' Коефіцієнти перераховуються від молодших ступенів до
старших ');
```

10.2.3. Реалізація методу найменших квадратів для довільних базисних функцій

```
% Least Squares_Method
% МЕТОД НАЙМЕНШИХ КВАДРАТІВ ДЛЯ ВІЛЬНО ВИЗНАЧУВАНИХ
% БАЗИСНИХ ФУНКЦІЙ
% Синтаксис виклику:
% [A]=LSM01Free(F,V)
```

```
% F - Вихідна матриця таблично заданої функції вигляду:
% x1 x2 x3 x4 .... xM
% y1 y2 y3 y4 .... yM
% V - Літеральний вектор стовпець, що описує базисні
% функції:
% v1
% v2
% .....
% vN
% Наприклад (підготовка тестових даних):
% x=[0:pi/100:2*pi];
% y=eval('x+sin(x)-log(x+1)');
% F=[x;y];
% V = ['x^0 '; 'x '; 'sin(x) '; 'log(x+1)']
% УВАГА! Довжина рядків усіх літералів має бути суворо
% однаковою і за необхідності розширюватися пробілами
% праворуч.
% Виклик методу:
% [A]=LSM01(F,V)
% Результат:
% Максимальна абсолютна похибка: 3.5527e-014
% A =
% 0.0000
% 1.0000
% 1.0000
% -1.0000
function [A]=LSM01Free(F,V)
if isempty(F) || isempty(V)
disp('Одна з вхідних матриць не задана');
A=NaN;
return
end;
% Мінімальний вхідний контроль
if (size(F,1) ~= 2)
disp('Розміри матриці [F] задано неправильно');
A=NaN;
return
end;
nV=size(V,1);
if (size(F,2) < nV)
disp('Недостатньо точок у [F] для визначення коефіцієнтів
для [V]');
```

```
A=NaN;
return
end;
% Підготовка робочих матриць
C=zeros(nV,nV);
B=zeros(nV,1);
% Побудуємо систему лінійних рівнянь для апроксимації
% Для всіх рядків матриці 'C'
for row=1:nV
% Для всіх 'x' матриці 'F'
B(row,1)=0;
for k=1:size(F,2)
x=F(1,k);
B(row,1)=B(row,1)+eval(V(row,:))*F(2,k);
end;
% Для всіх колонок матриці 'C'
for col=1:nV
% Для всіх 'x' матриці 'F'
C(row,col)=0;
for k=1:size(F,2)
x=F(1,k);
C(row,col)=C(row,col)+eval(V(row,:))*eval(V(col,:));
end;
end;
end;
% Обчислимо коефіцієнти для апроксимуючого виразу
A=C\B;
% Обчислимо значення функції за апроксимуючим виразом
% Для всіх 'x' матриці 'F'
for k=1:size(F,2)
x=F(1,k);
y(1,k)=0;
% Для всіх рядків матриці 'V'
for row=1:nV
y(1,k)=y(1,k)+A(row,1)*eval(V(row,:));
end;
end;
% Побудуємо графік вихідної табличної функції%
plot(F(1,:),F(2,:));
% Побудуємо графік за результатами апроксимації
% plot(F(1,:),y);
% Побудуємо графік абсолютної похибки
```

```
delta=F(2,:)-y;
maxdelta=max(abs(delta));
disp(strcat('Максимальна абсолютна похибка: ',
num2str(maxdelta)));
if max(abs(delta)) > 1e-16
plot(F(1,:),delta);
end;
```

10.2.4. Тест методу найменших квадратів для довільних базисних функцій

```
% SCRIPT Test02_LSM
hold on;
x=[0:pi/100:2*pi];
y=eval('x+sin(x)-3*log(x+1)');
% plot(x,y);
% ВИХІДНІ ДАНІ
F=[x;y];
% ТЕСТ1
V = ['sin(x)']
[A2]=LSM01Free(F,V)
replay=input('Для продовження натисніть Enter...');
% ТЕСТ2
[A1]=LSM01(F,20)
```

10.3. Індивідуальні завдання

1. Виконайте всі приклади програм із теоретичної частини лабораторної роботи.

2. Відповідно до номера *N* за списком у журналі групи, записаному у вигляді $N = CM$, де C – старша цифра, M – молодша цифра, визначте таблично задану функцію (табл. 10.1) і базисну функцію (табл 10.2).

3. Виконайте апроксимацію таблично заданої функції методом найменших квадратів для певної базисної функції.

4. Оформіть звіт з лабораторної роботи.

Таблиця 10.1

M	**Функція**
1	$x + \sin(x) - \log(x+1)$
2	$x + \cos(x) - \log(x+1)$
3	$x + \sin(x) / x$
4	$x + \cos(x) / x$
5	$x + \arcsin(x) - \log(x+1)$
6	$x + \arccos(x) - \log(x+1)$
7	$x + \exp(x) + \cos(x)$
8	$x + \log(x+1) + \sin(x)$
9	$x + \ln(x) + \cos(x)$
0	$x + \text{arctg}(x) + \sin(x)$

Таблиця 10.2

C	**Базисна функція**
0	*x*^2
1	sin(*x*)
2	log(*x*+1)

Лабораторна робота 11

РОЗВ'ЯЗАННЯ СИСТЕМ ЛІНІЙНИХ РІВНЯНЬ В ПАКЕТІ MATLAB

Мета лабораторної роботи: отримання та закріплення знань, формування практичних навичок роботи з пакетом MATLAB при розв'язанні систем лінійних рівнянь.

11.1. Короткі відомості з теорії

11.1.1. Розв'язання системи лінійних рівнянь методом Крамера.

Метод Крамера – метод розв’язання систем лінійних рівнянь за правилом Крамера, який застосовується до систем лінійних рівнянь, де кількість рівнянь дорівнює кількості змінних і визначник системи не дорівнює нулю.

Нехай задана вихідна система лінійних рівнянь:

$$\begin{cases} a_{00}x_0 + a_{01}x_1 + a_{02}x_2 + \cdots + a_{0n}x_n = b_0; \\ a_{10}x_0 + a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1; \\ a_{20}x_0 + a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2; \\ \cdots\ \cdots\ \cdots\ \cdots\ \cdots\ \cdots\ \cdots\ \cdots\ \cdots\ \cdots\ \cdots\ \cdots\ \cdots\ \cdots\ \cdots\ \cdots\ \cdots \\ a_{n0}x_0 + a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n = b_n; \end{cases}$$

Тоді:

$$\boldsymbol{A} = \begin{pmatrix} a_{00}\ a_{01}\ a_{02} \ldots a_{0n} \\ a_{10}\ a_{11}\ a_{12} \ldots a_{1n} \\ a_{20}\ a_{21}\ a_{22} \ldots a_{2n} \\ \cdots\ \cdots\ \cdots\ \cdots\ \cdots\ \cdots\ \cdots\ \cdots \\ a_{n0}\ a_{n1}\ a_{n2} \ldots a_{nn} \end{pmatrix}; \boldsymbol{B} = \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ \ldots \\ b_n \end{pmatrix}.$$

Основною ідеєю методу є обчислення головного та додаткових визначень (детермінантів) матриці ***A***.

Головний визначник *D* обчислюється за допомогою функції det у складі функцій MATLAB

```
D = det(A);
```

Обчислення додаткових визначників *Di* спочатку вимагає формування додаткових матриць ***ABi***, які отримують шляхом заміни *i*- стовпця матриці ***A*** на матрицю ***B***:

$$\boldsymbol{AB}_i = \begin{pmatrix} a_{00} \dots b_0 \dots a_{0n} \\ a_{10} \dots b_1 \dots a_{1n} \\ a_{20} \dots b_2 \dots a_{2n} \\ \dots \dots \dots \dots \dots \dots \dots \\ a_{n0} \dots b_i \dots a_{nn} \end{pmatrix}.$$

Відповідно значення додаткових визначників можна за допомогою MATLAB отримати таким чином:

```
Di = det(ABi);
```

Якщо головний визначник не дорівнює нулю, то корені системи лінійних рівнянь обчислюються за формулою:

$$X_i = \frac{D_i}{D}.$$

У разі рівності нулю головного визначника, рішення відсутнє, а відповідну систему лінійних рівнянь називають погано визначеною.

Система називається спільною, якщо вона має хоча б один розв'язок, і несумісною, якщо в неї немає жодного розв'язку. Розв'язки вважаються різними, якщо хоча б одне зі значень змінних не збігається. Спільна система з єдиним розв'язком називається визначеною, за наявності більш ніж одного розв'язку – недовизначеною.

11.1.2. . Розв'язання системи лінійних рівнянь методом Кронекера-Капеллі

Метод Кронекера-Капеллі – метод розв’язання систем лінійних рівнянь, який базується на теоремі Кронекера-Капеллі.

Теорема Кронекера-Капеллі.

Система спільна тоді й тільки тоді, коли ранг матриці цієї системи збігається з рангом її розширеної матриці: *rank*(*A*) = *rank*(*AB*)

Матриця, що отримується конкатенацією матриці $\boldsymbol{A}$ і стовпця правих частин $\boldsymbol{B}$ шляхом конкатенації за стовпцями називається розширеною матрицею $\boldsymbol{AB}$ системи лінійних рівнянь.

$$\boldsymbol{AB} = \begin{pmatrix} a_{00}\ a_{01}\ a_{02} \dots a_{0n}\ b_0 \\ a_{10}\ a_{11}\ a_{12} \dots a_{1n}\ b_1 \\ a_{20}\ a_{21}\ a_{22} \dots a_{2n}\ b_2 \\ \dots\ \dots\ \dots\ \dots\ \dots\ \dots\ \dots\ \dots \\ a_{n0}\ a_{n1}\ a_{n2} \dots a_{nn}\ b_n \end{pmatrix}.$$

Приклад 11.1. Реалізація метода Кронекера-Капеллі.

```
A = rand(8)
B = rand(8,1)
AB =cat(2,A,B) % Розширена матриця
rA = rank(A)
rAB = rank(AB)
if rA==rAB
disp('Система лінійних рівнянь спільна');
else
disp('Система лінійних рівнянь НЕ СПІЛЬНА');
end;
```

11.1.3. Розв'язання системи лінійних рівнянь методом Гауса.

Метод Гауса – основний метод розв’язання систем лінійних рівнянь, який базується на послідовних елементарних операціях над рівняннями з метою приведення системи до верхньої трикутної (або редукованої до вигляду ідентичності) форми, після чого розвязки для змінних знаходяться за зворотнім ходом.

Нехай задано систему лінійних рівнянь у вигляді:

$$\begin{cases} a_{00}x_0 + a_{01}x_1 + a_{02}x_2 + \cdots + a_{0n}x_n = b_0; \\ a_{10}x_0 + a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1; \\ a_{20}x_0 + a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2; \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ a_{n0}x_0 + a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n = b_n; \end{cases}$$

Або в компактній формі:

$$\{_{k=0}^{n}\left(\sum_{i=0}^{n} a_{ki}x_k = b_k\right),$$

де *k* - індекс рядка, *i* - індекс змінної.

Основою розв'язання системи лінійних рівнянь методом Гаусса є послідовне виключення незалежних змінних із рядків системи. Таке виключення дає змогу привести вихідну систему до наступного еквівалентного вигляду:

$$\begin{cases} a_{00}x_0 + a_{01}x_1 + a_{02}x_2 + \cdots + a_{0n}x_n = b_0; \\ 0 + a_{11}^{*}x_1 + a_{12}^{*}x_2 + \cdots + a_{1n}^{*}x_n = b_1^{*}; \\ 0 + 0 + a_{22}^{*}x_2 + \cdots + a_{2n}^{*}x_n = b_2^{*}; \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ 0 + 0 + 0 + 0 + \cdots + a_{nn}^{*}x_n = b_n^{*}. \end{cases}$$

Якщо виявляється можливим таке перетворення, то таку еквівалентну систему називатимемо системою рівнянь у формі Гауса або просто Гаусовою системою. Розв'язати систему рівнянь у формі Гауса досить просто. Для цього спочатку слід знайти корінь *x*:

$$x_n = \frac{b_n^{*}}{a_{nn}^{*}}.$$

Після чого, послідовно від $k = n - 1$ до $k = 0$ застосовується формула, яка використовує вже обчислені значення коренів:

$$x_k = \frac{b_k^* - \sum_{i=k+1}^{n} a_{ki}^* x_k}{a_{kk}^*}, \text{де } 0 \le k < n.$$

Таким чином, основним завданням розв'язання системи рівнянь стає завдання з послідовного виключення незалежних змінних із рядків системи рівнянь, причому таким чином, щоб після необхідних винятків нова система залишалася еквівалентною вихідній. Метод Гаусс передбачає розв'язувати названу задачу виключення послідовно, виключаючи в кожному наступному рядку системи рівнянь чергову незалежну змінну. Такий метод стає можливим, якщо згадати, що будь-яка рівність зберігається якщо:

- обидві частини рівності помножити або поділити на одну й ту саму величину;
- до однієї рівності додати або відняти іншу рівність.

Якщо такі операції застосувати до деякого рядка системи лінійних рівнянь і отриманим результатом замінити вихідний рядок, то утворена система рівнянь буде еквівалентною вихідному, тобто матиме такі самі корені, як і вихідна. Розглянемо сформульовану ідею методу Гауса за окремими кроками.

11.1.3.1. Метод виключення довільної змінної із зазначеного рівняння в системі лінійних рівнянь.

Для визначеності позначимо сенс використовуваних індексів:

- *n* - кількість рядків у системі лінійних рівнянь;
- *m* - індекс незалежної змінної *x*, яка підлягає виключенню;
- *k* - індекс рядка системи рівнянь, у якому виключається змінна x_m.

Для виключення змінної x_m з рівняння з *k*-тим рядковим індексом послідовно виконаємо наступне:

Крок 1. Помножимо рівняння з *m*-тим рядковим індексом на величину:

$$\frac{a_{km}}{a_{mm}}$$

В результаті такого множення отримаємо

$$a_{m0}\frac{a_{km}}{a_{mm}}x_0 + a_{m1}\frac{a_{km}}{a_{mm}}x_1 + \cdots + a_{mm}\frac{a_{km}}{a_{mm}}x_k + \cdots + a_{mn}\frac{a_{km}}{a_{mm}}x_n = b_m\frac{a_{km}}{a_{mm}},$$

або в компактному вигляді:

$$\sum_{i=0}^{n} a_{mi}\frac{a_{km}}{a_{mm}}x_i = b_k\frac{a_{km}}{a_{mm}}, \text{де } 0 < k < n.$$

Крок 2. Віднімемо рівняння, яке ми отримали в результаті виконання кроку 1, від вихідного рівняння з k-тим рядковим індексом. У результаті віднімання однієї рівності з іншої, отримаємо таке:

$$\left(a_{k0} - a_{m0}\frac{a_{km}}{a_{mm}}\right)x_0 + \cdots + \left(a_{km} - a_{mm}\frac{a_{km}}{a_{mm}}\right)x_k + \cdots + \left(a_{kn} - a_{mn}\frac{a_{km}}{a_{mm}}\right)x_n =$$
$$= b_k - b_m\frac{a_{km}}{a_{mm}},$$

або в компактному вигляді:

$$\sum_{i=0}^{n}\left(a_{ki} - a_{mi}\frac{a_{km}}{a_{mm}}\right)x_i = b_k - b_m\frac{a_{km}}{a_{mm}}.$$

Якщо, для наочності, ввести позначення:

$$a^*_{ki} = \left(a_{ki} - a_{mi}\frac{a_{km}}{a_{mm}}\right) \ldots b^*_k = \left(b_k - b_m\frac{a_{km}}{a_{mm}}\right),$$

то попередній вираз можна записати в ще більш компактній формі:

$$\sum_{i=0}^{n} a^*_{ki}x_i = b^*_k, \text{де } 0 < k \leq n.$$

Звернемо увагу, що коефіцієнт при змінній x_m у цьому виразі завжди дорівнюватиме нулю:

$$\left(a_{km} - a_{mm}\frac{a_{km}}{a_{mm}}\right) \equiv 0.$$

Оскільки цей нульовий коефіцієнт є співмножником для змінної x_m, то змінна x_m фактично виключається з виразу

$$\sum_{i=0}^{n} a_{ki}^{*} x_i = b_k^{*}, \text{де } 0 < k \leq n.$$

Крок 3. Крім того, вираз, отриманий на кроці 2, є лінійно-залежним варіантом вихідного рівняння з k-тим рядковим індексом, а отже, цей вираз може замінити собою вихідне рівняння і при цьому нова система рівнянь залишиться еквівалентною вихідній системі рівнянь.

11.1.3.2. Метод побудови трикутної матриці шляхом послідовного виключення змінних.

Отже, тепер ми маємо у своєму розпорядженні метод, який дає змогу виключити будь-яку змінну з будь-якого рівняння системи лінійних рівнянь. Таким чином, перетворення системи лінійних рівнянь можна здійснити шляхом послідовного застосування цього методу.

Розглянемо таке послідовне виключення:

Для початку, у вихідній системі рівнянь виключимо змінну x_0 з усіх рівнянь, крім рівняння з нульовим рядковим індексом. Це приведе нас до нової еквівалентної системи рівнянь виду:

$$\begin{cases} a_{00}x_0 + a_{01}x_1 + a_{02}x_2 + \cdots + a_{0n}x_n = b_0; \\ 0 + a_{11}^{*}x_1 + a_{12}^{*}x_2 + \cdots + a_{1n}^{*}x_n = b_1^{*}; \\ 0 + a_{21}^{*}x_1 + a_{22}^{*}x_2 + \cdots + a_{2n}^{*}x_n = b_2^{*}; \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ 0 + a_{n1}^{*}x_1 + a_{n2}^{*}x_2 + \cdots + a_{nn}^{*}x_n = b_n^{*}; \end{cases}$$

Звернемо увагу, що деяка частина в цій системі рівнянь, а саме:

$$\begin{cases} a_{11}^* x_1 + a_{12}^* x_2 + \cdots + a_{1n}^* x_n = b_1^*; \\ a_{21}^* x_1 + a_{22}^* x_2 + \cdots + a_{2n}^* x_n = b_2^*; \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ a_{n1}^* x_1 + a_{n2}^* x_2 + \cdots + a_{nn}^* x_n = b_n^*; \end{cases}$$

стала незалежною від змінної x_0, а отже, може бути розв'язана як самостійна система лінійних рівнянь. Назвемо таку систему рівнянь підсистемою. Особливо слід зазначити, що розмірність такої підсистеми стала на одиницю меншою стосовно вихідної системи. Для нас таке зниження розмірності може означати, що послідовне повторення процедури виключення чергової змінної в кожній наступній підсистемі обов'язково приведе до фінішної підсистеми з розмірністю, що дорівнює одиниці, а саме:

$$a_{nn}^* x_n = b_n^*.$$

Очевидно, що фінішну підсистему можна розв'язати як звичайне рівняння, а підсумкова система рівнянь, що включає фінішну підсистему, матиме вигляд:

$$\begin{cases} a_{00} x_0 + a_{01} x_1 + a_{02} x_2 + \cdots + a_{0n} x_n = b_0; \\ a_{11}^* x_1 + a_{12}^* x_2 + \cdots + a_{1n}^* x_n = b_1^*; \\ a_{22}^* x_2 + \cdots + a_{2n}^* x_n = b_2^*; \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ a_{nn}^* x_n = b_n^*; \end{cases}$$

Таким чином, у результаті послідовного виключення однієї змінної в кожній наступній підсистемі ми отримали відповідь на задачу і тепер можемо обчислити всі корені системи лінійних рівнянь.

11.2. Приклади розв'язання систем лінійних рівнянь.

11.2.1. Реалізація методу Крамера

Приклад 11.2. Покрокова реалізація методу Крамера

```
% Вихідні дані
a=[-1 2 7; 4 3 1; 7 5 -8]
x1=[1; 2; 3]
b=a*x1
% Метод Крамера. Покрокова реалізація
d=det(a)
% Обчислення x1
aw=a;
aw(:,1)=b;
d1=det(aw);
x1=d1/d
% Обчислення x2
aw=a;
aw(:,2)=b;
d2=det(aw);
x2=d2/d
aw=a;
% Обчислення x3
aw(:,3)=b;
d3=det(aw);
x3=d3/d
```

Приклад 11.3. Універсальна реалізація методу Крамера.

```
% Розв'язання системи лінійних рівнянь методом Крамера
% Синтаксис:
%              [x]=fkramer01(a,b)
% a - Вихідна матриця коефіцієнтів системи лінійних
% рівнянь виду:
%      a11 a12 a13 a14    a1N
%      a21 a22 a23 a24    a2N
%      ................
%       aN1 aN2 aN3 aN4   aNN
% b - Вихідна матриця вільних членів системи лінійних рівнянь
% виду:
```

```
%      b1
%      b2
%      ...
%      bN
% x - Обчислюваний вектор коренів %
function [x]=fkramer01(a,b)
if isempty(a) || isempty(b)
disp('Одна з матриць не задана');
x=NaN;
return
end;
if (size(a,1) ~= size(a,2)) || (size(a,1) ~= size(b,1))
disp('Розміри матриць системи лінійних рівнянь задано
неправильно');
x=NaN;
return
end;
d=det(a);
if d ~= 0
for n=1:size(a,2)
buf=a(:,n);
a(:,n)=b;
x(n,1)=det(a)/d;
a(:,n)=buf;
end;
else
x=NaN;
if (sum(abs(b))==0) && (d == 0)
x=zeros(size(a,2),1);
disp('Система має тривіальне рішення');
else
disp('Матриця погано зумовлена');
end;
end;
```

Приклад 11.4. Тест точності методу Крамера

```
function maxerr=ftest01kramer(n)
if n < 2
n=2
end;
if n > 500
```

```
n=500
end;
% Створити випадкову матрицю і випадкові корені
A = rand(n,n);
x1=rand(n,1);
% Обчислити відповідні вільні члени
b=A*x1;
% ВИКОНАТИ Тест
x2 = fkramer01(A,b);
err=x2-x1;
maxerr=max(abs(err));
grid on;
plot(err);
```

Приклад 11.5. Приклади розв'язання систем лінійних рівнянь засобами MATLAB. Тестування (*x*=*a**b*) методу MATLAB

```
% Тест точності методу (x=a\b)
function maxerr=ftest01LSE01(n)
if n < 2
n=2
end;
if n > 500
n=500
end;
% Створити випадкову матрицю і випадкові корені
a = rand(n,n);
x1=rand(n,1);
% Обчислити відповідні вільні члени
b=a*x1;
% ВИКОНАТИ Тест
x2 = a\b;
err=x2-x1;
maxerr=max(abs(err));
grid on;
plot(err);
```

Пример 11.6. Тестування (*x*=*inv*(*a*)**b*) методу лінійної алгебри

```
% Тест точності методу (x=inv(a)*b)
function maxerr=ftest01LSE02(n)
```

```
if n < 2
n=2
end;
if n > 500
n=500
end;
% Створити випадкову матрицю і випадкові корені
a = rand(n,n);
x1=rand(n,1);
% Обчислити відповідні вільні члени
b=a*x1;
% ВИКОНАТИ Тест
x2 = inv(a)*b;
err=x2-x1;
maxerr=max(abs(err));
grid on;
plot(err);
```

11.2.2. Приклади реалізації методу Гауса.

Приклад 11.7. Розв'язання системи лінійних рівнянь методом Гауса

```
% Синтаксис:
% [x]=fgauss01(ab)
% ab - Вихідна матриця системи лінійних
% рівнянь виду:
%      a11 a12 a13 a14    a1N b1
%      a21 a22 a23 a24    a2N b2
%      ..................
% aN1 aN2 aN3 aN4 .. aNN bN
% x - Обчислюваний вектор коренів
function [x]=fgauss01(ab) if isempty(ab)
disp('Матрицю системи лінійних рівнянь не задано');
x=NaN;
return end;
nb = size(ab,2);
nx = size(ab,1);
if (nb < 2) || (nb ~= (nx+1))
disp('Розміри  матриці  системи  лінійних  рівнянь  задані
неправильно');
x=NaN;
return end;
```

```
% Побудова Гаусової матриці
for nM=1:(nx-1)
% Встановлення на головну діагональ рядка з максимальним
елементом ab(nM,nM)
[v,n]=max(abs(ab(:,nM)));
if (n > nM)
s=ab(n,:);
ab(n,:) = ab(nM,:);
ab(nM,:)=s;
end;
% Пастка нерозв'язних ситуацій
if ab(nM,nM) == 0
disp('Матриця погано зумовлена');
x=NaN;
return;
end;
% Побудова трикутної Гаусової матриці
for row =(nM + 1):nx
koef =ab(row,nM)/ab(nM,nM);
for n=1:nb
if n == nM
ab(row,n)=0;
else
ab(row,n)=ab(row,n)-ab(nM,n)*koef;; end;
end;
end;
end;
% Обчислення коренів
% Пастка нерозв'язних ситуацій
if ab(nx,nx) == 0
disp('Матриця погано зумовлена');
x=NaN;
return;
end;
x=zeros(nx,1);
row = nx;
x(nx)=ab(row,nb)/ab(row,nx); % Останній корень while row > 1
row=row-1;
s=0;
for n=1:nx
if ab(row,n) ~= 0 s=s+ab(row,n)*x(n);
end;
```

```
end;
x(row)=(ab(row,nb)-s)/ab(row,row); % Черговий корінь end;
```

Приклад 11.8. Тестування реалізації методу Гауса

```
% Тест точності методу ГАУСА
function maxerr=ftest01gauss(n) if n < 2
n=2
end;
if n > 500
n=500
end;
% Створити випадкову матрицю і випадкові корені
a = rand(n,n);
x1=rand(n,1);
% Обчислити відповідні вільні члени b=a*x1;
% Создать матрицу линейных уравнений ab=cat(2,a,b);
% ВИКОНАТИ Тест
x2 = fgauss01(ab);
err=x2-x1;
maxerr=max(abs(err));
grid on;
plot(err);
```

11.3. Порівняльні тести методів розв'язання SLE

11.3.1. Порівняльний тест точності методів

```
function [maxerr]=ftest01LSE_All(n)
if n < 2
n=2
end;
if n > 100
n=100
end;
% Створити випадкову матрицю і випадкові корені
a = rand(n,n);
x1=rand(n,1);
b=a*x1;
ab=cat(2,a,b);
% Тест 1
x2 = a\b;
```

```
maxerr(1,1)=max(abs(x2-x1));
% Тест 2
x2 = inv(a)*b;
maxerr(2,1)=max(abs(x2-x1));
% Тест 3 (Kramer)
x2 = fkramer01(a,b);
maxerr(3,1)=max(abs(x2-x1));
% Тест 4 (Gauss)
x2 = fgauss01(ab);
maxerr(4,1)=max(abs(x2-x1));
% Отчет
[v,ind]=max(abs(maxerr));
switch ind
case 1
disp(strcat('Max.  помилка  при  x=A\b.  Помилка  :  ',
num2str(v)));
case 2
disp(strcat('Max.  помилка  при  x=inv(A)*b.  Помилка  :  ',
num2str(v)));
case 3
disp(strcat('Max. помилка при x=fkramer01(A,b). Помилка : ',
num2str(v)));
case 4
disp(strcat('Max. помилка при x=fgauss01(ab). Помилка : ',
num2str(v)));
end;
```

11.3.2. Порівняльний тест швидкості методів Гауса і Крамера

```
function ftest02_Gauss_Kramer(nm)
if (nm < 100) || (nm > 200)
nm=150;
disp(strcat('Параметр виклику замінено на :', num2str(nm)));
end;
disp('Порівняльний тест ШВИДКОСТІ методів GAUSS (blue) і
KRAMER (red)');
disp(strcat('Зараз  буде  виконано  обчислення:  ',
num2str(2*nm), '
систем уравнений'));
disp('ОЧІКУЙТЕ ЗАВЕРШЕННЯ тривалої операції');
pproc=0;
nproc=0;
```

```
for n = 1:nm
A = rand(n,n);
b = rand(n,1);
AB = cat(2,A,b);
x(n)=n;
tic
y2 = fgauss01(AB);
t1(n) = toc;
tic
y2 = fkramer01(A,b);
t2(n) = toc;
% Відсоток виконання
nproc=100*(n-rem(n,10))/nm;
if nproc > pproc
disp(strcat('Виконано :', num2str(nproc), '%'));
pproc=nproc;
end;
end;
clf; % Очищення графіків
disp('ГОТОВО! Дивіться графіки...');
hold on;
plot(x,t1); % GAUSS
plot(x,t2,'-r'); % KRAMER
```

11.4. Індивідуальні завдання

1. Виконайте всі приклади з теоретичної частини лабораторної роботи.

2. Задайтеся системою лінійних рівнянь *N*-го порядку (порядок позначає кількість рівнянь у системі), де *N* - будь-яке число в діапазоні від 3 до 6.

3. Розв'яжіть складену систему лінійних рівнянь методами Крамера і Гауса.

4. Виконайте порівняння точності та швидкості роботи методів Крамера і Гауса.

5. Оформіть звіт з лабораторної роботи.

Лабораторна робота 12

ІНТЕГРАЛЬНА ФОРМА ЗАДАЧІ АПРОКСИМАЦІЇ ТАБЛИЧНО ЗАДАНОЇ ФУНКЦІЇ В ПАКЕТІ MATLAB

Мета лабораторної роботи: отримання та закріплення знань, формування практичних навичок роботи з пакетом MATLAB під час інтегральної апроксимації таблично заданої функції.

12.1. Короткі відомості з теорії

На рис. 12.1. легко помітити, що множину точок, яка містяться між кривими $F(x)$ і $P(x)$, можна розглядати з позиції площі (у геометричному сенсі цього слова). Іншими словами, можна стверджувати: якщо загальна площа, утворена областями відхилень між функціями $F(x)$ і $P(x)$, наближається до нуля, то в граничному випадку початкова і апроксимувальна функція стають ідентичними.

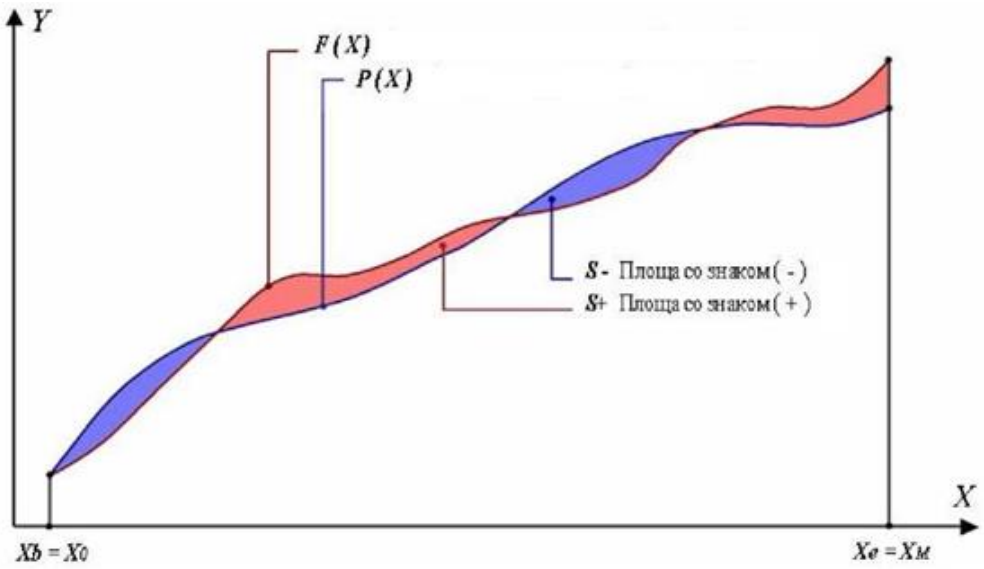


Рис. 12.1. Інтегральне відхилення між апроксимованою та апроксимованою функціям.

Це, природним чином, приводить нас до ідеї узагальнити в інтегральній формі умову мінімізації оцінки *D*. Виконаємо таке узагальнення шляхом множення сторін рівності на значення:

$$\Delta x = x_{i+1} - x_i.$$

В результаті отримаємо:

$$D\Delta x = \sum_{m=0}^{M} \left(F(x_m) - \sum_{i=0}^{k} a_i v_i(x_m) \right)^2 \Delta x.$$

При прагненні Δx до нуля ми автоматично отримуємо нашу оцінку в інтегральній формі:

$$S = \lim_{\Delta x \to 0}(D\Delta x) = \int_{X_h}^{X_e} \left(F(x_m) - \sum_{i=0}^{k} a_i v_i(x) \right)^2 dx$$

Геометричний зміст такої оцінки можна сформулювати як площу, що утворена квадратичною різницею між функціями $F(x)$ і $P(x)$. Таким чином, в інтегральній формі, умова для пошуку коефіцієнтів a, за якого досягається мінімальне значення інтегральної оцінки S, можна записати у вигляді:

$$\frac{\partial}{\partial a_0} S\Big(F(x), (a_0, \dots, a_N), \big(v_0(x), \dots, v_N(x)\big)\Big) = 0;$$

$$\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots$$

$$\frac{\partial}{\partial a_N} S\Big(F(x), (a_0, \dots, a_N), \big(v_0(x), \dots, v_N(x)\big)\Big) = 0.$$

З огляду на те, що перехід до інтегральної форми зводиться фактично до заміщення звичайної суми за аргументом *x* на суму інтегральну, то такий перехід можна виконати в усіх виразах, що розглядалися в дискретній формі розв'язання

задачі апроксимації, шляхом простої заміни:

$$\sum_{m=0}^{M} \varphi(x_m) \to \int_{X_b}^{X_e} \varphi(x)dx.$$

12.1.1. Розв'язання задачі апроксимації в інтегральній формі

Якщо опустити послідовний ланцюжок кроків, які ми розглядали під час розв'язування задачі апроксимації в дискретній формі, то з урахуванням переходу від індексів дискретних значень аргументу *x*, до меж інтегрування:

$$x_0 = x_b;$$
$$x_M = x_e.$$

Розв'язок задачі апроксимації в інтегральній формі можна одразу подати в такому вигляд.

Система лінійних рівнянь:

$$\begin{cases} a_0 C_{0,0} + \cdots + a_N C_{0,N} = B_0; \\ a_0 C_{1,0} + \cdots + a_N C_{1,N} = B_1; \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ a_0 C_{N,0} + \cdots + a_N C_{N,N} = B_N. \end{cases}$$

Константи при незалежних змінних a системи лінійних рівнянь:

$$B_k = \int_{X_b}^{X_e} F(x) \cdot v_k(x)dx;$$
$$C_{k,n} = \int_{X_b}^{X_e} v_k(x) \cdot v_n(x)dx,$$

де *n* (*n* = 0, ..., *N*) – індекс, перелічує базисні функції *v(x)* або рядки системи лінійних рівнянь; *k* (*k* = 0, ..., *N*) – індекс, що вказує на конкретний рядок у системі лінійних рівнянь.

12.1.2. Вибір базисних функцій

Виконуючи розв'язання задачі в дискретній та інтегральній формі, єдиною вимогою, яку ми накладали на апроксимуючу функцію $P(x)$, була вимога про відомий аналітичний вигляд базисних функцій $v(x)$:

$$P(x) = \sum_{n=0}^{N} a_n v_n(x).$$

Таке широке трактування класу функцій $v(x)$ виявилася можливою завдяки тому, що ми не застосовували жодних операцій, які потребують урахування властивостей і внутрішньої структури цих функцій. Строго кажучи, нам було достатньо тільки того, що ці функції зображували деяке перетворення над аргументом x. Таким чином, множину функцій $v(x)$ вдалося зберегти досить широкою, що безумовно є перевагою для математичного аналізу, але, водночас, є явним недоліком для практичного застосування. Якщо акцентувати наші зусилля на практичному застосуванні, то буде потрібно вводити (для отримання практичних результатів) деякі додаткові умови або вимоги, які природним чином обмежать множину цих функцій.

12.1.3. Завдання апроксимації в ортогональних базисах

Одним із таких підходів, який (серед іншого) істотно спрощує практичне розв'язання задач апроксимації, є підхід, що накладає на множину функцій *v(x),* вимоги взаємної ортогональності. Така вимога, в контексті нашої задачі, призводить до її істотного спрощення і може бути сформульована, в її найпростішій формі, як:

$$C_{k,n} = \int_{X_h}^{X_e} x_k(x) \cdot x_n(x) dx = \begin{cases} 0, k \neq n; \\ C, k = n. \end{cases}$$

При виконанні наведеної вимоги, система лінійних рівнянь, необхідних нам для обчислення коефіцієнтів функції *P(x),* набуває вигляду:

$$\begin{cases} a_0 C + 0 + 0 + \cdots + 0 = B_0; \\ 0 + a_1 C + 0 + \cdots + 0 = B_1; \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ 0 + 0 + a_k C + \cdots + 0 = B_k; \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ 0 + 0 + 0 + \cdots + a_N C = B_N; \end{cases}$$

Вочевидь, що наша система не тільки спростилася, а й набула вже повністю вирішеного характеру. Іншими словами, будь-який шуканий коефіцієнт може бути обчислений як:

$$a_k = \frac{B_k}{C} = \frac{1}{C}\int_{X_b}^{X_e} F(x) \cdot v_k(x) dx.$$

Особливо слід підкреслити, що такий підхід призводить до того, що коефіцієнти *a* можуть обчислюватися незалежно один від одного. Важливість цього зауваження можна показати таким чином:

Нехай, у нас є результат розв'язаної задачі апроксимації:

$$F(x) \cong P(x) = \sum_{n=0}^{N} a_n v_n(x).$$

Якщо базисні функції $v(x)$, взаємно ортогональні, то з функції $P(x)$ можна вилучити будь-яку функцію $v(x)$ разом з її коефіцієнтом, і це не потребуватиме перерахунку всіх інших коефіцієнтів. Єдиним наслідком такого виключення буде підвищення помилки апроксимації пропорційно вазі виключеної функції.

У тому ж випадку, коли базисні функції $v(x)$ не є взаємно ортогональними, виключення деякої функції $v(x)$ з функції $P(x)$ потребуватиме повного перерахунку всієї задачі. Це зумовлено тим, що зміна розмірності базису призведе до зміни розмірності системи лінійних рівнянь, а отже, до необхідності її перерахунку.

12.1.4. Класичний розв'язок задачі апроксимації в ортогональному базисі

Для розуміння деяких особливостей ортогональних наборів функцій, виконаємо низку перетворень, які повторюють шлях розв'язання задачі апроксимації в ортогональному базисі максимально компактна. Отже, для вихідного наближення:

$$F(x) \cong \sum_{n=0}^{N} a_n v_n(x).$$

Помножимо обидві сторони цього виразу на добуток двох функцій: (роль функції гамма, буде визначена нижче за текстом):

$$F(x) \cdot \gamma(x) \cdot v_k(x) \cong \sum_{n=0}^{N} a_n \cdot \gamma(x) \cdot v_k(x) \cdot v_n(x).$$

Проінтегруємо обидві сторони отриманого виразу в деякій області визначення аргументу:

$$\int_{X_b}^{X_e} F(x) \cdot \gamma(x) \cdot v_k(x) dx \cong \sum_{n=0}^{N} a_n \cdot \int_{X_b}^{X_e} \gamma(x) \cdot v_k(x) \cdot v_n(x) dx\,.$$

У цьому випадку, умова ортогональності набуде такого вигляду:

$$\int_{X_b}^{X_e} \gamma(x) \cdot v_k(x) \cdot v_n(x) dx = \begin{cases} 0, k \neq n; \\ C, k = n. \end{cases}$$

У результаті таких формальних перетворень, легко отримати вже знайоме нам рішення:

$$a_k \cong \frac{1}{C} \int_{X_b}^{X_e} F(x) \cdot \gamma(x) \cdot v_k(x) dx.$$

Особливістю наведених перетворень є включення в них спеціальної (вагової) функції $Y(x)$, роль якої розширити множину функцій, здатних задовольняти умову ортогональності, або забезпечити рівність $C = 1$.

На сьогоднішній день, результати дослідження функцій, дозволили отримати безліч наборів ортогональних функцій. Питання знаходження таких наборів перебувають у компетенції спеціальних розділів математики і виходять за рамки цієї роботи. З цієї причини, ми коротко згадаємо тільки незначний ряд класичних ортогональних наборів. У числі такої низки класичних наборів ортогональних функцій, як правило, називають:

- Поліноми Лежандра;
- Поліноми Чебишева;
- Поліноми Лагерра;
- Поліноми Ерміта;
- Функції, що утворюють ряди Фур'є.

12.1.5. Приклади класичних ортогональних функцій

Поліноми Чебишева

Основною особливістю, що визначила включення поліномів Чебишева до складу прикладів, є можливість записати функції Чебишева як у тригонометричній, так і в степеневій формі.

Аналітичний запис функції.

Тригонометрична форма:

$$v_k(x) = \cos(k \cdot \arccos(x)).$$

Степенева форма. Степеневу форму функцій Чебишева представимо рекурентними правилами їхнього обчислення, які дають змогу оптимізувати алгоритми обчислення значень цих функцій:

$$v_0(x) = 1;$$

$$v_1(x) = x;$$
$$v_{k+1}(x) = 2x \cdot v_k(x) - v_{k-1}(x);$$

Вагова функція.

$$\gamma(x) = \frac{1}{\sqrt{1-x^2}}.$$

Виконання умов ортогональності та інтервал ортогональності.

$$\int_{X_b}^{X_e} \gamma(x) \cdot v_k(x) \cdot v_n(x) dx = \begin{cases} 0, k \neq n; \\ \frac{\pi}{2}, k = n \neq 0; \\ \pi, k = n = 0, \end{cases}$$

де:

$$X_b = -1; X_e = +1.$$

Функції, що утворюють ряди Фур'є

Основною особливістю, яка визначила включення до складу прикладів функцій, що утворюють ряди Фур'є, є широке використання рядів Фур'є в різних технічних додатках.

Аналітичний запис функції.

$$v_k(x) = a_k \sin(kx) + b_k \cos(kx).$$

При цьому функцію апроксимації $P(x)$, зазвичай представляють у формі з попередньо обчисленим для нульового індексу значенням $v(x)$. Тобто:

$$P(x) = b_0 + \sum_{k=1}^{\infty} a_k \sin(kx) + \sum_{k=1}^{\infty} b_k \cos(kx).$$

Вагова функція.

$$\gamma(x) = 1.$$

Виконання умов ортогональності та інтервал ортогональності.

Інтервал ортогональності для функцій Фур'є визначається таким чином:

$$X_e = \alpha + 2\pi;$$
$$X_b = \alpha,$$

де *a* – будь-яка початкова фаза.

Виконання умов ортогональності для функцій Фур'є, традиційно доводять для окремих добутків, що виникають під час перемноження функцій *v*(*x*).

$$\int_{X_b}^{X_e} \cos(kx) \cdot \cos(nx)\, dx = \begin{cases} 0, k \neq n; \\ \pi, k = n. \end{cases}$$
$$\int_{X_b}^{X_e} \sin(kx) \cdot \sin(nx)\, dx = \begin{cases} 0, k \neq n; \\ \pi, k = n. \end{cases}$$
$$\int_{X_b}^{X_e} \sin(kx) \cdot \cos(nx)\, dx = 0.$$

Таким чином, розв'язок задачі апроксимації в базисі функцій Фур'є можна (пропускаючи очевидні проміжні перетворення) записати у вигляді:

$$a_k = \frac{1}{\pi} \int_{\alpha}^{\alpha+2\pi} F(x) \sin(kx)\, dx;$$
$$b_k = \frac{1}{\pi} \int_{\alpha}^{\alpha+2\pi} F(x) \cos(kx)\, dx;$$
$$b_0 = \frac{1}{2\pi} \int_{\alpha}^{\alpha+2\pi} F(x) dx.$$

Примітка. Як уже зазначалося, базис Фур'є для дискретного спектра має вигляд:

$$v_k(x) = a_k \sin(kx) + b_k \cos(kx).$$

Звернемо увагу, що складові цього виразу досить просто інтерпретувати як проекції векторів з довжинами a_k і b_k відповідно спроектовані на осі координат Y і X. Така особливість дає змогу представити результат апроксимації в базисі Фур'є в компактнішій формі, яка, як правило, застосовується в більшості інженерних рішень.

Для отримання згаданої компактної форми, розглянемо відомі тригонометричні формули:

$$a \cdot \sin(x) + b \cdot \cos(x) = \sqrt{a^2 + b^2} \cdot \sin(x + \varphi)$$

де

$$\sin(\varphi) = \frac{b}{\sqrt{a^2 + b^2}}.$$

За допомогою наведених формул остаточний результат апроксимації рядом Фур'є в такому компактному вигляді:

$$P(x) = b_0 + \sum_{k=1}^{\infty} L_k \cdot \sin(kx + \varphi_k)$$

де

$$L_k = \sqrt{a_k^2 + b_k^2}.$$

12.1.6. Перенормування інтервалу ортогональності

Розглянуті нами приклади ортогональних функцій, показують нам цілком конкретні інтервали на яких виконується умова ортогональності.

Якщо раніше, коли умови ортогональності тільки формулювалися, це не обмежувало сферу визначення функції $F(x)$, оскільки сама $F(x)$ неявно задавала цей інтервал, то під час конкретизації базису картина дзеркально змінилася. При цьому, стає очевидним, що практично для всіх функції $F(x)$ буде потрібно приведення області їх визначення до конкретного інтервалу ортогональності.

Таке завдання отримало назву перенормування інтервалу ортогональності і зводиться до знаходження такого масштабу, за допомогою якого область визначення функції $F(x)$ цілком відображається в інтервал ортогональності.

Для розв'язання задачі перенормування визначимо функцію F як функцію від незалежної змінної t, з областю визначення (t_b, t_e).

У цьому випадку можна показати, що незалежні змінні x і t можуть бути пов'язані простим відношенням, яке забезпечить нам необхідне масштабне перетворення:

$$x - x_b = \frac{x_e - x_b}{t_e - t_b} \cdot (t - t_b).$$

Подальша підстановка цього виразу в усі, розглянуті для ортогональних функцій, перетворення дасть змогу здійснити перехід, практично до довільних меж області визначення функції $F(t)$.

Примітка. Наведемо приклад перенормування.

Найбільш відомим із прикладів є перенормування ортогональних функцій Фур'є для вирішення завдань аналізу сигналів:

$$x = \frac{2\pi}{t_e - t_b} \cdot t = \frac{2\pi}{T} = \omega t.$$

За такого перенормування, розглянуті раніше вирази для апроксимації рядом Фур'є набудуть вигляду:

$$F(t) \cong b_0 + \sum_{k=1}^{N} a_k \sin(k\omega t) + \sum_{k=1}^{N} b_k \cos(k\omega t).$$

Відповідно, коефіцієнти або розв'язок задачі, можна представити таким чином:

$$a_k = \frac{2}{T} \int_{t_b}^{t_e} F(\omega t) \sin(k\omega t)\, dt;$$

$$b_k = \frac{2}{T}\int_{t_b}^{t_e} F(\omega t)\cos(k\omega t)\,dt;$$

$$b_0 = \frac{1}{T}\int_{t_b}^{t_e} F(\omega t)dt.$$

Як результат, ми перейшли від кордонів, заданих у кутових одиницях, до кордонів, які задані в необхідних нам межах часу. Аналогічним чином виконується перенормування для меж, заданих відстанню, що зазвичай застосовується під час обробки растрових зображень.

12.1.7. Оцінка якості розв'язання задачі апроксимації

Розглядаючи задачу апроксимації в дискретній формі (на кроці 2) ми вводили відхилення між функцією, що апроксимується, та функцією, що апроксимується, у формі простої різниці:

$$d(x) = F(x) - P(x).$$

Легко помітити, що така різниця рівнозначна визначенню абсолютної помилки зі знаком, яке дається в метрології. Як правило, крім абсолютної помилки, метрологія також вводить відносні помилки як у загальній, так і в наведеній до кінця шкали формі. У рамках уже прийнятих нами позначень, це можна записати в такому вигляді:

$$\delta = \frac{d(x)}{F(x)} \cdot 100\%;$$

$$\delta_{\text{пр}} = \frac{\max(d(x))}{F(x_e)} \cdot 100\%.$$

На перший погляд, такі інструменти оцінювання якості можна розглядати чи не як єдині під час оцінювання розв'язання нашої задачі. Дійсно, до розв'язання конкретної задачі апроксимації ми стикаємося зі значним числом невизначеностей:

- До функції $F(x)$ ми не пред'являли, крім загальних, якихось особливих

вимог або обмежень, що впливають на якість апроксимації.

- Практично завжди, функції $F(x)$ містять у собі всілякі шуми і помилки, оскільки, як правило, є результатами тих чи інших вимірювань.
- До отримання конкретного екземпляра функції $F(x)$, практично нічого не можна сказати про вибір (за типом і розміром) базису функції $P(x)$.

12.2. Укрупнені алгоритми розв'язання задачі апроксимації

У процесі розгляду шляхів розв'язання задачі апроксимації, нами було отримано дві форми її розв'язання, а саме - неортогональне та ортогональне розв'язання. Розглянемо ці форми розв'язання послідовно.

12.2.1. Дискретне рішення в неортогональному базисі.

Отже, дискретний розв'язок задачі в неортогональному базисі було отримано нами як розв'язок системи лінійних рівнянь відносно невідомих масштабних коефіцієнтів a:

$$\begin{cases} a_0 C_{0,0} + \cdots + a_N C_{0,N} = B_0; \\ a_0 C_{1,0} + \cdots + a_N C_{1,N} = B_1; \\ \cdots\ \cdots\ \cdots\ \cdots\ \cdots\ \cdots\ \cdots\ \cdots\ \cdots\ \cdots\ \cdots\ \cdots \\ a_0 C_{N,0} + \cdots + a_N C_{N,N} = B_N. \end{cases}$$

де

$$B_k = \sum_{m=0}^{M} F(x_m) \cdot v_k(x_m);$$

$$C_{k,n} = \sum_{m=0}^{M} v_k(x_m) \cdot v_n(x_m),$$

m (m = 0, ..., m) - індекс перелічує точки аргументу функції $F(x)$; n (n = 0, ..., n) – індекс перераховує базисні функції $v(x)$ або рядки системи лінійних рівнянь; k (k = 0, ..., n) - індекс вказує на конкретний рядок у системі лінійних рівнянь.

Алгоритм для обчислення коефіцієнтів системи лінійних рівнянь.

Вихідні дані. *Я*к вихідні дані алгоритму визначимо:

Функцію $F(x)$, представлену двовимірним масивом $FTAB[m, i]$ розмірністю $[0\ldots m, 0...1]$, у якому індекс m перераховуватиме точки функції $F(x)$, а індекс i відповідно вказуватиме на значення аргументу x (за $i = 0$) або значення функції $F(x)$. (при $i = 1$). Функцію $vk(x)$, представлену процедурною функцією $v(k, x)$.

Вихідні дані. *Я*к вихідні дані алгоритму розглядатимемо систему лінійних рівнянь, представлену як двовимірний масив $ESTAB[k, j]$ розмірністю $[0...n, 0\ldots n+1]$, у якому індекс k перелічуватиме рядки системи лінійних рівнянь, а індекс $\boldsymbol{j}$, відповідно, вказуватиме на стовпці цієї системи.

Завдання алгоритму. Завданням алгоритму будемо вважати, обчислення всіх значень масиву ESTAB$[k, j]$.

Умова оптимальності. Будемо вважати алгоритм достатньо оптимальним, якщо в процесі його виконання знадобиться мінімальна кількість звернень до процедури $v(k, x)$.

Попередній аналіз.

Крок 1. Розглядаючи вирази для обчислення коефіцієнтів B_k і $C_{k,n}$, слід зауважити, що при будь-якій організації алгоритму, нам належить обчислити функцію $vk(x)$, для всіх значень її інденкса k, а також для всіх значень аргументу X. Більш того, аналіз системи лінійних рівнянь показує, що обчислення $vk(x)$ буде багаторазово дублюватися, якщо алгоритмічно побудувати обчислення коефіцієнтів B_k і $C_{k,n}$, у формі незалежних процедур.

Висновок кроку 1. З огляду на те, що в сучасних засобах обчислювальної техніки пам'ять перестала бути критичним і дорогим ресурсом, оптимальним рішенням можна вважати створення й обчислення додаткового двовимірного масиву $VTAB\ [m, k]$ розмірністю $[0...m, 0\ldots n]$. В масиві $VTAB\ [m, k]$ індекс m перераховуватиме значення незалежної змінної x, взятої з масиву $FTAB[m, 0]$, а індекс k, відповідно позначати, що результат обчислений функцією $vk(x)$.

Крок 2. Обчислення масиву *VTAB* [*m, k*] можна оптимізувати в частині кількості операцій, якщо виконувати його обчислення рядково, тобто обчислювати значення всіх функції *vk*(*x*) для конкретного значення незалежної змінної *x* за допомогою окремої процедури. Нехай така процедура буде розроблена як процедура *VS*(*x, k*).

Висновок кроку 2. Якщо базисні функції *vk(x)* можна записати в рекурентній формі, то обчислення рядка *VTAB*[*m, k*] за допомогою процедури *VS(x, k)* можна реалізувати простим циклом з мінімальним числом операцій. Наприклад:

Нехай функції базису в явному вигляді визначені як $vk(x)=x^k$, тобто, є елементами степеневого ряду.

У цьому випадку такі функції можна також представити в рекурентній формі $vk+1(x)=x\ vk(x)$, де $v_0(x)=1$.

Очевидно, що для обчислення рядка *VTAB* [*m, k*] нам знадобиться тільки *n*-1 раз виконати множення *x* на *x*. Ще оптимальніший ефект виходить у разі застосування рекурентної форми, якщо базис задано функціями Чебишева або іншими спеціальними функціями, явний вигляд яких, як правило, істотно перевищує за складністю відповідні рекурентні форми.

Крок 3. Тепер ми маємо в своєму розпорядженні всі числові значення для обчислення коефіцієнтів *Bk* і $Ck_{,n}$, а отже і заповнення масиву *ESTAB*[*k, j*]. Якщо розглядати комірки масиву як накопичувачі відповідних сум, то завдання заповнення комірок зводиться до виконання трьох послідовно вкладених циклів. Зовнішній цикл визначає рядок масиву, перший внутрішній цикл визначає стовпець масиву, другий внутрішній цикл перераховує індекси незалежної змінної *x* і виконує обчислення елементів, що додаються, в суми B_k і $C_{k,n}$.

Крок 4. Остаточний розв'язок. Остаточне розв'язування, тобто розв'язування системи лінійних рівнянь (масив *ESTAB*), як правило, виконується відповідними бібліотечними процедурами. З позиції оптимальності, або найменшої кількості виконуваних операцій, слід обирати процедури, які виконують розв'язування

систем лінійних рівнянь за методом Гауса.

12.2.2. Рішення в інтегральному вигляді для неортогонального базису.

Розв'язання в інтегральному вигляді незначно відрізняється від алгоритму, який ми розглянули для дискретної форми.

Основна відмінність полягає в процедурах для обчислення коефіцієнтів системи рівнянь:

$$B_k = \int_{X_b}^{X_e} F(x) \cdot \gamma(x) \cdot v_k(x) dx;$$

$$C_{k,n} = \int_{X_b}^{X_e} \gamma(x) \cdot v_k(x) \cdot v_n(x) dx.$$

У цьому випадку, замість накопичення звичайної суми в комірках масиву *ESTAB*[*k, j*] (див. крок 3), виконуватиме накопичення суми, що складається з приватних квадратур, які обчислюються за тією чи іншою квадратурною формулою. Як квадратури, зазвичай розглядають квадратури, побудовані за ***методом трапецій*** або ***методом Сімпсона***. Особливістю таких квадратур є простота їх обчислення, особливо в разі рівномірного кроку для значень аргументу підінтегральних функцій.

Для початку розглянемо часткові квадратури, які обчислюються за методом трапецій, тобто, описують елементарні площі як трапеції, утворені двома сусідніми точками табличних функцій:

$$s_m = \frac{y_{m+1} + y_m}{2} (x_{m+1} - x_m).$$

У цьому випадку, повна квадратура (або значення інтеграла) *S* за рівномірного кроку аргументу може бути представлена сумою *S**, яка є доволі зручною для обчислення:

$$S = \left[\frac{y_M + y_0}{2} + \sum_{m=1}^{M-1} y_m\right] \cdot \Delta x = S^* \cdot \Delta x.$$

Приймаючи рівномірний крок для аргументу, як додаткову умову, ми можемо одержати ще більш істотні переваги як з погляду необхідного числа операцій, так, і це особливо важливо, з погляду точності обчислень.

Для обґрунтування цього твердження розглянемо деякий рядок системи лінійних рівнянь, що описують розв'язок задачі апроксимації. Нехай цей рядок має індекс - *k*.

$$\sum_{n=0}^{N} a_n \cdot C_{k,n} = B_k.$$

Якщо коефіцієнти *C* і *B* обчислені за квадратурними формулами з постійним кроком, то множник кроку можна скоротити, не порушивши рівність:

$$\sum_{n=0}^{N} a_n \cdot C^*_{k,n} \cdot \Delta x = B^*_k \cdot \Delta x$$

тобто

$$\sum_{n=0}^{N} a_n \cdot C^*_{k,n} = B^*_k.$$

Як важливий наслідок, усувається операція множення сторін рівності на мале число, яке, відповідно до переходу до інтегральної форми, має спрямовуватися до нульового значення. Таким чином, істотно знижуються невизначеності під час розв'язання системи лінійних рівнянь, а також усуваються зайві операції множення.

Використання квадратур Сімпсона (в обмін на вищу точність обчислення інтегралів) дещо збільшує складність обчислень, однак для рівномірного кроку аргументу, також дає змогу усунути a_n з процедур, які обчислюють коефіцієнти

системи рівнянь.

Як довідку, наведемо вираз для обчислення часткової квадратури за методом Сімпсона для рівномірного кроку аргументу:

$$s_m = (y_m + 4y_{m+1} + y_{m+2})\frac{(x_{m+1} - x_m)}{3};$$

$$s_m = (y_m + 4y_{m+1} + y_{m+2})\frac{\Delta x}{3}.$$

Розглядаючи шляхи підвищення якості апроксимації, ми вже зазначали, що нерівномірність кроку аргументу дає нам певні евристичні можливості, які втрачаються під час подання оцінки *D* в інтегральній формі, тобто, у формі площі. Це дає змогу сформулювати наведений нижче висновок.

Висновок. Вимога рівномірності кроку аргументу табличної функції для інтегральної форми задачі апроксимації жодним чином не посилює вже наявні обмеження та вимоги, однак дає змогу істотно скоротити обсяг обчислень.

12.2.3. Розв'язання в ортогональному базисі.

Нагадаємо коротко вигляд розв'язання задачі апроксимації в ортогональному базисі:

$$a_k \cong \frac{\int_{X_b}^{X_e} F(x) \cdot \gamma(x) \cdot v_k(x)dx}{\int_{X_b}^{X_e} \gamma(x) \cdot v_k(x) \cdot v_n(x)dx}.$$

Зазначимо важливу особливість. Як правило, визначений інтеграл у знаменнику наведеного виразу заздалегідь обчислюється як точна константа *C*:

$$\int_{X_b}^{X_e} \gamma(x) \cdot v_k(x) \cdot v_n(x)dx = \begin{cases} 0, k \neq n; \\ C, k = n. \end{cases}$$

Точність обчислення константи *C* забезпечується виконанням інтегрування в аналітичному вигляді з подальшою підстановкою меж інтегрування. Як наслідок,

ми отримуємо значення коефіцієнтів апроксимації в традиційному вигляді:

$$a_k \cong \frac{1}{C}\int_{X_b}^{X_e} F(x)\cdot\gamma(x)\cdot v_k(x)dx.$$

Спроби замінити цей інтеграл простою сумою (тобто, сформулювати задачу в суто дискретній формі, особливо для нерівномірного кроку за аргументом) вимагають перевірок виконання умов ортогональності для всіх співвідношень індексів базисних функцій. Як наслідок, такі спроби висувають особливі умови на вибір множини аргументів для табличних функцій і замість очікуваного зниження складності обчислень ми отримуємо результат прямо протилежний.

Висновок. Розглянута особливість рівнозначна вимозі обчислення інтеграла в чисельнику (інтеграла за участю табличної функції $F(x)$) тільки за допомогою квадратурних форм, оскільки константа C у знаменнику має розмірність площі.

Оскільки питання обчислення інтегралів за допомогою квадратурних формул досить добре розглянуті в різноманітній літературі, крім того, в попередньому розділі ми вже приділили увагу таким обчисленням, то питання складання алгоритмів такої задачі залишимо як питання вільного вибору.

12.3. Опис програми для апроксимації табличних функцій степеневими рядами та довільними рядами в системі MATLAB

Програма апроксимації табличних функцій степеневими та довільними рядами розроблена в системі MATLAB.

```
function [A,AbsErr]=LSM03Free(F,V,b1,b2,b3)
% МЕТОД НАЙМЕНШИХ КВАДРАТІВ ДЛЯ ВІЛЬНО ВИЗНАЧУВАНИХ
% БАЗИСНИХ ФУНКЦІЙ
% Синтаксис виклику:
% [A,AbsErr]=LSM03Free(F,V,b1,b2,b3)
% F - Вихідна матриця таблична функція вигляду:
% x1 x2 x3 x4 .... xM
% y1 y2 y3 y4 .... yM
```

```
% V - Літеральний вектор стовпець, що описує базисні
функції:
% v1
% v2
% v3
% ...
% vN
% Наприклад (підготовка вектора базисних функцій):
% Побудова списку базисних функцій (вектора V):
% v1='x.^0';
% v2='1-exp(-x./b2)';
% v3='sin(x.*b3)./(1+x.^2)';
% ....
% vN= '...';
% Збірка вектора базисних функцій
% V=char(v1, v2, v3, ... vN);
% Мінімальний вхідний контроль
if (size(F,1) ~= 2)
disp('Розміри матриці [F] задано неправильно');
A=NaN;
return
end;
nV=size(V,1);
if (size(F,2) < nV)
disp('Недостатньо точок у [F] для визначення коефіцієнтів
для [V]');
A=NaN;
return
end;
% Подготовка рабочих матриц (наперед сформированные матрицы
ускоряют алгоритм)
C=zeros(nV,nV);
B=zeros(nV,1);
% Побудуємо систему лінійних рівнянь для апроксимації
% Для всіх рядків матриці 'C'
for row=1:nV
% Для всіх 'x' матриці 'F'
B(row,1)=0;
for k=1:size(F,2)
x=F(1,k);
B(row,1)=B(row,1)+eval(V(row,:))*F(2,k);
end;
```

```
% Для всіх колонок матриці 'C'
for col=1:nV
% Для всіх 'x' матриці 'F'
C(row,col)=0;
for k=1:size(F,2)
x=F(1,k);
C(row,col)=C(row,col)+eval(V(row,:))*eval(V(col,:));
end;
end;
end;
% Обчислимо коефіцієнти для апроксимуючого виразу
ab=cat(2,C,B);
A = fgauss01(ab);
% Обчислимо значення функції за апроксимуючим виразом
% Для всіх 'x' матриці 'F'
for k=1:size(F,2)
x=F(1,k);
y(1,k)=0;
% Для всіх рядків матриці 'V'
for row=1:nV
y(1,k)=y(1,k)+A(row,1)*eval(V(row,:));
end;
end;
clf;
hold on;
disp('Графік вихідної табличної функції');
% Побудуємо графік вихідної табличної функції
plot(F(1,:),F(2,:),':b');
grid on;
disp('Далі...');
reply=input('Побудувати графік за результатами
апроксимації? y/n [y]: ','s');
if isempty(reply)
reply = 'y';
end;
if reply == 'y'
% Побудуємо графік за результатами апроксимації
plot(F(1,:),y,'-r');
grid on;
end;
disp('Далі...');
delta=F(2,:)-y;
```

```
AbsErr=max(abs(delta));
disp(strcat('Максимальна абсолютна похибка: ',
num2str(AbsErr)));
disp('Далее...');
reply=input(' Побудувати графік абсолютної помилки
апроксимації? y/n [y]:
','s');
hold off;
if isempty(reply)
reply = 'y';
end;
if reply == 'y'
% Побудуємо графік абсолютної похибки
if max(abs(delta)) > 1e-16
plot(F(1,:),delta);
grid on;
end;
end;
disp('Далі...');
reply=input('Для завершення програми натисніть Enter...');
close all hidden;
```

При таких вхідних даних:

```
% Встановити коефіцієнти при базисних функціях
a1=0.1;
a2=14;
a3=14;
% Встановити коефіцієнти всередині базисних функцій
b1=1;
b2=2;
b3=1;
% Визначити базисні функції
v1='x.^0';
v2='1-exp(-x./b2)';
v3='sin(x.*b3)./(1+x.^2)';
% Визначити діапазон аргументу
xb=0;
xe=40;
```

Такі вихідні дані можна представити графіком рис. 12.2:

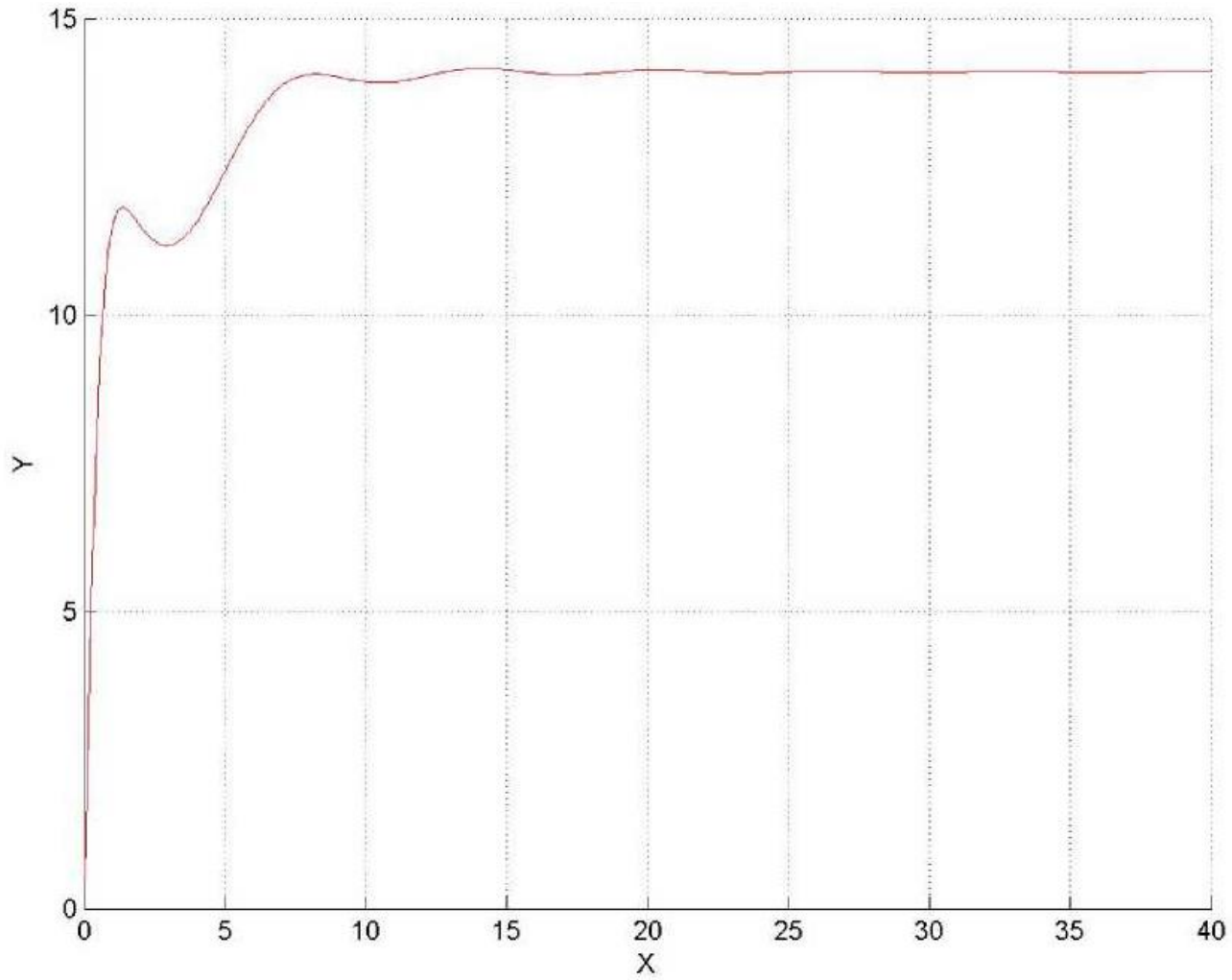


Рис. 12.2. Вихідні дані роботи.

У результаті апроксимації ми отримаємо такі результати:

Коефіцієнти членів ряду:

*A*1=9.999999999741845e-002

*A*2=1.400000000000263e+001

*A*3=1.400000000000453e+001

Максимальна абсолютна похибка = 2.5816e-012

Графік абсолютної помилки матиме такий вигляд, який наведено на рис. 12.3.

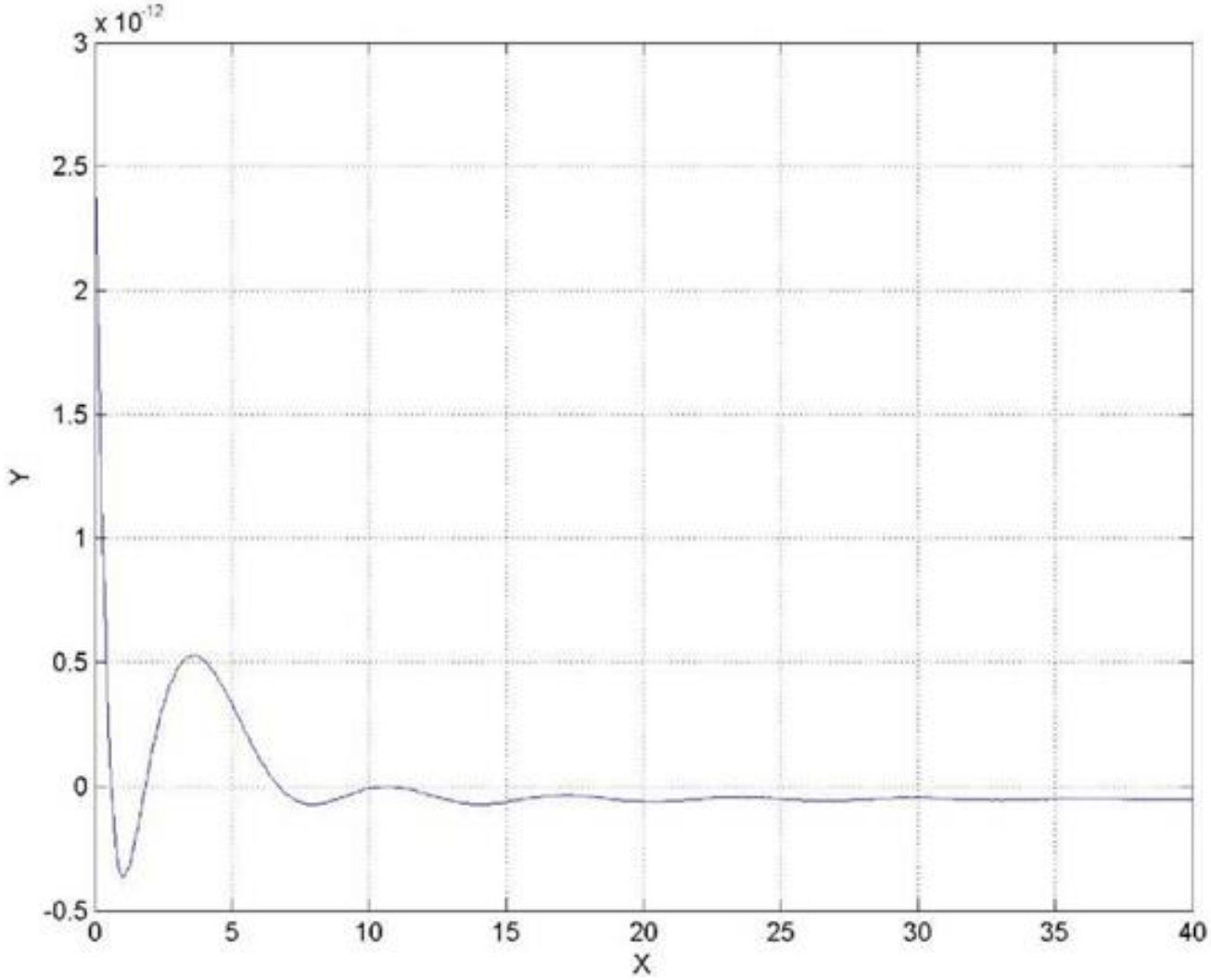


Рис. 12.3. Графік абсолютної помилки апроксимації.

12.4. Індивідуальні завдання

1. Реалізуйте програму з пункту 12.3 лабораторної роботи.

2. Виконайте з використанням наведеної програми апроксимацію таблично заданої функції з лабораторної роботи 10 методом найменших квадратів для вільно визначуваних базисних функцій. Для розв'язання системи лінійних рівнянь у даній функції для апроксимації використати метод Гауса, функція реалізації якого розглядалася в лабораторній роботі №11 (*fgauss*01).

3. Оформіть звіт з лабораторної роботи.

Лабораторна робота 13

АПРОКСИМАЦІЯ ФУНКЦІЙ СТЕПЕНЕВИМИ РЯДАМИ, ЩО СХОДЯТЬСЯ

Мета лабораторної роботи: здобуття та закріплення знань, формування практичних навичок роботи з програмами з апроксимації функцій степеневими рядами, що сходяться.

13.1. Короткі відомості з теорії

13.1.1. Ряди Тейлора - Маклорена (визначення)

Ряди Тейлора є степеневими рядами, які використовують для апроксимації різноманітних функцій, що в низці випадків значно спрощує їхній аналіз і перетворення таких функцій.

Традиційно ряд Тейлора визначають таким чином:

$$f(x) = \sum_{k=0}^{\infty} \frac{f^{(k)}(a)}{k!}(x-a)^k$$

при цьому передбачається, що функція $f(x)$ нескінченно диференційована в околиці точки a.

З математичної точки зору нескінченна довжина такого ряду не є перешкодою для його розгляду та подальших аналітичних перетворень. З точки зору реальних обчислень, доводиться обмежуватися деякою кінцевою довжиною ряду. Іншими словами, доводиться обмежувати верхній індекс ряд Тейлора деяким скінченним значенням N, що приводить ряд до такого вигляду:

$$f(x) = \sum_{k=0}^{N} \frac{f^{(k)}(a)}{k!}(x-a)^k + R_N,$$

де R_N називають залишковим членом, тобто, деяким числом, що відображає помилку представлення функції обмеженим рядом, а його значення оцінюють (як правило, у формі Лагранжа) таким чином:

$$R_N = \frac{f^{(N-1)}(\delta)}{(N+1)!}(x-a)^{N+1}, a < \delta < x.$$

При виконанні умови a = 0, коли всі похідні обчислюються в нульовій точці, ряд Тейлора набуває вигляду, відомого як ряд Маклорена:

$$f(x) = \sum_{k=0}^{N} \frac{f^{(k)}(0)}{k!}(x)^k + R_N.$$

13.1.2. Виведення формули для ряду Маклорена

Класичний вивід коефіцієнтів степеневого ряду у формі ряду Маклорена отримують шляхом аналітичного диференціювання степеневого ряду, записаного в загальному вигляді.

Нехай є можливим представити деяку функцію наступним степеневим рядом:

$$f(x) = b_0 + b_1x^1 + b_2x^2 + b_3x^3 + b_4x^4 + \cdots + b_Nx^N + R_N.$$

Оскільки степеневий ряд досить просто диференціювати, обчислимо всі його похідні до *N*-го порядку включно:

$$f^{(1)}(x) = b_1 + 2 \cdot b_2x^1 + 3 \cdot b_3x^2 + 4 \cdot b_4x^3 + \cdots + N \cdot b_Nx^{N-1};$$
$$f^{(2)}(x) = 2 \cdot b_2 + 3 \cdot 2 \cdot b_3x^1 + 4 \cdot 3 \cdot b_4x^2 + \cdots + N \cdot (N-1) \cdot b_Nx^{N-2};$$
$$f^{(3)}(x) = 3 \cdot 2 \cdot b_3 + 4 \cdot 3 \cdot 2 \cdot b_4x^1 + \cdots + N \cdot (N-1) \cdot (N-2) \cdot b_Nx^{N-3};$$
$$f^{(4)}(x) = 4 \cdot 3 \cdot 2 \cdot b_4 + \cdots + N \cdot (N-1) \cdot (N-2) \cdot (N-3) \cdot b_Nx^{N-4}.$$

Далі, застосовуючи індукцію, можна записати:

$$f^{(N)}(x) = N \cdot (N-1) \cdot (N-2) \cdot (N-3) \cdot \ldots \cdot (N-(N-1)) b_N x^{N-N}.$$

або

$$f^{(N)}(x) = N! \cdot b_N.$$

Обчислимо всі ці похідні в нульовій точці (тобто, при $x = 0$). У цьому випадку, всі члени суми, що містять x, окрім $x^0 = 1$, обнуляться і обчислені похідні набудуть вигляду:

$$f^{(1)}(0) = b_1;$$
$$f^{(2)}(0) = 2! \cdot b_2;$$
$$\ldots\ldots\ldots\ldots\ldots$$
$$f^{(N)}(0) = N! \cdot b_N.$$

Відповідно коефіцієнти b_k, можна легко визначити таким чином:

$$b_k = \frac{f^{(k)}(0)}{k!}, 0 \le k \le N,$$

що автоматично дає змогу перезаписати вихідний степеневий ряд у вигляді ряду Маклорена:

$$f(x) = \sum_{k=0}^{N} \frac{f^{(k)}(0)}{k!} (x)^k + R_N.$$

Аналогічним чином можна перевизначити степеневий ряд і у вигляді ряду Тейлора.

Як правило, багато функцій апроксимуються саме рядами Маклорена, що дещо спрощує обчислення похідних цих функцій і дає змогу отримувати компактніші форми запису рядів. Яскравим прикладом є апроксимація рядом Маклорена експоненціальної функції:

$$e^x = \sum_{k=0}^{N} \frac{1}{k!}(x)^k + R_N.$$

Дійсно, якщо

$$\frac{d}{dx}(e^x) = e^x,$$

а будь-яке число в нульовому ступені дорівнює одиниці, то похідні всіх порядків функції e^x обчислені в нулі, дорівнюватимуть одиницям і відповідний ряд набуде наведеного вище вигляду.

Однак, слід також зазначити функції, які для їх подання в компактній формі рядом Маклорена, додатково вимагають обчислення спеціальних чисел, таких як числа Бернуллі або числа Ейлера у формі подання таких чисел для рядів Тейлора. Так, наприклад, тангенс:

$$\operatorname{tg}(x) = x + \frac{x^3}{3} + \frac{2x^5}{15} + \frac{17x^7}{315} + \cdots = \sum_{k=1}^{\infty} \frac{B_{2k}(-4)^k(1-4^k)}{(2k)!} x^{2k-1} + R_N,$$

$$|x| < \frac{\pi}{2}, B_{2k} - \text{числа Бернуллі}$$

або секанс:

$$\sec(x) = \sum_{k=1}^{\infty} \frac{E_{2k}(-1)^k}{(2k)!} x^{2k} + R_N,$$

$$|x| < \frac{\pi}{2}, E_{2k} - \text{числа Ейлера.}$$

Ці спеціальні числа ми розглянемо дещо пізніше, а поки що зосередимося на питаннях можливості й точності представлення функцій за допомогою рядів Тейлора і Маклорена.

13.1.3. Ознаки збіжності рядів Тейлора - Маклорена

Початкова вимога, що формулюється під час виведення формул для рядів Тейлора і Маклорена, передбачає існування у функції похідних будь-яких порядків.

Як правило, це має на увазі, що їхні значення $f^{(k)}(a)$, є скінченними величинами.

Інакше:

- у разі принципової неможливості визначення однієї або кількох похідних, невизначеною буде і вся сума;
- у разі $x \neq 0$ і нескінченно великих значень (з одним знаком) однієї або декількох похідних уся сума буде нескінченно великою величиною;
- у разі $x \neq 0$ і знакозмінних нескінченних значень у кількох похідних, питання про скінченність усієї суми стає нетривіальним і потребує окремого аналізу.

Вимога скінченних значень усіх похідних, хоча і є необхідною умовою, проте не є умовою достатньою, оскільки нескінченна сума скінченних величин, також може призводити до нескінченно великих значень.

Таким чином, питання про можливість представлення функцій за допомогою рядів Тейлора і Маклорена, крім вимоги існування у функції скінченних похідних будь-яких порядків, повинне включати і вимогу про обов'язкову збіжність цих рядів.

Питання збіжності числових рядів досліджувалися багатьма математиками. У результаті досліджень, було сформульовано досить багато критеріїв і ознак для аналізу збіжності рядів. Найбільш загальним є критерій Коші збіжності числового ряду. На основі цього критерію були доведені ознаки збіжності рядів, які переважно використовуються для перевірки збіжності рядів Тейлора та Маклорена.

а) Ознака Даламбера, яка формулюється таким чином:

Якщо не дорівнюють нулю знакопозитивні члени p_k деякого ряду

$$\sum_{k=0}^{\infty} p_k$$

строго задовольняють умові

$$D_{k+1} = \frac{p_{k+1}}{p_k} < 1$$

для всіх k, то цей ряд сходиться.

б) Ознака Даламбера в граничній формі, яка формулюється таким чином:

Якщо для не рівних нулю знакопозитивних членів p_k деякого ряду

$$\sum_{k=0}^{\infty} p_k$$

строго виконується

$$\lim_{k\to\infty} \frac{p_{k+1}}{p_k} = q < 1,$$

то даний ряд сходиться.

в) Ознака Даламбера була узагальнена Лейбніцем для знакочередуючих рядів, тобто:

Якщо знакочередуючі члени

$$(-1)^k p_k,$$

деякого ряду

$$\sum_{k=0}^{\infty} (-1)^k p_k.$$

Задовільняють умові

$$0 < p_{k+1} < p_k$$

і виконується

$$\lim_{k\to\infty}(p_k) = 0,$$

то даний ряд зходиться.

З названих вище ознак, найчастіше, для перевірки збіжності рядів, використовується ознака Даламбера в граничній формі, або її узагальнення, виконане Лейбніцем. Оскільки названі ознаки оперують зі знакопозитивними членами ряду, то для аналізу збіжності ряду зручно застосовувати абсолютні величини цих членів.

13.1.4. Оцінка залишкових членів рядів Тейлора - Маклорена

Тепер розглянемо оцінки для точності подання деякої функції рядом Тейлора або Маклорена. Для цього запишемо ряд Тейлора в наступному вигляді:

$$f(x) - \sum_{k=0}^{N} \frac{f^{(k)}(0)}{k!}(x)^k = R_N.$$

У цьому випадку, залишковий член R_N можна розглядати як помилку апроксимації. Для оцінки таких помилок отримано кілька оціночних форм, а саме:
форма Пеано

$$R_N = o((x-a)^N), \text{при } x \to a;$$

форма Коші

$$R_N = \frac{f^{(N+1)}\big(a + \theta_1(x-a)\big)}{N!}(1-\theta_1)^N(x-a)^{N+1}; 0 < \theta_1 < 1;$$

форма Лагранжа

$$R_N = \frac{f^{(N+1)}(\delta)}{(N+1)!}(x-a)^{N+1}; a < \delta < x;$$

Найчастіше, для оцінки залишкового члена в рядах Тейлора або Маклорена використовується форма Лагранжа. Запишемо цю оцінку у вигляді нерівності, що дає змогу оцінити максимальну абсолютну помилку:

$$|R_N| \leq \frac{\left|f^{(N+1)}(\delta)\right|}{(N+1)!}|(x-a)^{N+1}|, a < \delta < x.$$

У цьому разі необхідно знайти таке δ, за якого значення похідної буде максимальним.

Як приклад розглянемо функцію $f(x) = e^x$, при $x = 1$. Вибір аргументу, що дорівнює одиниці, дає змогу розглядати приклад, як задачу про визначення довжини ряду Тейлора для обчислення значення e (або підстави натурального логарифма) із заданою абсолютною помилкою.

Отже, нехай $a = 0$, тоді відповідна нерівність у цьому прикладі матиме вигляд:

$$|R_N| \leq \frac{\left|f^{(N+1)}(\delta)\right|}{(N+1)!}, 0 < \delta < x.$$

Оскільки

$$f(x) = e^x \rightarrow f^{(k)}(x) = e^x,$$

або будь-яка похідна цієї функції є монотонно зростаючою функцією, то її максимальне значення визначатиметься в точці $\delta = x$. В таблиці 13.1 для різної довжини ряду показано:

- індекс останнього члена ряду (колонка 1);
- значення факторіала для останнього члена ряду (колонка 2);
- відповідно до довжини ряду обчислене значення ряду (колонка 3);
- в колонках 4 і 5 показано значення залишкових членів для граничних випадків вибору точки δ.

Таблиця 13.1.

N	$N!$	$e^x = \sum_{k=0}^{N} \frac{1}{k!}(x)^k$	$\lvert R_N \rvert;\quad \delta = x$	$\lvert R_N \rvert;\quad \delta = 0$
1	2	3	4	5
0	1,0000000000000000E+00	1,00000000000000000	2,718281828	1
1	1,0000000000000000E+00	2,00000000000000000	1,359140914	0,5
2	2,0000000000000000E+00	2,50000000000000000	0,453046971	0,166666667
3	6,0000000000000000E+00	2,66666666666667000	0,113261743	0,041666667
4	2,4000000000000000E+01	2,70833333333333000	0,022652349	0,008333333
5	1,2000000000000000E+02	2,71666666666667000	0,003775391	0,001388889
6	7,2000000000000000E+02	2,71805555555556000	0,000539342	0,000198413
7	5,0400000000000000E+03	2,71825396825397000	6,74177E-05	2,48016E-05
8	4,0320000000000000E+04	2,71827876984127000	7,49086E-06	2,75573E-06
9	3,6288000000000000E+05	2,71828152557319000	7,49086E-07	2,75573E-07
10	3,6288000000000000E+06	2,71828180114638000	6,80987E-08	2,50521E-08
11	3,9916800000000000E+07	2,71828182619849000	5,67489E-09	2,08768E-09
12	4,7900160000000000E+08	2,71828182828617000	4,3653E-10	1,6059E-10
13	6,2270208000000000E+09	2,71828182844676000	3,11807E-11	1,14707E-11
14	8,7178291200000000E+10	2,71828182845823000	2,07871E-12	7,64716E-13
15	1,3076743680000000E+12	2,71828182845899000	1,2992E-13	4,77948E-14
16	2,0922789888000000E+13	2,71828182845904000	7,64233E-15	2,81146E-15
17	3,5568742809600000E+14	**2,71828182845905000**	4,24574E-16	1,56192E-16
18	6,4023737057280000E+15	**2,71828182845905000**	2,2346E-17	8,22064E-18
19	1,2164510040832000E+17	**2,71828182845905000**	1,1173E-18	4,11032E-19
20	2,4329020081766400E+18	**2,71828182845905000**	5,32048E-20	1,95729E-20
21	5,1090942171709400E+19	**2,71828182845905000**	2,4184E-21	8,89679E-22
22	1,1240007277776100E+21	**2,71828182845905000**		

Більш точне значення для основи натурального логарифма відоме як: e = 2,7182818284590452353602874713 52…

13.2. Обчислення рядів Тейлора - Маклорена в кінцевій розрядній сітці

Починаючи з індексу 17 (табл. 13.1., колонка 1) значення ряду (табл. 13.1., колонка 2) залишаються незмінними і являють собою округлене до 15 десяткових знаків значення основи натурального логарифма. Таке округлення виникає під час підсумовування відмінних на багато порядків дійсних чисел у системах програмування з обмеженою розрядною сіткою.

Це досить просто пояснити таким чином: якщо ступінь першого знака мантиси дійсного числа в першому операнді суми дорівнює або більший на величину довжини розрядної сітки, ніж ступінь першого знака мантиси другого

операнда, то значення другого операнда можна використовувати або як ознаку округлення, або воно просто зникає за межами розрядної сітки.

Примітка: сформульована властивість суворо застосовується тільки з урахуванням конкретної позиційної системи, застосовуваної для подання дійсних чисел. Якщо використовується кілька систем подання чисел (як, наприклад, двійкова і десяткова в комп'ютерах), то оцінка робиться в тій системі, в якій виконуються операції з числами.

Ряди Тейлора-Маклорена, які сходяться, під час їх обчислення в арифметиці з плаваючою крапкою, при досягненні деякої довжини ряду N можуть припинити асимптотичне наближення до обчислюваного значення функції. Причиною цього є вихід значень чергового ненульового члена ряду p_k +1 за межі значущих величин поточної суми ряду S_k, якщо ця сума обчислюється в кінцевій розрядній сітці.

Як наслідок, в арифметиці з плаваючою крапкою, можна сформулювати ознаку "межі точності":

якщо під час обчислення деякого значення безперервно збіжного ряду, що сходиться $|D_{k+1}| > 0$ умови:

$$|p_{k+1}| > 0,\ S_k - S_{k+1} = 0$$

виконуються один або більше разів при збільшенні k, то це означає, що досягнуто межі точності обчислення цього значення в кінцевій розрядній сітці.

Перевірка такої ознаки в алгоритмах реалізується досить просто, однак при цьому може суттєво прискорити їхнє виконання.

13.2.1. Приклад програми обчислення ряду Маклорена

```
// Обчислити степеневий ряд, що сходиться, для конкретного
значення //X. Функція повертає результат обчислення Sum,
індекс ряду //HInd при якому підсумкова сума в кінцевій
розрядній сітці //припиняє змінюватися, а також і ознака
безаварійного //завернення - Result.
function CalcSeriesForX (
```

```
RqCoeffArr : TD1AttExt; // Масив коефіцієнтів ряду
RqX : extended; // Аргумент
var HInd : integer; // Підсумковий індекс кінця ряду
var Sum : extended // Підсумкова сума ряду
) : boolean;
var Ind : integer; // Робочий індекс
PwX : extended; // X у поточному ступені
PSum : extended; // Попереднє значення суми
CtIPL : integer; // Лічильник ознаки межі точності
begin
Result := False; PwX := 1; HInd := 0; Sum := 0; CtIPL := 0;
if Length(RqCoeffArr) < 1 then Exit;
try
for Ind := Low(RqCoeffArr) to High(RqCoeffArr)
do begin
PSum := Sum; HInd := Ind;
if (Abs(RqCoeffArr[Ind]) > 0) and (Abs(PwX) > 0)
then begin // Пропуск нульових членів ряду
Sum := Sum + RqCoeffArr[Ind]* PwX;
if Abs(PSum - Sum) = 0 // Пастка ознаки межі точності
then begin
Inc(CtIPL); if CtIPL > 1 then Break;
end else CtIPL := 0;
end;
PwX := PwX * RqX;
end;
Result := True;
except
Beep; MessageDlg('Помилка обчислення елементів для' + #13#10
+ 'члена ряду з індексом: ' + IntToStr(Ind),
mtError, [mbOk], 0);
end;
end;
```

У системі програмування *Delphi*, під час використання представлення чисел за допомогою типу *extended*, розглянута особливість підсумовування проявляє себе, коли відмінність для перших десяткових знаків операндів досягає значень приблизно 18 - 19 порядків. На рис. 13.1, показано як змінюється індекс максимального ступеня ряду (вісь *Y*) для ряду Маклорена функції $f(x) = e^x$ при

досягненні граничної абсолютної похибки для різних значень аргументу функції x (вісь X) на проміжку $-1 < x < 4{,}1717175$.

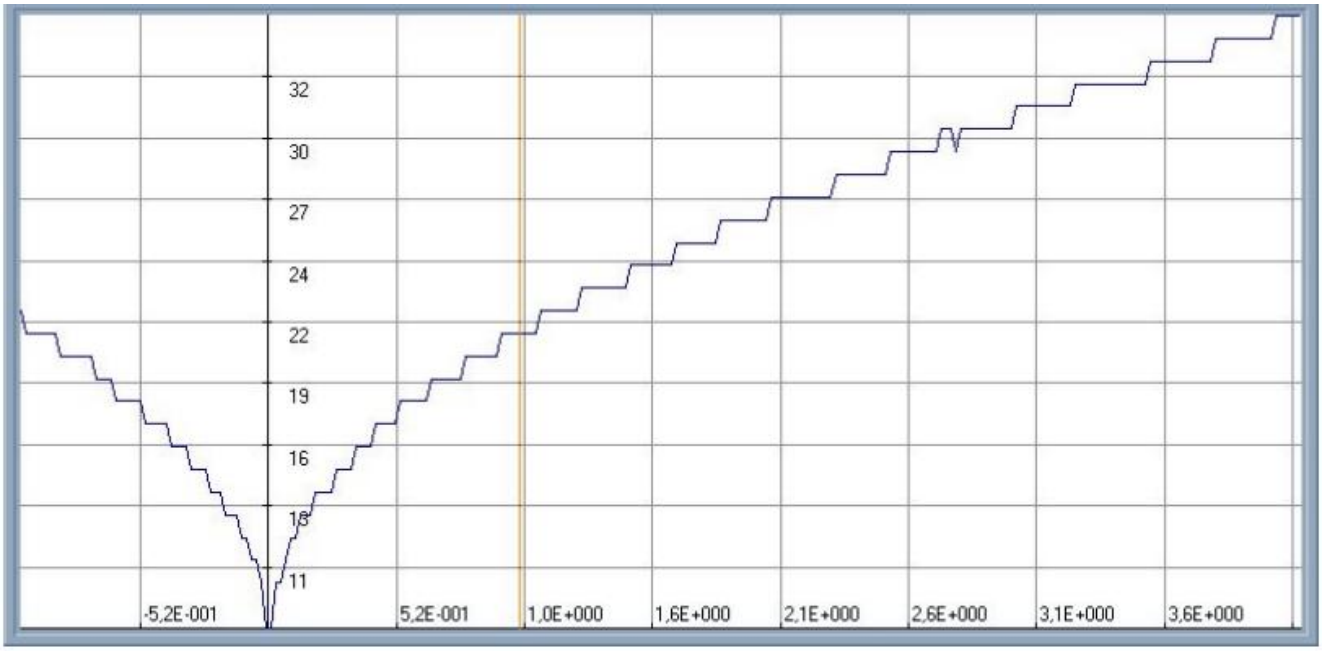


Рис. 13.1. Зміна індексу максимального ступеня для ряду Маклорена

Як видно з цього чисельного експерименту, під час обчислення підстави натурального алгоритму в плаваючій арифметиці *Delphi*, починаючи з довжини рядка Маклорена, що дорівнює 21, значення наступних членів рядка виходять за межі розрядної сітки суми, а отже, алгоритм може бути зупинено. На рис. 13.2 показано різницю обчислення функції $f(x) = e^x$ рядом Маклорена і стандартною процедурою бібліотеки *Delphi*.

13.2.2. Способи підвищення точності під час обчислення функцій рядами Тейлора і Маклорена

Таких способів існує кілька, проте найбільш традиційним із них є використання властивостей самоподібності функцій і подання функцій із розбиттям області їх визначення на ділянки.

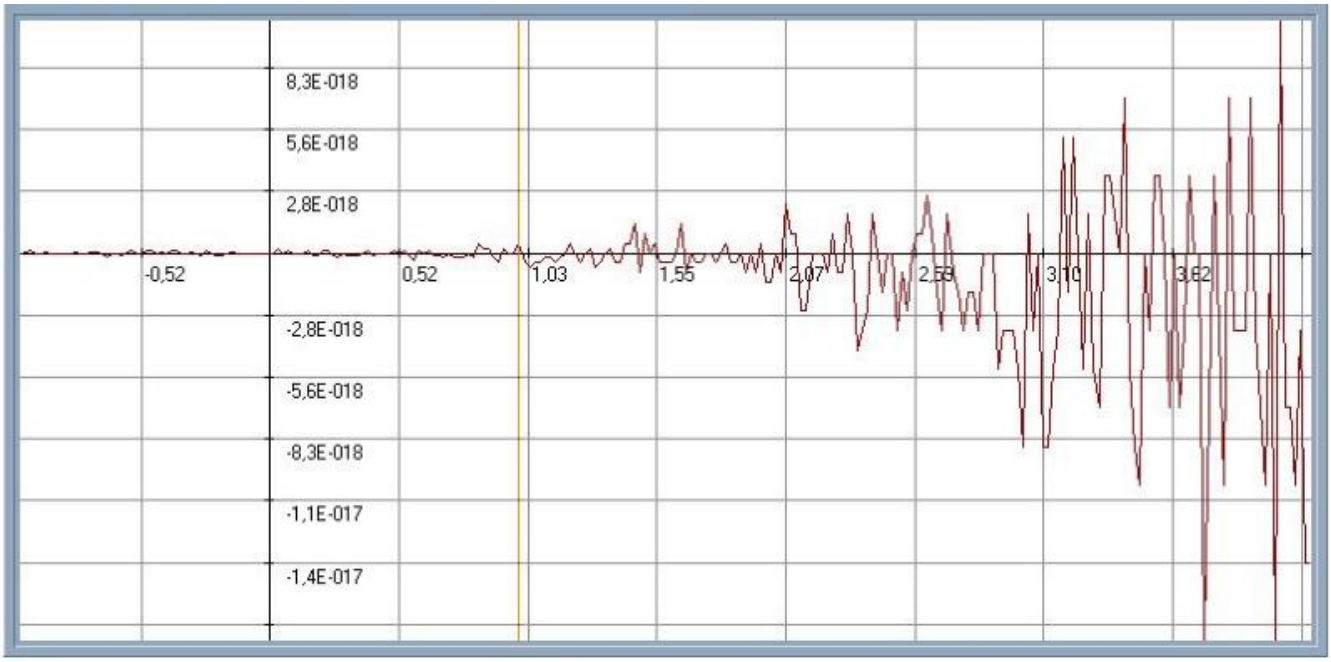


Рис. 13.2. Різниця обчисляння.

Використання періодичної самоподібності функцій

Більшість тригонометричних функцій повторюють свої значення на будь-яких інтервалах кратних їхньому періоду *T*. Таку властивість тригонометричних функцій можна використати для перенесення великих значень аргументу функції, де ряди Маклорена починають помітно відхиляться від відповідних значень функції, в область її першого періоду. Оскільки похідні функції для рядів Маклорена обчислюються при $x = 0$, то в області близькій до нульових значень аргументу $| x | < T$ ряди Маклорена дозволяють обчислити значення функції з найкращою точністю. У цьому випадку, додаткові алгоритмічні витрати на таке перенесення аргументу є більш ніж виправданими. Алгоритм цього перенесення надзвичайно простий:

Якщо $| x | > T$, то послідовно виконуючи $| x_{k+1} | = | x_k | - T$ доб'ємося виконання умови $0 < | x_{k+1} | < T$ і, тим самим, отримаємо шукану величину x_T

У тривіальному виконанні алгоритму це найпростіший цикл. При більш уважному погляді на алгоритм, легко помітити, що його результатом є залишок від ділення величини x на величину *T*. Якщо ми маємо в своєму розпорядженні деяку

функцію *Frac*, яка дає змогу отримати дробову частину дійсного числа (*fractional part of a real number*), то такий залишок можна отримати в такий спосіб:

$$x_T = T \cdot \mathrm{Frac}\left(\frac{x}{T}\right).$$

У більшості систем програмування функція *Frac* входить до складу бібліотеки *Math* стандартних математичних процедур.

Необхідно також зауважити, що крім розглянутої самоподібності на періоді в окремих тригонометричних функцій (наприклад *sin, cos*) спостерігається ще низка додаткових симетрій. Використання таких симетрій дає змогу помітно скоротити необхідну область визначення аргументів для обчислення відповідного ряду Маклорена. Наприклад:

$$\sin(x)|_{0 \le x \le \pi} = -\sin(x)|_{\pi \le x \le 2\pi}.$$

Як наслідок, якщо для досягнення ознаки межі точності функції *sin* на межах періоду потрібен ряд Маклорена, у якого ступінь аргументу в останньому члені дорівнює ~63, то на межах напівперіоду необхідний ступінь знижується до значення ~37, що значно прискорює обчислення ряду.

У системі програмування *Delphi* таке прискорення можна реалізувати таким чином:

```
// SeriesSin(X) - обчислення sin рядом Маклорена
// QuickSeriesSin – швидке обчислення sin рядом Маклорена
//
function QuickSeriesSin (X : extended) : extended;
begin
Result := 2 * Pi * Frac(X /(2 * Pi)); // Змістимо аргумент у
перший період
if Abs(Result) <= Pi
then Result := SeriesSin(Result)
else begin // Використовуємо симетрію напівперіодів
Result := Abs(Result) – Pi; // Змістимо аргумент у перший
напівперіод
```

```
if X >= 0
then Result := - SeriesSin(Result)
else Result := SeriesSin(Result);
end;
end;
```

Використання самоподібності як результату афінних перетворень

Афінне перетворення є відображенням принципу незалежності властивостей об'єкта від вибору точки спостереження і математично реалізується за допомогою операцій лінійного зсуву точки початку координат, лінійного масштабування, а також векторного обертання. Формально, розглянуту вище періодичну самоподібність тригонометричних функцій можна віднести до операції афінного зсуву.

У цьому підрозділі, на прикладі функції b^x, розглянемо самоподібність, що утворюється в результаті операцій афінного зсуву та масштабування. Отже, для будь-якої показникової функції, включно з експонентою, справедливо наступне:

$$f(x) = b^x = b^{a+(x-a)} = b^a \cdot b^{x-a};$$
$$f(x) = b^x = b^a \cdot b^\delta, \qquad \delta = x - a,$$

де точку (a) можна розглядати як точку нового початку координат незалежної змінної δ, а множник b^a як масштабний множник. Однак, важливо зазначити таке: якщо $a = 0$ то $b^x = b^\delta$.

Отже, показову функцію можна подати як періодичну, якщо на кожному періоді (ділянці) з індексом k її аргумент змінюватиметься в межах інтервалу $0 < \delta < a$, а для отримання кінцевого результату b^x на конкретному періоді функції застосувати афінні операції, тобто:

$$0 \le \delta = x - a_k \le a, \;\; a_k = k \cdot a, \;\; k \in [0, K];$$
$$b^x = b^{a_k} b^\delta.$$

Інакше кажучи, нами показано самоподібність потенційної функції на різних ділянках її аргументу, досягнуту внаслідок застосування афінних перетворень до значень функції на її початковій ділянці k=0. З погляду найпростішого алгоритму, який реалізує обчислення функції із застосуванням ділянок, це означає необхідність використання масиву попередньо обчислених констант для операцій масштабування на цих ділянках (періодах).

Наявність такого масиву або іншого способу одержати масштабувальні константи b^{a_k} кожної ділянки є природною ціною за скорочений діапазон аргументів, у якому обчислюють названу функцію, і може обґрунтовуватися підвищенням швидкості й точності обчислення, особливо під час обчислення функцій низками, що сходяться.

З позиції застосування операцій зсуву і масштабування досить цікаво подивитися на ряд Тейлора:

$$f(x) = \sum_{k=0}^{N} \frac{f^{(k)}(a)}{k!}(x-a)^k + R_N.$$

Легко помітити, що цей ряд можна інтерпретувати як суму функцій x^k, початок координат яких зсунуто в точку $x = a$, а як масштаби використовуються константи $f^{(k)}(a)/k!$. Такий погляд дозволяє ще раз підкреслити фундаментальність властивостей афінного перетворення.

Як уже згадувалося раніше, до складу операцій афінного перетворення входить також операція векторного обертання точок деякого об'єкта, відносно довільно обраної точки, що іменується центром обертання. Оскільки питання застосування цієї операції істотно відводять нас від розгляду рядів Тейлора і Маклорена для дійсного аргументу, обмежимося лише ілюстрацією такого підходу в графічному вигляді.

Отже, якщо на деякій ділянці функція $f(x)$ та її апроксимація рядом $f^*(x)$ мають подібну кривизну, як це показано на рисунку (a), то обертання на кут у

кожній точці функції $f^*(x)$ відносно центру обертання x_0 і подальше зрушення $x_0 + \Delta x$ (для повернення точок в область ділянки), дає змогу помітно зменшити абсолютну помилку апроксимації. На рис. 13.3 (б) показано функцію $f^{**}(x)$ як кінцевий результат застосування операцій обертання і подальшого зсуву.

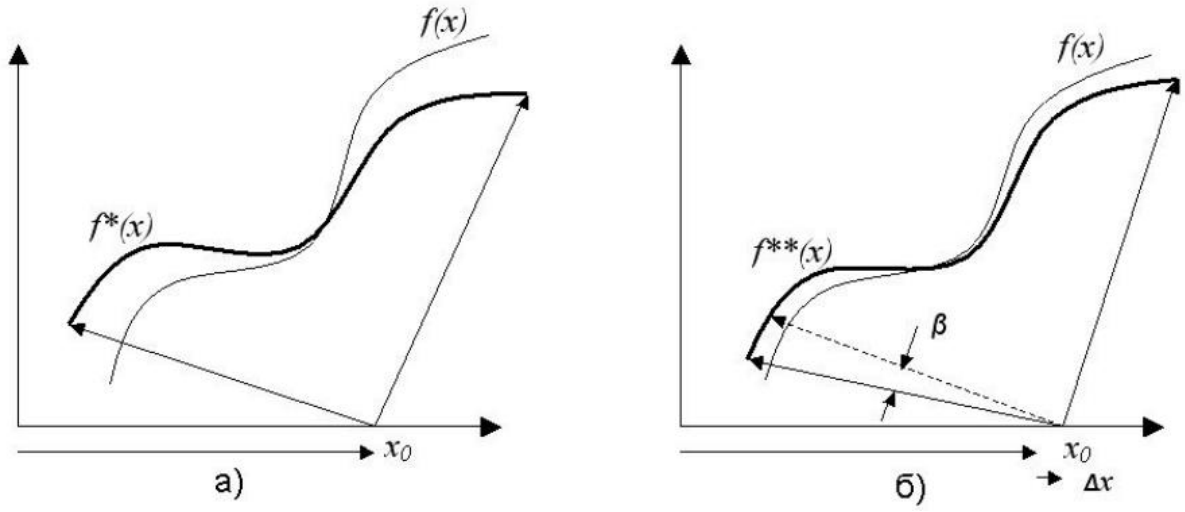


Рис. 13.3. Функція як кінцевий результат застосування операцій обертання і подальшого зрушення

В інженерній практиці повний набір афінних операцій зазвичай застосовують у тих випадках, коли, маючи у своєму розпорядженні ряд Маклорена з уже обчисленими коефіцієнтами, з різних причин важко або неможливо обчислити значення похідних у точках, відмінних від нуля. Однак у разі коли похідні всіх порядків можуть бути обчислені в будь-якій точці розглянутої області визначення, як правило, для підвищення точності область визначення функції розбивається на ділянки і для кожної ділянки обчислюється відповідний ряд Тейлора. Можливість представляти функції окремими рядами на ділянках обраної довжини дає змогу не тільки підвищити точність апроксимації таких функцій, а й підвищити швидкість алгоритмів за рахунок скорочення довжини рядів. При цьому слід пам'ятати, що доводиться в той чи інший спосіб збільшувати кількість збережених або обчислюваних значень, які описують параметри рядів на цих ділянках.

13.3. Спеціальні числа для рядів Тейлора-Маклорена

13.3.1. Числа Бернуллі

Історія спеціальних чисел багато в чому зумовлена завданням обчислення суми послідовних натуральних чисел, піднесених в один і той самий ступінь:

$$S_{K-1} = \sum_{k=1}^{K-1} k^n.$$

Така сума була обчислена Якобом Бернуллі в такій формі:

$$S_{K-1} = \sum_{k=1}^{K-1} k^n = \frac{1}{n+1} \sum_{i=0}^{n} C_{n+1}^{i} \cdot B_i \cdot K^{n+1-i}.$$

Наведену суму записано в досить компактному вигляді, що виявилося можливим завдяки низці спеціальних чисел, а саме - біноміальних коефіцієнтів C^k_n, а також чисел Бернуллі B_k. Ці спеціальні числа визначаються таким чином:

Біноміальні коефіцієнти - це послідовність раціональних чисел, наприклад:

$$C_0^K, C_1^K, C_2^K, \dots, C_K^K$$

яка обчислюється за допомогою факторіалів:

$$C_n^k = \frac{n!}{k!\,(n-k)!}.$$

Числа Бернуллі — це послідовність раціональних чисел

$$B_0, B_1, B_2, \dots, B_K,$$

яка може обчислюватися рекурентною формулою:

$$B_0 = 1,\ \ B_k = \frac{-1}{k+1} \cdot \sum_{n=1}^{k} C_{n+1}^{k+1} \cdot B_{k-n}.$$

Крім того, всі числа Бернуллі з непарним індексом дорівнюють нулю.

У деяких випадках проводиться переіндексація послідовності для того, щоб опустити непарні номери чисел Бернуллі з нульовим значенням (B_{2n}). Зазвичай такі числа Бернуллі застосовуються при розкладанні в ряд Тейлора функцій *tan*(*x*) і *tanh*(*x*).

13.3.2. Числа Ейлера для рядів Тейлора

У теорії чисел, числа Ейлера – це послідовність En цілих чисел, які визначаються в результаті розкладання в ряд Тейлора виразу:

$$\frac{1}{\cosh(t)} = \frac{2}{e^t + e^{-t}} = \sum\nolimits_{n=0}^{\infty} \frac{E_n}{n!} t^n,$$

де cosh(*t*) гіперболічний косинус. У загальному випадку, числа Ейлера є спеціальними значеннями многочленів Ейлера. Усі непарні числа Ейлера дорівнюють нулю. У деяких випадках проводиться переіндексація послідовності для того, щоб опустити непарні номери чисел Ейлера з нульовим значенням (E_{2n}). Зазвичай такі числа Ейлера застосовують при розкладанні в ряд Тейлора функцій *sec(x)* і *sech(x)*. Вони також виходять у комбінаториці, зокрема, під час підрахунку кількості перестановок, що чергуються, на множині з парним числом елементів.

Явна формула для чисел Ейлера:

$$E_{2n} = i \sum_{k=1}^{2n+1} \sum_{j=0}^{k} \frac{(-1)^j (k-2j)^{2n+1}}{2^k i^k k},$$

де *i* позначає уявну одиницю $i^2 = -1$.

У програмних додатках, необхідність обчислення чисел Ейлера досить часто виникає спільно з необхідністю обчислювати числа Бернуллі. У цьому випадку для обчислення чисел Ейлера зазвичай використовується формула такого вигляду:

$$E_k = \sum_{n=1}^{k} C_{n-1}^{k} \frac{(2^n - 4^n)}{n} B_n\,,\ \ E_0 = 1, E_1 = 0, k \geq 2.$$

де C_{n-1}^{k} відповідні біноміальні коефіцієнти, а B_n – відповідні числа Бернуллі.

13.3.3. Приклад програм для обчислення чисел Бернуллі та Ейлера

```
// -----------------------------------------------------
// Тип для одновимірного динамічного масиву значень значень
спеціальних чисел
type TD1AttExt = array of extended;
// Факторіал
function Fact(RqInd : integer) : extended;
var Ind : word;
begin
Result := 1;
if RqInd <= 0 then Exit;
for Ind := 1 to RqInd do Result := Result * Ind;
end;
// Біноміальний коефіцієнт
function Ckn(k, // Верхній індекс
n : integer // Нижній індекс
) : extended;
begin
Result := Fact(k) / Fact(n);
Result := Result / Fact(k-n);
end;
// Побудувати масив біноміальних коефіцієнтів з мінливим
індексом - n
// при заданому верхньому індексі (RqK)
procedure MakeCknArr(RqK : integer; var Arr : TD1AttExt);
var n : integer;
begin
SetLength(Arr, RqK + 1);
for n := 0 to RqK do Arr[n] := Ckn(RqK, n);end;
```

```
// Побудувати масив Arr з числами Бернуллі з максимальним індексом RqK
procedure MakeBernoulliArr(RqK : integer; var Arr : TD1AttExt);
var n, k, i : integer;
begin
SetLength(Arr, RqK + 1);
Arr[0] := 1;
for k := 1 to RqK
do begin
Arr[k] := 0;
for n := 1 to k
do Arr[k] := Arr[k] + (-1 /(k + 1)) * Ckn(k + 1, n + 1)* Arr[k - n];
end;
for i := 2 to RqK
do if (i mod 2) <> 0 then Arr[i] := 0;
end;
// Побудувати динамічний масив Arr з числами Ейлера в кількості RqK
procedure MakeTEulerArr(RqK : integer; var Arr : TD1AttExt);
var k, n : integer;
Item : extended;
BArr : TD1AttExt;
begin
// Побудувати масив Arr з числами Бернуллі
MakeBernoulliArr(RqK, BArr);
// Підготувати масив чисел Ейлера
SetLength(Arr, RqK + 1);
Arr[0] := 1; Arr[1] := 0;
for k := 2 to RqK
do begin
Arr[k] := 0;
if BArr[k] <> 0
then begin
for n := 1 to k
do begin
Item := Ckn(k, n-1) * BArr[n] * (IntPower(2, n) - IntPower(4, n)) /n;
Arr[k] := Arr[k] + Item;
end; end;
end; end;
```

Для ясного сприйняття, вихідні тексти програм наведено в максимально спрощеному вигляді. З тесту виключено елементи вхідного контролю, елементи перехоплення виняткових ситуацій тощо. Але навіть у такому вигляді процедури безаварійно дозволяють будувати таблиці до 1024 чисел.

```
k = 1019          0
k = 1020          -7,1251404225847E1813
k = 1021          0
k = 1022          1,883260203116 77E1818
k = 1023          0
k = 1024          -4,99719368710874E1822
```

Чи означає це високу точність обчислення спеціальних чисел? Для відповіді на дане запитання необхідно ще раз підкреслити, що в багатьох мовах кількість символів (десяткових знаків) для подання дійсних чисел у бінарній формі обмежена.

Раніше (під час розгляду рядів, що сходяться) ми вже зазначали, що для члена ряду ступінь першої значущої цифри якого був меншим за ступінь останньої значущої цифри в поточній сумі, спостерігалася втрата значення такого члена ряду в підсумковій сумі. При обчисленні спеціальних чисел має місце дзеркальна картина. Інакше кажучи, якщо член суми такий, що ступінь його останньої цифри більший, ніж ступінь останньої цифри в сумі (представленої максимальною кількістю знаків), то відбувається витіснення останньої цифри підсумкової суми за межі розрядної сітки. Наприклад, розглянемо результат обчислення чисел Бернуллі алгоритмами з необмеженою розрядною сіткою (таблиця 13.1). Легко помітити, що починаючи десь із $B(36)$ або $B(40)$ чисельник дробу (у випадку *Delphi*) повинен автоматично округлятися в середньому до 18 значущих цифр.

Як наслідок, спроби необмежено збільшувати довжину ряду Маклорена для підвищення точності апроксимації в системах програмування зі скінченною розрядною сіткою можуть призвести до прямо протилежного результату. Наприклад, для функції *tan(x),* у якій застосовуються числа Бернуллі, мала

швидкість збіжності вимагає збільшення довжини ряду, а втрата точності коефіцієнтів (за умови скінченної розрядної сітки) вимагає зменшення довжини цього ряду. Пошук оптимального балансу між цими вимогами безумовно цікавий з наукового погляду, проте далеко виходить за межі інженерних задач.

Таблиця 13.2. Числа Бернуллі, обчислені алгоритмами з необмеженою розрядною сіткою

B(0) = 1 / 1
B(1) = –1 / 2
B(2) = 1 / 6
B(4) = –1 / 30
B(6) = 1 / 42
B(8) = –1 / 30
B(10) = 5 / 66
B(12) = –691 / 2730
B(14) = 7 / 6
B(16) = –3617 / 510
B(18) = 43867 / 798
B(20) = –174611 / 330
B(22) = 854513 / 138
B(24) = –236364091 / 2730
B(26) = 8553103 / 6
B(28) = –23749461029 / 870
B(30) = 8615841276005 / 14322
B(32) = –7709321041217 / 510
B(34) = 2577687858367 / 6
B(36) = –26315271553053477373 / 1919190
B(38) = 2929993913841559 / 6
B(40) = –261082718496449122051 / 13530
B(42) = 1520097643918070802691 / 1806
B(44) = –27833269579301024235023 / 690
B(46) = 596451111593912163277961 / 282
B(48) = –560940336899781768624912754 7 / 46410
B(50) = 495057205241079648212477525 / 66

Таким чином, як висновок, можна сформулювати потребу в пошуку альтернативних методів для апроксимації функцій з повільною збіжністю при їхньому поданні рядами Тейлора або Маклорена.

13.4. Альтернативні апроксимації для різних функцій

Використання степеневих рядів, як основної форми для апроксимації функцій, зумовлене тим, що степеневий ряд описується лише за допомогою елементарних арифметичних операцій: підсумовування, віднімання, множення та ділення. Однак на базі цих операцій можна побудувати й інші регулярні або ітераційні структури.

Розглянемо деякі, найбільш популярні структури регулярного або ітераційного типу.

13.4.1. Функції як відношення двох степеневих рядів

Відношення двох степеневих рядів зазвичай записується в такій формі:

$$f(x) = \frac{\sum_{n=0}^{\infty} a_n x^n}{\sum_{m=0}^{\infty} a_m x^m}.$$

Хорошим прикладом такого відношення є відношення рядів, що представляють відповідно функції *sin*(*x*) і *cos*(*x*), що прямим чином застосовується для обчислення *tan*(*x*) у бібліотеці *Math* системи *Delphi*.

$$\tan(x) = \frac{\sin(x)}{\cos(x)}.$$

Ряди *sin*(*x*) і *cos*(*x*) досить добре сходяться і, за відносно невеликої довжини (~60-70) членів, забезпечують високу точність. Це дає змогу з високою точністю отримати функції *tan*(*x*) і *cotan*(*x*) у широкому діапазоні їхнього аргументу. Нагадаємо, що для функцій *tan*(*x*) і *cotan*(*x*) звичайний ряд Маклорена збігається

дуже повільно, а в ділянці, близькій до точок розриву, теоретично вимагає нескінченної кількості членів, що за кінцевої розрядної сітки призводить до серйозних проблем.

Також корисно зазначити, що якщо вдається знайти дійсні або комплексні корені степеневих рядів (x_n і x_m), то згідно з формулами Вієта відношення двох степеневих рядів можна представити в такому вигляді:

$$f(x) = \frac{\sum_{n=0}^{N} a_n x^n}{\sum_{m=0}^{M} a_m x^m} = \frac{\prod_{n=0}^{N}(x - x_n)}{\prod_{m=0}^{M}(x - x_m)}.$$

В інженерній практиці така форма використовується для представлення різних передавальних функцій.

13.4.2. Обчислення функцій за допомогою ланцюгових дробів

Одній і тій самій функції можна зіставити різні моделі, вибір яких залежить від розв'язуваної задачі. Для широкого класу функцій з погляду можливості отримання їхніх значень із наперед заданою точністю за найменшу кількість арифметичних дій (за найменший машинний час) найкращими моделями є відповідні дроби ланцюгових дробів.

час) найкращими моделями є відповідні дроби ланцюгових дробів.

Ланцюговим (безперервним) дробом, називається вираз виду:

$$b_0 + \cfrac{a_1}{b_1 + \cfrac{a_2}{b_2 + \cdots}}.$$

Через громіздкість запису ланцюговий дріб зазвичай записується в таких формах:

$$b_0 + \frac{a_1}{b_1 + \cdots} \quad \frac{a_2}{b_2 + \cdots} \ldots \frac{a_n}{b_n + \cdots} \ \ldots, b_0 + K_{n=1}^{\infty}\left(\frac{a_n}{b_n}\right), b_0 + \Phi\left(\frac{a_n}{b_n}\right).$$

Для дійсного аргументу реалізація в *Delphi* такого ланцюгового дробу матиме вигляд:

```
// Ітераційний алгоритм обчислення тангенса ланцюговим дробом
function tgx(X : extended) : extended;
var n, i : integer;
s, g : extended;
begin
n := 16; // Загальне число ланок дробу
g := 0;
Result := g;
if X = 0 then Exit;
s := X * X / (2 * n -1); // Остання ланка ланцюга
for i := n -1 downto 1
do begin
g := X * X / (2 * i - 1 - s); // Чергова ланка ланцюга
s := g;
end;
Result := g / X; // Поправити ступінь першої ланки
end;
```

Як видно з вихідного тексту, довжина дробу становить лише 16 ланок. При цьому в діапазоні від 0 до 89,990 градуса абсолютна помилка апроксимації функції *tan*(*x*) не гірше ніж 2e-12. На рис. 13.4, показано графік *tan*(*x*) поблизу точки розриву (90 градусів).

Абсолютна помилка (рис. 13.5) обчислювалась відносно бібліотечної функції *Math.Tan*(*x*).

У рамках цього прикладу корисним також зазначити, що за значеннями *tan*(*x*) можна обчислити *sin*(*x*) і *cos*(*x*) використовуючи формули:

$$\sin(x) = \frac{2\tan(\frac{x}{2})}{1+\tan^2(\frac{x}{2})};\ \cos(x) = \frac{1-\tan^2(\frac{x}{2})}{1+\tan^2(\frac{x}{2})}.$$

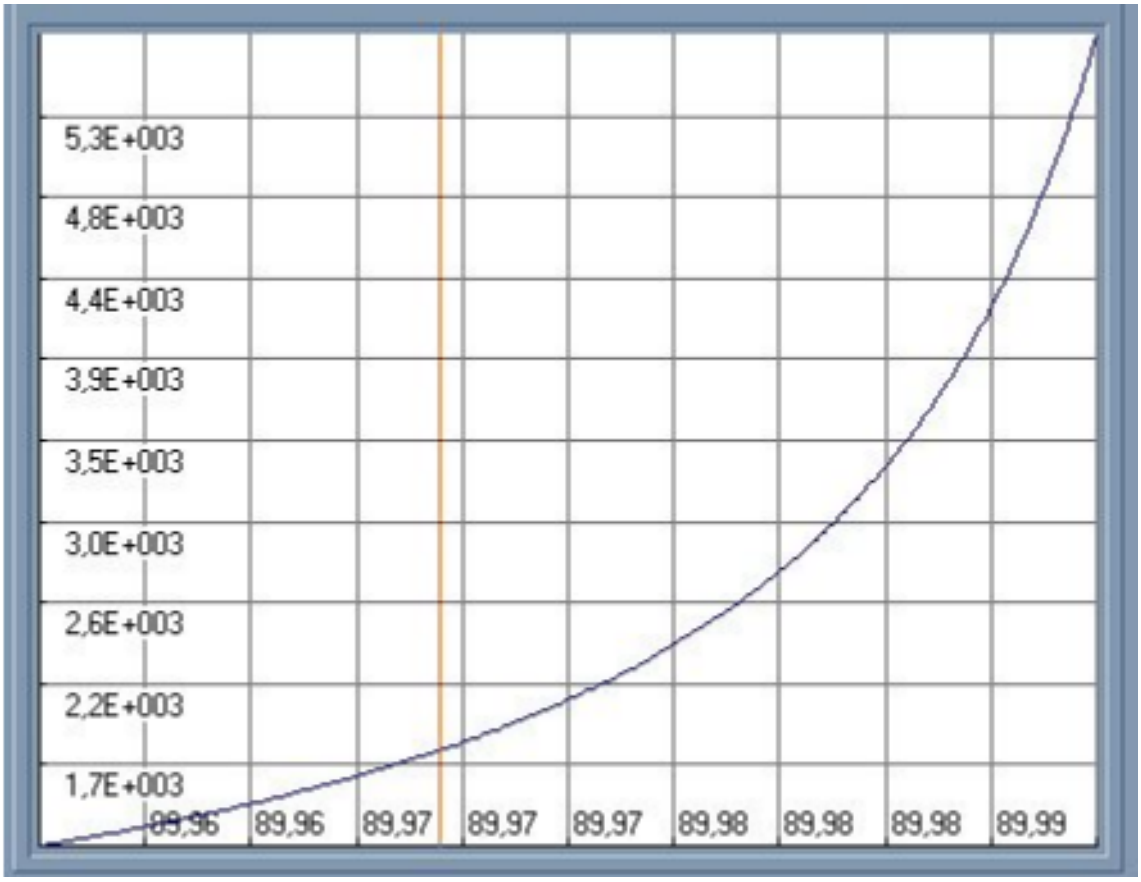


Рис. 13.4. Графік *tan*(*x*).

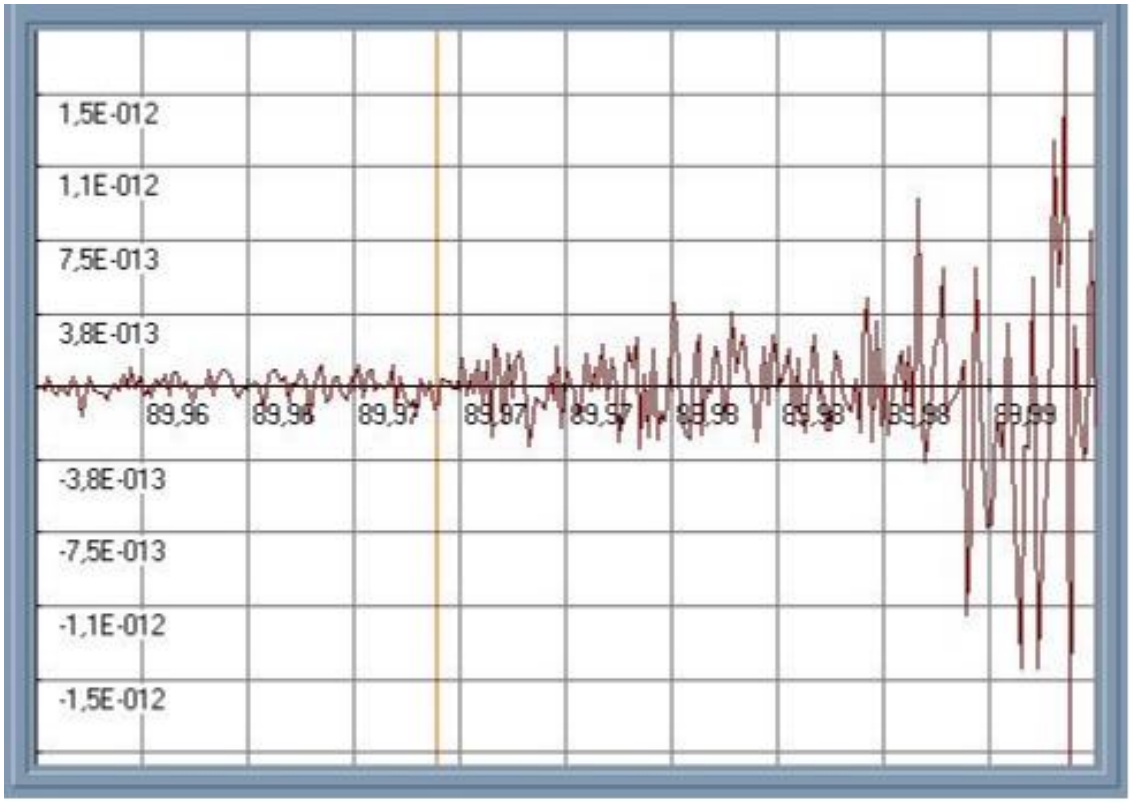


Рис. 13.5. Абсолютна помилка.

Обчислення *tan(x)* за допомогою ланцюгового дробу, за схожих характеристик точності, помітно швидше за метод, розглянутий у 13.4.1. Така особливість дає змогу рекомендувати цей метод для систем, які працюють у режимі реального часу.

13.4.3. Обчислення функцій за допомогою геометричних перетворень

Обчислення функцій за допомогою геометричних перетворень є одним із різновидів ітераційних методів. Особливістю цього методу є перетворення числових характеристик геометричних фігур для обчислення відповідних функцій, якими пов'язані ці числові характеристики.

Як приклад, розглянемо обчислення коренів різного ступеня від дійсного, додатного аргументу.

Для початку, сформулюємо логіку методу для обчислення квадратного кореня.

Квадратний корінь. Візьмемо прямокутник зі сторонами a=1 і b= X. Площа цього прямокутника дорівнює $S = a * b = 1* X = X$. Перетворивши прямокутник на квадрат так, що його площа залишиться тією самою, отримаємо довжину сторони, що дорівнює кореню квадратному з площі фігури, тобто зі значення x.

Перетворення прямокутника на квадрат будемо виконувати за такими ітераційними формулами

$$a_{n+1} = \frac{(a_n + b_n)}{2};$$

$$b_{n+1} = \frac{S}{a_{n+1}}.$$

При цьому, кожні наступні значення a і b будуть усереднюватися, наближаючись до рівності $a = b$, однак площа фігури залишатиметься незмінною. Логіка цього методу програмним кодом реалізується таким чином:

```
// Ітераційний алгоритм обчислення кореня квадратного
// методом перетворення прямокутника на квадрат
function sqrt_ab(X : extended) : extended;
const EPS = 1e-18; // Необхідна відносна похибка
var s, a, b : extended;
begin
Result := 0;
if X < EPS then Exit;
s := Abs(X); a := 1; b := s;
while Abs(1 - b/a) > EPS
do begin
a := (a+b)/2; // Чергове середнє значення сторін
b := s/a; // Значення b для збереження незмінної площі
end;
Result := (a+b)/2;
end;
```

Число ітерацій (для досягнення абсолютної помилки 1*e*-18) у середньому діапазоні аргументу *X* (від 0,9999e-14 до 0,9999e14) становить приблизно 25 - 30 ітерацій. Таким чином, ми отримуємо високоточний і швидкий алгоритм, придатний для застосування в системах реального часу.

Логіку методу обчислення квадратного кореня досить просто поширити на методи обчислення кубічного кореня і кореня довільного цілого ступеня. У цьому випадку логіку можна сформулювати таким чином:

Корінь кубічний. Візьмемо прямокутну піраміду зі сторонами *a*=1, *a*=1 і *b*= *X*. Об'єм цієї піраміди дорівнює $V = a * a * b = 1*1* X = X$. Перетворивши піраміду на куб так, щоб його об'єм залишиться колишнім, отримаємо довжину сторони куба, що дорівнює кореню кубічному з об'єму фігури.

Корінь N-ступеня. Візьмемо прямокутну гіперпіраміду з (*N*-1) ребрами *a*=1 і ребром *b*=*X*. Гіпероб'єм цієї піраміди дорівнює $V = (N\text{-}1) * a * b = 1 * X = X$. Перетворивши піраміду на гіперкуб так, що його гіпероб'єм залишиться тим самим, отримаємо довжину ребра гіперкуба, що дорівнює кореню *N*-ступеня з гіпероб'єму цієї фігури.

Відповідні методам вихідні тексти мають такий вигляд:

```
// Ітераційний алгоритм обчислення кубічного кореня
function Root3_ab(X : extended; var Count : integer) :
extended;
const EPS = 1e-18; // Необхідна відносна похибка
var a, b : extended;
begin
Count := 0;
Result := 0;
if Abs(X) < 0 then Exit;
a := 1; b := Abs(X);
while (Abs(1-b/a) > EPS)
do begin
a := (2*a + b)/3; // Чергове середнє значення ребер
b := X/a; b := b/a; // Значення b для збереження незмінного
обсягу
Inc(Count);
end;
Result := (2*a + b)/3;
end;
// Ітераційний алгоритм обчислення кореня ступеня N
function RootN_ab(X : extended; N : word; var Count :
integer) : extended;
const EPS = 1e-18; // Необхідна відносна похибка
var a, b : extended;
i : integer;
begin
Count := 0;
Result := 0;
if (Abs(X) = 0) or (N < 2) then Exit;
a := 1; b := Abs(X);
while Abs(1 - b/a) > EPS
do begin
a := ((N-1)*a + b)/N; // Чергове середнє значення ребер
// Значення b для збереження незмінного гіпероб'єму
b := Abs(X);
for i:=1 to (N-1) do b := b/a;
Inc(Count);
end;
Result := ((N-1)*a + b)/N;
end;
```

Параметр функцій *Count* використовується для отримання кількості ітерацій, за яких досягається задана абсолютна помилка *EPS*. Знак аргументу *X* ігнорується, тобто, корені в будь-якому разі обчислюються для додатних значень *X*.

На рис. 13.6, який подано нижче, показано динаміку кількості ітерацій для кореня п'ятого ступеня в діапазоні аргументів від 0 до 0,999e14 для досягнення відносної помилки 1e-18.

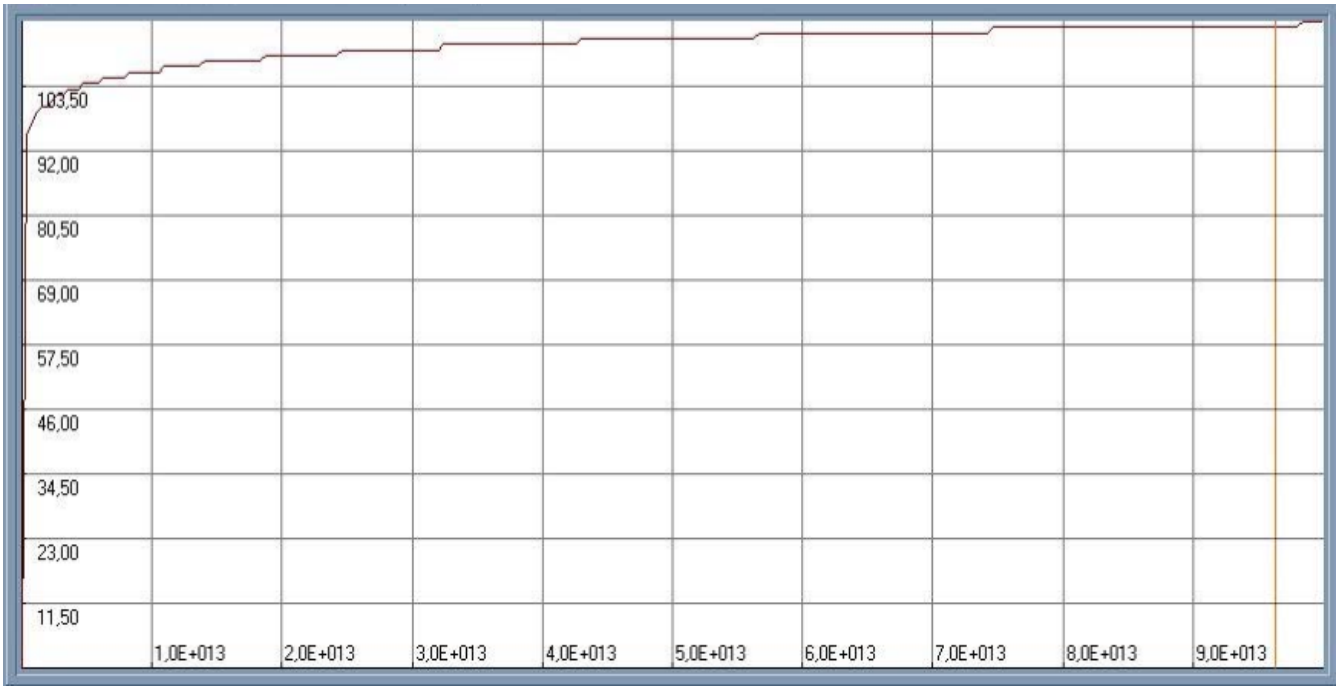


Рис. 13.6. Динаміка кількості ітерацій

В алгоритмах обчислення коренів замість оцінки точності у вигляді абсолютної помилки застосовується оцінка за відносною помилкою. Це зумовлено широким діапазоном аргументу, для якого умова виходу з циклу за абсолютною помилкою за великих значень *X* виявляється недосяжною через кінцеву розрядну сітку.

13.5. Індивідуальні завдання

1. Для проведення чисельних експериментів з апроксимації функцій степеневими рядами, що сходяться, використовувати програму TaylorSeries.

2. Відповідно до номера N за списком у журналі групи, записаному у вигляді $N = CM$, де C – старша цифра, M – молодша цифра, виберіть дві функції для апроксимації з табл. 13.3. і спеціальні числа для рядів Тейлора - Маклорена з табл 13.4.

3. За допомогою програми *TaylorSeries* у різних діапазонах зміни інтервалу в радіанах і градусах побудуйте графіки:

- обраних функцій;
- абсолютної похибки відносно *Math*-функції;
- довжини ряду для досягнення мінімальної похибки.

Таблиця 13.2

M	**Функції для апроксимації**	
1	sqrt(*x*+1), |*x*| < 1	arctan(*x*), |*x*| <= 1
2	1/(1-*x*), |*x*| < 1	sinh(*x*)
3	exp(*x*)	cosh(*x*)
4	ln(*x*+1), |*x*| < 1	tanh(*x*), |*x*|< pi/2
5	ln(1-*x*), |*x*| < 1	arcsinh(*x*), |*x*| <= 1
6	sin(*x*)	arctanh(*x*), |*x*| < 1
7	cos(*x*)	sin(*x*)/*x*
8	tan(*x*), |*x*|< pi/2	si(*x*)
9	sec(*x*), |*x*|< pi/2	wo(*x*)
0	arcsin(*x*), |*x*| <= 1	exp(*x*)

Таблиця 13.3

C	**Спеціальні числа для рядів Тейлора - Маклорена**
0	Числа Бернуллі
1	Числа Ейлера

4. Для різних значень максимальної довжини ряду *Max* (*k*) за допомогою програми *TaylorSeries* обчислити біноміальні коефіцієнти та значення спеціальних чисел для рядів Тейлора-Маклорена.

5. За допомогою програми *TaylorSeries* проведіть дослідження збіжності ряду обраних функцій. Побудуйте графіки ступенів ряду та ознак збіжності ряду (ознака Даламбера). Змінюючи значення довжини ряду *Max.Ind* і *X*-аргумента прослідкуйте і зробіть висновок як змінюється значення функції *F*(*X*) і максимальна ознака Даламбера.

6. Оформіть звіт із лабораторної роботи.

Лабораторна робота 14

КВАДРАТУРНІ МЕТОДИ ДЛЯ ОБЧИСЛЕННЯ ІНТЕГРАЛІВ ТАБЛИЧНИХ ФУНКЦІЙ З ПОСТІЙНИМ КРОКОМ ЗА АРГУМЕНТУ

Мета лабораторної роботи: здобуття та закріплення знань, формування практичних навичок роботи з програмами під час обчислення інтегралів табличних функцій з постійним кроком за аргументом.

14.1. Короткі відомості з теорії

Для обчислення інтегралів таблично заданих функцій застосовують квадратурні методи, до числа яких належать найпопулярніші методи лівих і правих прямокутників, трапецій, а також квадратурний метод Сімпсона.

Для початку згадаємо, що інтегралом деякої функції $F(X)$ у межах від X_{beg} до X_{end} є значення площі S, яка обмежена віссю X і самою функцією $F(X)$ на заданій ділянці.

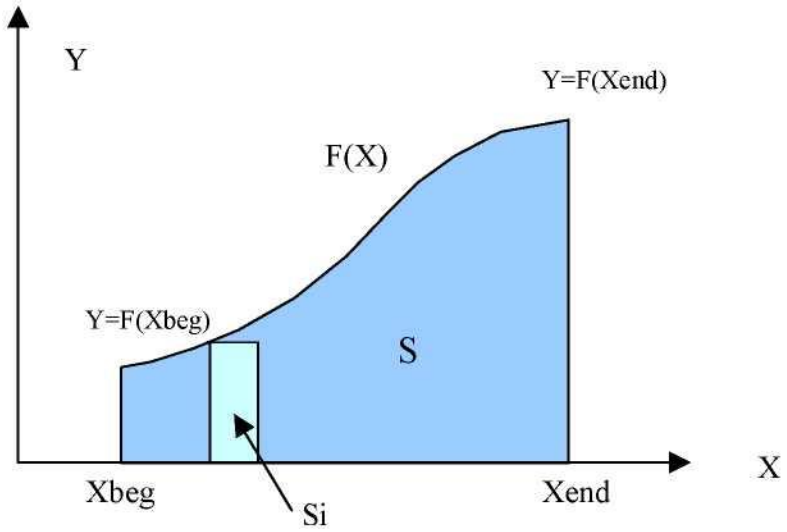


Рис. 14.1. Площа під кривою $F(X)$

Таке значення інтеграла безпосередньо випливає з його математичного визначення:

$$\int_{X_{beg}}^{X_{end}} F(X)dx = S = \lim_{\Delta X \to 0} \sum_{i=0}^{n=(X_{end}-X_{beg})/\Delta X} S_i =$$

$$= \lim_{\Delta X \to 0} \sum_{i=0}^{n=(X_{end}-X_{beg})/\Delta X} F(X_i)\Delta X,$$

де *Si* це площі елементарних прямокутників, які щільним чином розміщуються всередині площі *S*, повністю її заповнюючи. Площі таких елементарних прямокутників легко обчислюються як розмір їхньої основи *F*(*Xi*) на їхню висоту Δ*X*. У разі граничного переходу (коли висота елементарних прямокутників прямує до нуля) площа, що обчислюється інтегральною сумою *S*, нескінченно близько наближається до значення інтеграла, чим і визначає його сенс.

У науковій або технічній діяльності функції *F*(*X*), як правило, спочатку стають відомими як набори результатів вимірювань. Зауважимо, що при прагненні висоти елементарних прямокутників до нуля, їхня кількість n прагне до нескінченності. Очевидно, що практично неможливо вимагати нескінченне число значень (відліків *Y*) у таких наборах. Таким чином, виникає завдання обчислення інтегральної суми *S* на обмеженій кількості ділянок в умовах, коли аналітичний вигляд функції *F*(*X*) нам ще невідомий і застосування аналітичного інтегрування неможливе. Таке завдання обчислення інтегралів належить до класу завдань, що отримали найменування "чисельні методи математики».

Таким чином, вимушено обмежуючись скінченним числом *Si* у складі суми *S*, ми отримуватимемо тільки наближені значення обчислювальних інтегралів, причому точність обчислення буде тим вищою, чим більше число елементарних площ Si ми можемо собі дозволити.

Отже, безпосередньо з описаного вище способу обчислення інтегральних сум *S* визначаються два класичні квадратурні методи чисельного обчислення інтегралів. Це методи лівих і правих прямокутників. Відмінність між цими методами полягає тільки в тому, яке з найближчих значень $F(X_i)$ використовується для обчислення елементарних площ *Si*. Розглянемо ці квадратурні методи.

14.1.1. Квадратурний метод лівих прямокутників

Площу елементарного прямокутника Si під кривою обчислюють як площу прямокутника, основа якого визначається за лівим значенням аргументу функції на ділянці.

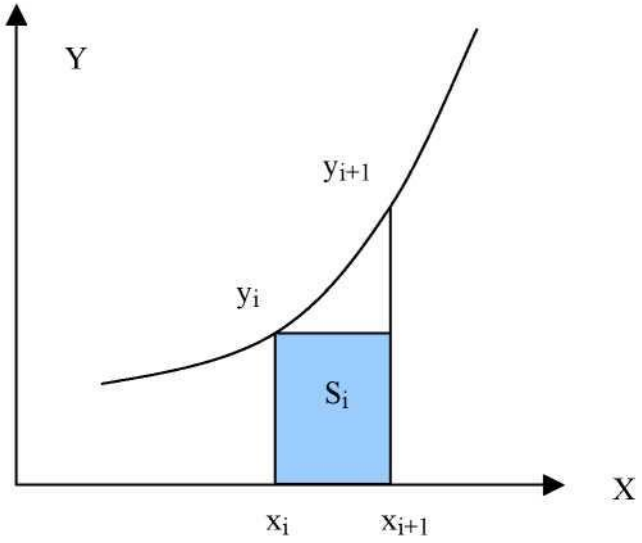


Рис. 14.2. Елементарний прямокутник з основою за лівим відліком $F(X_i)$

Для простоти обмежимося умовою, коли висоти елементарних прямокутників ($X_{i+1} – Xi$) рівні між собою.

Таке обмеження пояснюється тим, що для зручності подальшого опрацювання найрізноманітнішими математичними методами переважну більшість технічних вимірів і технічних характеристик виконують або подають із постійним кроком за аргументом.

Приймаючи таке обмеження, досить просто обчислити значення для довільного *Si*:

$$S_i = y_i(x_{i+1} - x_i) = y_i h; \quad x_{i+1} - x_i = const = h.$$

Нехай індексація ділянок (або індекс і) починається з нуля. Якщо в нашому розпорядженні *n*-ділянок, то значення функції в точці $F(X_{end})$ вже не використовуватиметься, оскільки воно визначить *Si* вже за межами границь інтегрування. Таким чином, інтегральна сума *S* матиме вигляд:

$$S = \sum_{i=0}^{i=n-1} S_i = \sum_{i=0}^{i=n-1} y_i \cdot h$$

де висота кожного елементарного прямокутника дорівнює:

$$h = \left(\frac{X_{end} - X_{beg}}{n}\right).$$

Для прикладу, наведемо графік зміни абсолютної помилки *AE* під час обчислення інтеграла методом лівих прямокутників:

$$AE(X) = S(X) - \int_{X_{beg}}^{X_{end}} X^2 dx.$$

Зазначимо, що точне значення інтеграла (як функції від *X*) і відповідно площа *S*(*X*) змінюються в міру того, як аргумент *X* з кроком *h* зростає в межах від $X_{beg} = 0$ до $X_{end}=1$. Загальне число ділянок (або *n*) дорівнює 63:

14.1.2. Квадратурний метод правих прямокутників

Метод правих прямокутників у певному сенсі є дзеркальним щодо методу лівих прямокутників. Іншими словами, площу елементарного прямокутника S_i обчислюють як площу прямокутника, основа якого визначається за ***правим*** значенням функції на ділянці.

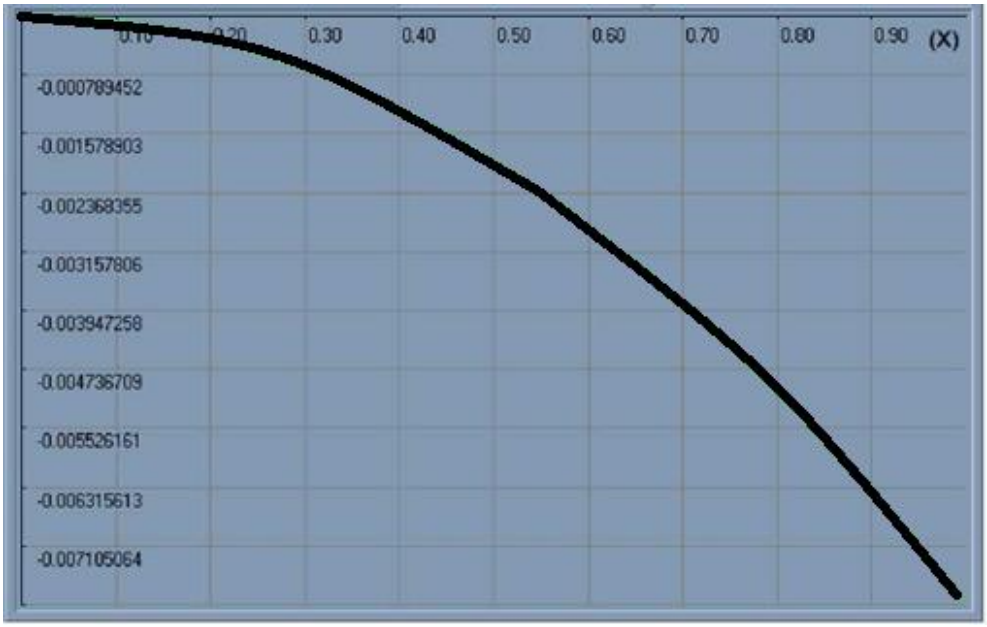


Рис. 14.3. Графік абсолютної помилки обчислення інтеграла як функції його верхньої межі методом лівих прямокутників

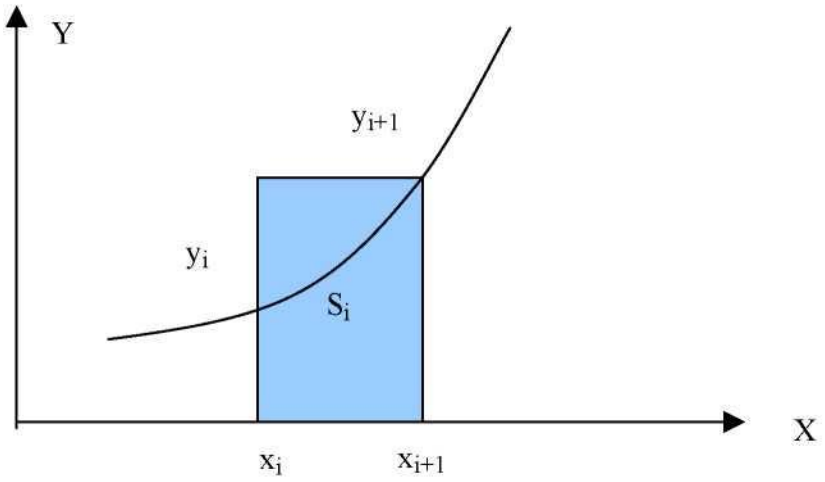


Рис. 14.4. Елементарний прямокутник з основою за правим відліком $F(X_{i+1})$

Приймаючи аналогічне обмеження:

$$S_i = y_{i+1}(x_{i+1} - x_i) = y_{i+1}h; \quad x_{i+1} - x_i = const = h.$$

А, також враховуючи, що основа елементарного прямокутника пов'язана з правим відліком функції, підсумкова інтегральна сума матиме вигляд:

$$S = \sum_{i=1}^{i=n} S_i = \sum_{i=1}^{i=n} y_i \cdot h$$

де висота кожного елементарного прямокутника дорівнює діапазону, поділеному на число ділянок n:

$$h = \left(\frac{X_{end} - X_{beg}}{n}\right).$$

За аналогією з методом лівих прямокутників наведемо графік зміни абсолютної помилки AE під час обчислення інтеграла методом правих прямокутників:

$$AE(X) = S(X) - \int_{X_{beg}}^{X_{end}} X^2 dx.$$

Зазначимо, що точне значення інтеграла (як функції від X) і відповідно площа $S(X)$, змінюються в міру того, як аргумент X з кроком h зростає в межах від $X_{\text{beg}} = 0$ до $X_{\text{end}}=1$. Загальна кількість ділянок (або n) дорівнює 63:

Порівнюючи помилки методів лівих і правих прямокутників (графіки 3 і 5), можна звернути увагу на їхній майже дзеркальний характер відносно осі X. Зазначена симетрія не є абсолютно точною, проте дає змогу зробити певні припущення.

Суть цих припущень приводить до думки про можливість ці помилки усереднити. Тобто, обчислюючи елементарну площу S_i на одній і тій самій ділянці, як результат взяти середнє між її значенням за методом лівих і правих прямокутників. Такий підхід приводить нас до методу трапецій.

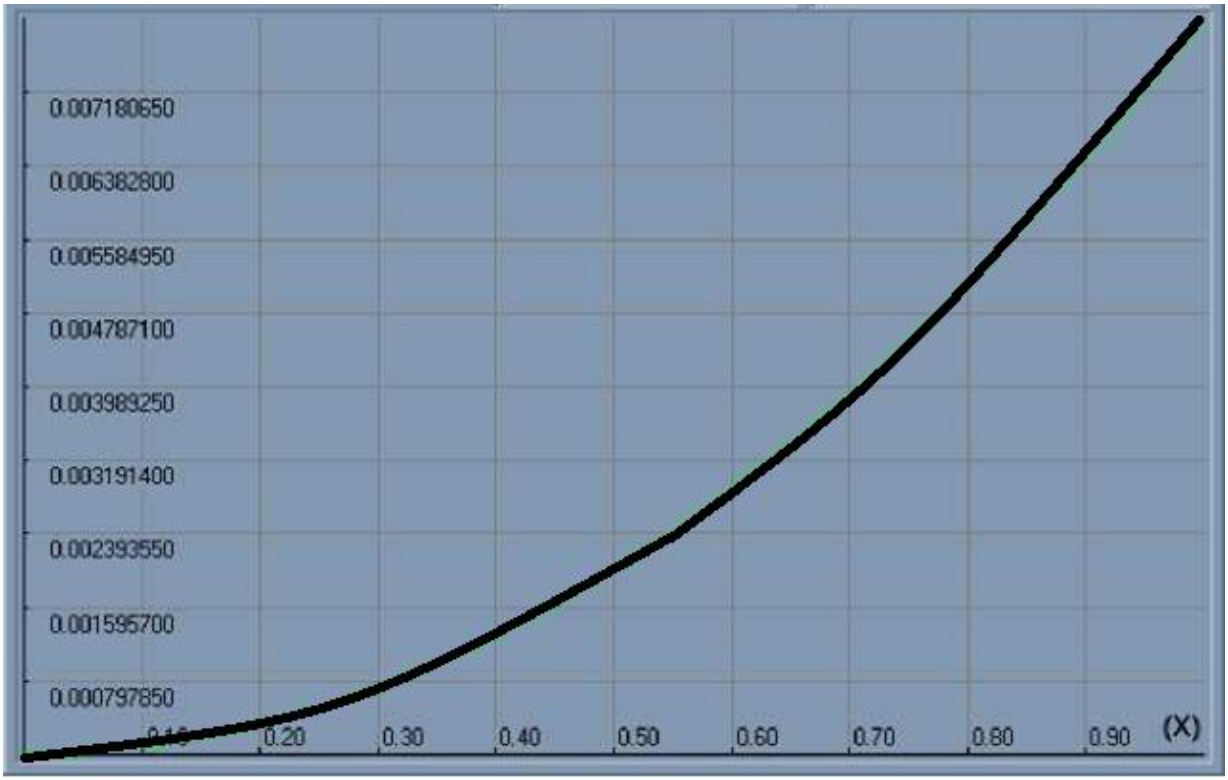


Рис. 14.5. Графік абсолютної помилки обчислення інтеграла як функції його верхньої межі методом правих прямокутників

14.1.3. Квадратурний метод трапецій

Напівсуму або середнє між її значеннями S_i за методом лівих і правих прямокутників можна записати в такому вигляді:

$$S_i = \frac{y_i(x_{i+1} - x_i) + y_{i+1}(x_{i+1} - x_i)}{2} = \frac{1}{2}(y_i + y_{i+1})h; \quad x_{i+1} - x_i = const = h.$$

Крім того, обчислення площі як півсуми основ, помножених на висоту, це визначення відомої геометричної формули для обчислення площі трапеції, що й визначило назву цього методу.

Оскільки тепер ми не можемо називати елементарні площі S_i квадратами, для таких елементарних ділянок було обрано найменування "квадратура", а всі подальші методи отримали назву "квадратурні методи".

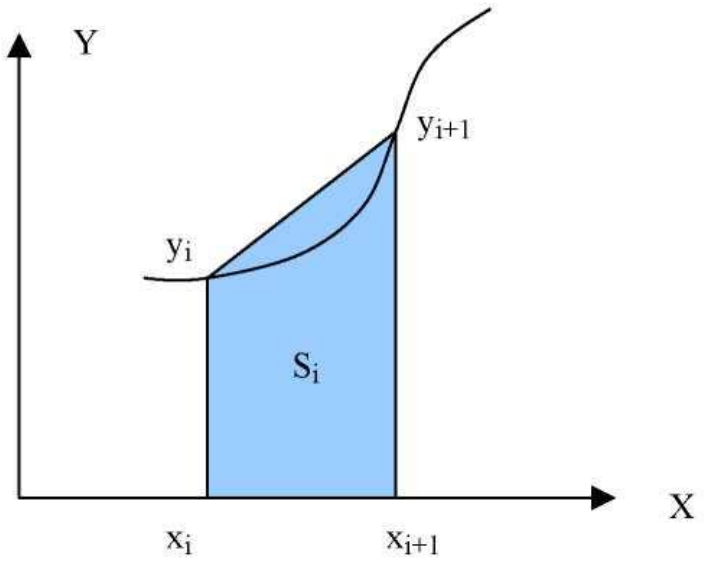


Рис. 14.6. Квадратура у вигляді трапеції з основами $F(Xi)$ і $F(X_{i+1})$

У методі трапецій для кожної ділянки висотою h буде використовуватися як правий, так і лівий відліки функції $F(X)$, а загальна площа під кривою (чисельне значення інтеграла) буде визначатися за формулою:

$$S = \sum_{i=0}^{i=n-1} S_i = \sum_{i=0}^{i=n-1} \frac{1}{2}(y_i + y_{i+1}) \cdot h = \frac{h}{2} \sum_{i=0}^{i=n-1} (y_i + y_{i+1})$$

де висота кожного елементарного прямокутника дорівнює діапазону, поділеному на число ділянок n:

$$h = \left(\frac{X_{end} - X_{beg}}{n}\right).$$

Для порівняння з методами лівих і правих прямокутників наведемо графік зміни абсолютної помилки AE під час обчислення інтеграла методом трапецій:

$$AE(X) = S(X) - \int_{X_{beg}}^{X_{end}} X^2 dx.$$

Зазначимо, що точне значення інтеграла (як функції від X) і відповідно площа $S(X)$, змінюються в міру того, як аргумент X з кроком h зростає в межах від $X_{beg} = 0$ до $X_{end}=1$. Загальне число ділянок (або n) дорівнює 63:

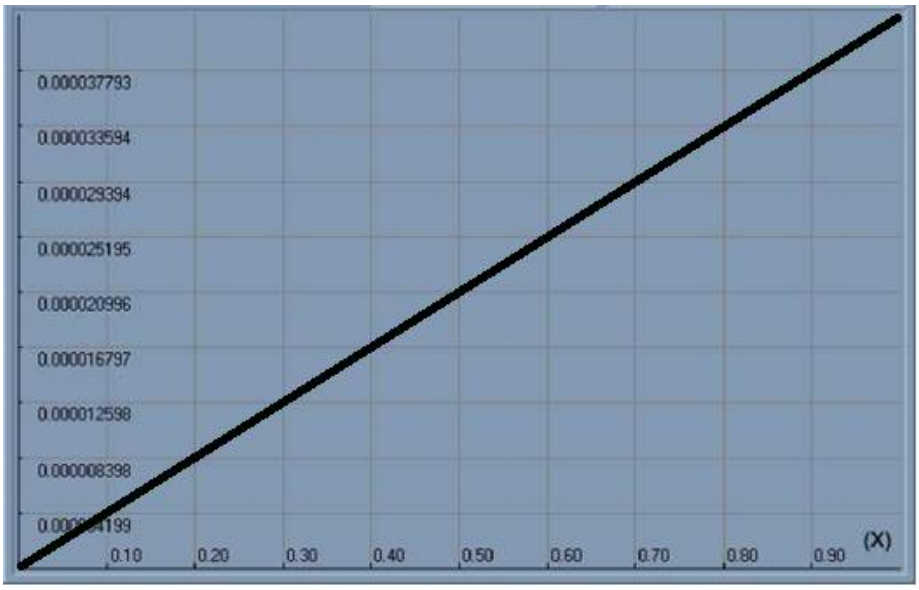


Рис. 14.7. Графік відносної помилки інтегрування параболи X^2 у межах від 0 до 1 на 63 ділянках методом трапецій

Як видно з графіків і значень абсолютних помилок, для інтегралів з верхньою межею інтегрування, що дорівнює 1, точність методу трапецій значно вища за точність методів окремо лівих або правих прямокутників. У випадку параболи з нашого прикладу, абсолютна помилка зменшується майже на два порядки.

14.1.4. Шляхи підвищення точності квадратурних методів.

Уважний читач одразу помітить, що чисельний приклад з інтегруванням параболи є окремим випадком і дає нам лише напрямок розвитку для ідеї підвищення точності обчислень. Це справді так. Однак ідея усереднення квадратур, отриманих різними методами для деякої ділянки зі знакосиметричними абсолютними помилками має конструктивний характер. Прикладом тому можуть слугувати інтерполяційні поліноми Стірлінга, з якими Ви ознайомитеся або на старших курсах, або (через необхідність) самостійно.

У нашому випадку зосередимо увагу на відмінностях між квадратурами лівих, правих прямокутників і квадратурами трапецій. Суть цих відмінностей полягає не тільки в геометричних уявленнях квадратур, але також і в аналітичних описах для обчислення таких квадратур. Тобто, якщо для опису квадратур використати деяку аналітичну функцію $P(T)$ від незалежної змінної T, то таку функцію можна буде інтегрувати аналітичним чином (наприклад, за допомогою табличних інтегралів) і, тим самим, отримувати чисельні значення конкретних S_i.

Так для квадратур лівих і правих прямокутників така функція може відповідно мати вигляд:

$$P(T) = A_0, \text{де } A_0 = F(X_i) \text{ або } A_0 = F(X_{i+1}).$$

Для квадратур методу трапецій функцію $P(T)$ можна записати так:

$$P(T) = A_0 + A_1 \cdot T, \qquad \text{де } A_0 = F(X_i), \qquad A_1 = \frac{F(X_{i+1}) - F(X_i)}{X_{i+1} - X_i},$$

якщо змінна T змінюватиметься від нуля до значення $(X_{i+1} - X_i)$.

Із запропонованих геометричних та аналітичних описів квадратур легко зробити кілька висновків.

1. Якщо для квадратур лівих і правих прямокутників ми фактично замінюємо вихідну функцію $F(X)$ сходинкою $P(T)$, то для квадратур за методом трапецій $P(T)$ являє собою вже опис ліній для кожної конкретної ділянки. Підкреслимо, що сходинка і лінія є відповідно степеневими поліномами нульового і першого ступеня. Крім того, лінія ближче примикає до кривої $F(X)$ ніж це робить сходинка. Очевидно, що якщо функції $F(X)$ і $P(T)$ на кожній конкретній ділянці повністю накладаються одна на одну, то абсолютна помилка буде прагнути до нуля. Таким чином, логічно (використовуючи принцип індукції) запропонувати як узагальнений вигляд функції $P(T)$ поліном k-того ступеня, що буде максимально примикати до кривої $F(X)$:

$$P(T) = \sum_{i=0}^{k} A_i T^i.$$

2. Природним чином виникає питання про спосіб обчислення коефіцієнтів такого полінома. У наведеному порівнянні ми також можемо помітити підказку тому, як це робити. Звернемо увагу, що значення A_1 для квадратури трапецій, який (при граничному переході) прагне до першої похідної від функції $F(X)$ у точці X_i.

$$\lim_{(X_{i+1}-X_i)\to 0} \left[\frac{F(X_{i+1}) - F(X_i)}{X_{i+1} - X_i}\right] \to \frac{F(X)}{dX}.$$

Тим самим як спосіб для визначення коефіцієнта поліномів $P(T)$ логічно запропонувати спосіб обчислення коефіцієнтів, який застосовують у рядах Тейлора або Маклорена. Це передбачає обчислення на основі різниць функції $F(X)$ її похідних різних порядків для відповідних точок X_i. Цей підхід використовувався багатьма відомими математиками, наприклад, Ньютоном, Гаусом, Беселем та іншими.

Однак, для початку, ми розглянемо логічно прозоріший спосіб визначення шуканих коефіцієнтів A_i. Ідея цього способу полягає в знаходженні деякого полінома $P(T)$, який на конкретній ділянці гарантовано проходить через задані відліки $F(X_i)$. Коефіцієнти A_i такого полінома можна визначити шляхом складання системи лінійних рівнянь. Підкреслимо, що невідомими в такій системі лінійних рівнянь будуть шукані нами коефіцієнти A_i.

Розглянемо приклад складання такої системи лінійних рівнянь для полінома $P(T)$ другого ступеня:

$$P(T) = A_0 + A_0 T + A_0 T^2.$$

Як і у випадку, раніше розглянутих квадратур, нехай відліки *Yi* або чисельні значення функції $F(X_i)$ задано нам із постійним кроком за аргументом:

$$X_{i+1} - X_i = const = h.$$

Оскільки аргумент полінома $P(T)$ змінюється починаючи від нуля, а обчислені значення полінома повинні збігатися з відліками Y_i, за відповідних аргументів, то система лінійних рівнянь, яку складають, матиме вигляд:

$$\begin{cases} Y_i = A_0; \\ Y_{i+1} = A_0 + A_1 h + A_2 h^2; \\ Y_{i+2} = A_0 + A_1(2 \cdot h) + A_2(2 \cdot h)^2. \end{cases}$$

Оскільки коефіцієнт A_0 визначається автоматично, то система лінійних рівнянь розв'язується досить просто. З цієї причини ми не будемо зупинятися на деталях її розв'язання і відразу наведемо остаточні результати:

$$A_0 = Y_i;$$
$$A_1 = \frac{-3 \cdot Y_i + 4 \cdot Y_{i+1} - Y_{i+2}}{2h};$$
$$A_2 = \frac{Y_i - 2 \cdot Y_{i+1} + Y_{i+2}}{2h^2}.$$

Тепер, коли функцію $F(X)$ на ділянці X_i ... X_{i+1} ми замінили поліномом $P(X)$ можна обчислити площу відповідної квадратури S_i. Для цього скористаємося табличним інтегралом:

$$\int_0^{2h} x^m dx = \frac{x^{m+1}}{m+1}\Big|_0^{2h}$$

по відношенню до кожного члена полінома $P(X)$:

$$\int_0^{2h} P(T)dT = (A_0 T + \frac{A_1 T^2}{2} + \frac{A_2 T^3}{3})\Big|_0^{2h}.$$

Після підстановки меж інтегрування отримаємо:

$$S_i = 2A_0h + 2A_1h^2 + \frac{8A_2h^3}{3}.$$

Після підстановки обчислених значень коефіцієнтів A_i і приведення подібних, остаточний результат матиме вигляд:

$$S_i = \frac{h}{3}(Y_i + 4Y_{i+1} + Y_{i+2}).$$

Очевидно, що незважаючи на відносно громіздкі побудови і перетворення, остаточний результат має досить простий і компактний вигляд, а квадратура, обчислена в такий спосіб, отримала найменування квадратури Сімпсона.

14.1.5. Квадратурна формула Сімпсона

Площа елементарної квадратурної форми, яка обчислюється шляхом аналітичного інтегрування найближчого до $F(X)$ полінома другого ступеня $P(T)$ на ділянці (X_i ...X_{i+2}) при рівномірному кроці аргументу має вигляд:

$$X_{j+1} - X_j = const = h;$$

$$S_i = \frac{1}{3}(y_i + 4y_{i+1} + y_{i+2})h.$$

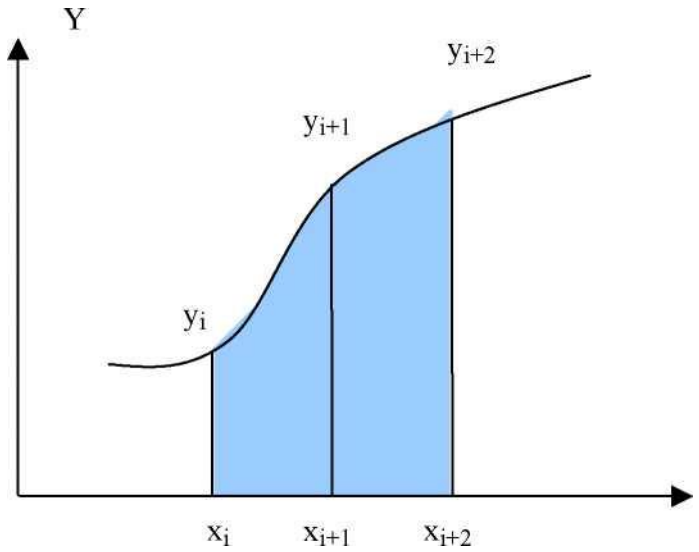


Рис. 14.8. Квадратура Сімпсона, побудована за відліками $F(X_i)$, $F(X_{i+1})$ и $F(X_{i+2})$

Звернемо увагу, що для побудови однієї квадратури Сімпсона необхідно дві ділянки h. Це означає, що для обчислення всього інтеграла в межах від X_{beg} до X_{end}, загальне число ділянок n має бути парним, тобто кратним двійці. У цьому випадку, коли число ділянок парне, загальна площа під кривою (чисельне значення інтеграла) визначатиметься за формулою:

$$S_i = \sum_{i=0}^{i=n/2} S_i = \frac{h}{3} \sum_{i=0}^{i=n/2} (y_{2i} + 4y_{2i+1} + y_{2i+2}).$$

де довжина кожної ділянки $h = (x_{end} - x_{beg})/n$, а значення n – парна кількість ділянок.

При цьому одразу виникає питання - а що робити, якщо загальне число доступних ділянок є непарним числом? Відповідь досить проста - одну з ділянок слід обчислити методом трапецій, набір ділянок, що залишився, стане кількісно парним.

Оскільки квадратура Сімпсона описується поліномом другого ступеня, то у випадку, коли вихідна функція $F(X)$ також є поліномом другого ступеня, абсолютна помилка інтегрування дорівнюватиме нулю. З цієї причини, для ілюстрації абсолютних помилок AE скористаємося таким прикладом:

$$AE(X) = S(X) - \int_{X_{beg}}^{X_{end}} X^3 dx.$$

Зазначимо, що точне значення інтеграла (як функції від X) і відповідно площа $S(X)$, змінюються в міру того, як аргумент X з кроком h зростає в межах від $X_{beg} = 0$ до $X_{end}=1$. Загальне число ділянок (або n) парне і дорівнює 62:

Легко помітити, що коли X відповідає парній кількості ділянок, абсолютна помилка досить мала і становить 5.42101086242752E-20. За непарної кількості ділянок трапеція на останній ділянці вносить вельми відчутну похибку (коли X відповідає трьом ділянкам, значення похибки дорівнює 1.69189439114E-8, коли X

відповідає 61-й ділянці значення досягає 2.04719221327939E-6). Співвідношення таких помилок сягає майже 16-порядків. Таким чином, стає очевидним, що слід продовжити пошук прийнятного поєднання квадратурних методів.

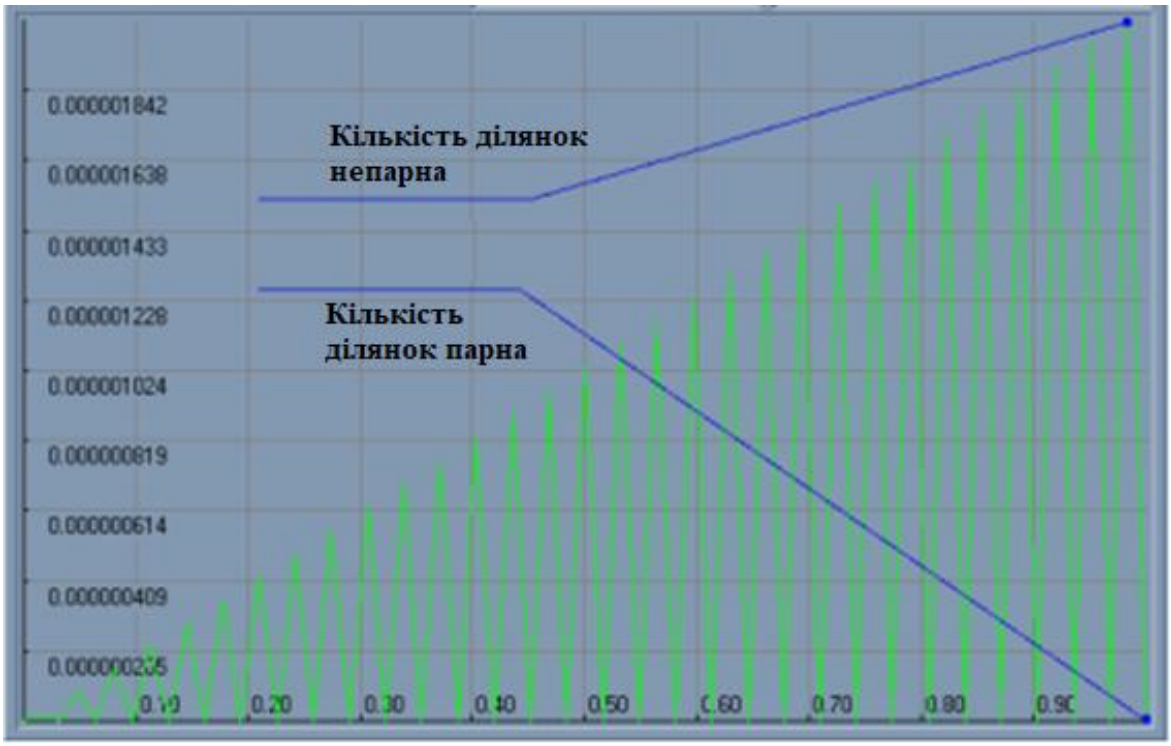


Рис. 14.9. Графік відносної помилки інтегрування функції X^3 у межах від 0 до 1 на 62 ділянках методом Сімпсона й останньої непарної ділянки методом трапецій.

14.1.6. Кілька модифікацій квадратури Сімпсона

Розглядаючи спосіб виведення формули Сімпсона, ми використовували поліном $P(T)$, значення якого збігаються зі значеннями вихідної функції $F(X)$ у трьох сусідніх значеннях аргументу або на ділянці довжиною $2h$. Відповідно квадратуру Si обчислювали шляхом аналітичного інтегрування цього полінома також на ділянці $2h$. Однак нам також доступно виконати таке інтегрування як на лівій, так і на правій ділянці довжиною h. Тим самим обчислити квадратуру одиночної непарної ділянки. Оскільки деталі такого інтегрування ми вже розглядали, то відразу наведемо остаточний результат.

Квадратура для лівої ділянки:

$$S_i^{(L)} = \int_0^h P(T)dT = \frac{h}{12}(5Y_i + 8Y_{i+1} - Y_{i+2}).$$

Квадратура для правої ділянки:

$$S_i^{(R)} = \int_h^{2h} P(T)dT = \frac{h}{12}(-Y_i + 8Y_{i+1} + 5Y_{i+2}).$$

Для порівнянності скористаємося оцінкою абсолютних помилок *AE*, розглянутою в попередньому прикладі:

$$AE(X) = S(X) - \int_{X_{beg}}^{X_{end}} X^3 dx.$$

Зазначимо, що точне значення інтеграла (як функції від *X*) і відповідно площа *S*(*X*), змінюються в міру того, як аргумент *X* з кроком *h* зростає в межах від X_{beg} 0 до X_{end}=1. Загальне число ділянок (або *n*) парне і дорівнює 62.

Таким чином, коли для останніх непарних ділянок замість трапецій ми застосовуємо квадратури $S_i^{(R)}$, точність обчислення покращується. В рамках цього прикладу за будь-якої парної кількості ділянок абсолютна помилка становить -5.42101086242752E-20, а за будь-якої непарної кількості 1.69189439113977E-8, що становить відмінність уже не на16, а на12 порядків.

Розглянуте поєднання методів уже становить інженерний інтерес. З цієї причини наведемо варіант програмної процедури, яка реалізує такий підхід.

Нехай, дано заповнений відліками $F(X_i)$ масив такого вигляду:

```
type TD1ArrP = array of record
Y, // Значення полиному
X // Аргумент поліному
: extended;end;
ArrP : TD1ArrP; // Масив точок F(Xi)
```

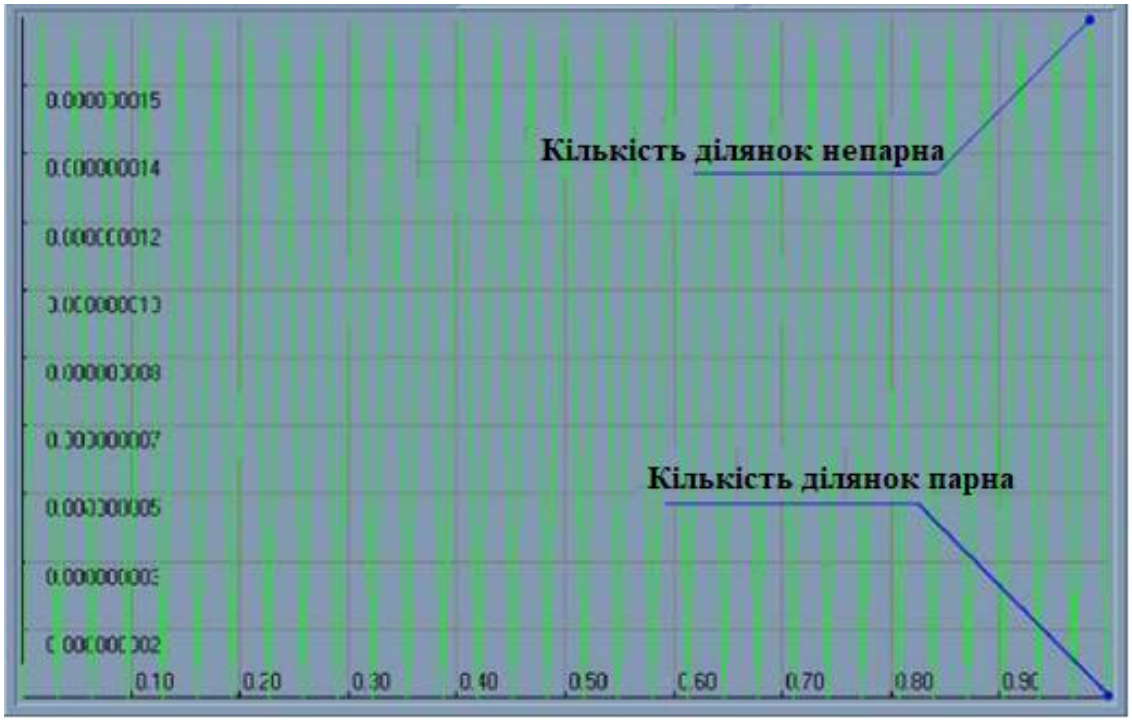


Рис. 14.10. Використання для останніх непарних ділянок квадратур $S_i^{(R)}$.

Тоді для обчислення інтеграла функцією *Simpson2IntegralIndBE*, необхідно вказати початковий (*IndB*) і кінцевий (*IndE*) індекси в цьому масиві, а також значення (*dX*) або *h*.

```
// Розрахунок інтеграла квадратурами Сімпсона на парних
ділянках і спеціальною правою квадратурою Сімпсона для
останньої непарної ділянки.

function Simpson2IntegralIndBE (RqArrP : TD1ArrP;
IndB, IndE : integer;
dX: extended) : extended;
var Ind : integer;
begin
Result := 0;
// Умова можливості інтегрування
if (Length(RqArrP) < 2) or (IndB >= IndE) or
(IndB < Low(RqArrP)) or (IndE > High(RqArrP)) then Exit;
// Умова застосування методу Сімпсона
if (IndE - IndB) >= 2
then begin
// Квадратури Сімпсона на парних ділянках
```

```
Ind := IndB;
while (Ind <= (IndE - 2))
do begin
Result := Result + (RqArrP[Ind].Y
+ 4 * RqArrP[Ind + 1].Y
+ RqArrP[Ind + 2].Y);
Ind := Ind + 2
end;
Result := Result / 3;
if ((IndE - IndB) mod 2) > 0
then begin
// Права квадратура Сімпсона на останній непарній ділянці
Result := Result
+ ( - RqArrP[IndE - 2].Y
+ 8 * RqArrP[IndE - 1].Y
+ 5 * RqArrP[IndE].Y ) / 12;
end;
Result := Result * dX;
end
// Для малого числа відліків застосувати альтернативний метод
else Result := TrapezIntegralIndBE (RqArrP, IndB, IndE, dX);
end; // of Simpson2IntegralIndBE
```

Розглядаючи перехід від квадратур лівих і правих прямокутників до квадратур трапецій ми застосовували усереднення. Чи можливий такий підхід для лівих і правих квадратур Сімпсона. Для відповіді на це запитання розглянемо відхилення між вихідною функцією $F(X)$ і поліномом $P(X)$.

На графіку (рис. 14.11) в діапазоні X_{beg}=0, X_{end}=1 представлені функція $F(X)$ і поліном $P(X)$, які мають однакові значення в точках X=0; X=0,5; X=1. При цьому:

$$F(X) = X^3;$$

$$P(T) = 0 - 0{,}5T + 1{,}5T^2, \text{де } T = X.$$

Як видно з цього чисельного прикладу, відхилення між функцією $F(X)$ і поліномом $P(X)$ мають на ділянках знакозмінні значення. Таким чином, усереднення двох, зрушених відносно одна одної лівої і правої квадратур Сімпсона, може мати конструктивний характер.

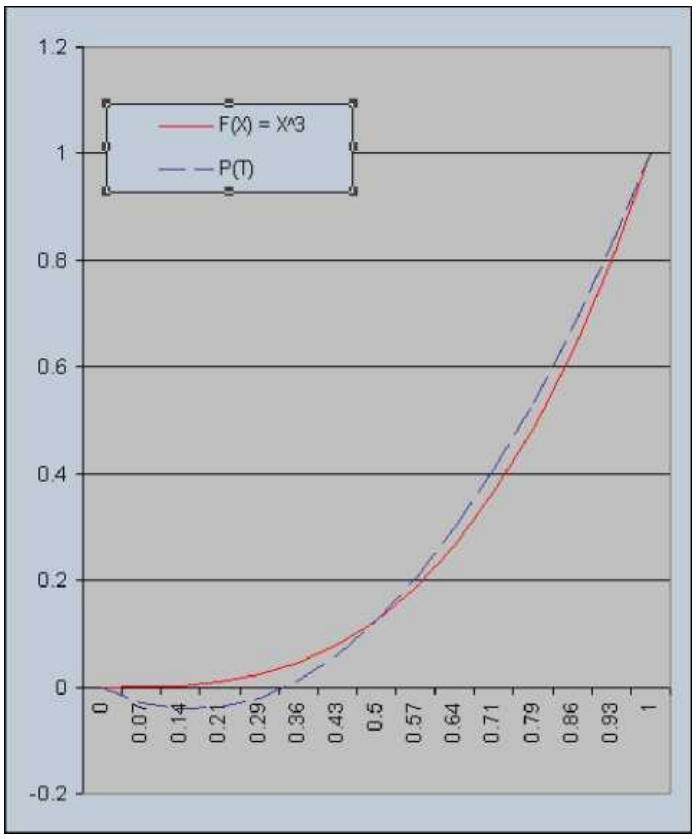


Рис. 14.11. Графіки функції $F(X)$ і полінома $P(X)$

Результат такого усереднення, після приведення "подібних" матиме вигляд:

$$S^{(M)} = \frac{S_i^{(R)} + S_{i+1}^{(L)}}{2} = \frac{h}{24}(13(Y_{i+1} + Y_{i+2}) - (Y_i + Y_{i+3}).$$

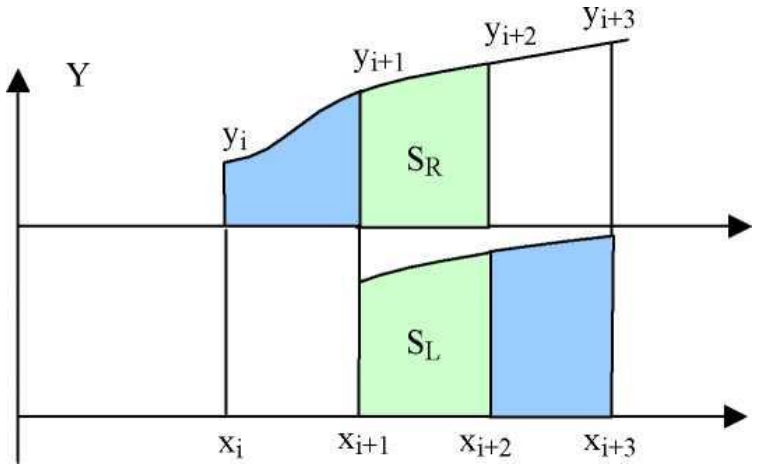


Рис. 14.12. Зсунуті відносно одна одної ліва і права квадратури Сімпсона

Для алгоритмічної реалізації обчислення непарної ділянки методом усередненої квадратури Сімпсона нам буде потрібно як мінімум 5 ділянок або 6 відліків функції *F*(*X*). При цьому в нас з'являється можливість обчислити дві класичні квадратури Сімпсона й одну усереднену.

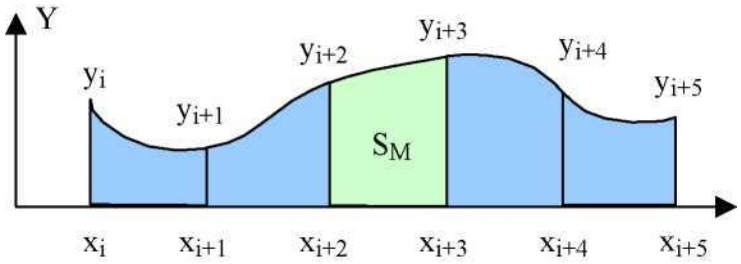


Рис. 14.13. Схема обчислення непарної ділянки усередненою усередненою квадратурою Сімпсона

Початковий текст процедури для реалізації такого алгоритму, в якому усереднену квадратуру Сімпсона обчислюють на початку області інтегрування, подано нижче:

```
// Розрахунок інтеграла квадратурами Сімпсона в діапазоні
індексів IndB, IndE.
// Для непарної ділянки застосовується усереднена квадратура
Сімпсона.
function Simpson3IntegralIndBE (RqArrP : TD1ArrP;
IndB, IndE : integer;
dX: extended) : extended;
var Ind : integer;
begin
Result := 0;
// Умова можливості інтегрування
if (Length(RqArrP) < 2) or (IndB >= IndE) or
(IndB < Low(RqArrP)) or (IndE > High(RqArrP)) then Exit;
// Умова застосування методу
if (IndE - IndB) >= 6
then begin
if ((IndE - IndB) mod 2 > 0)
```

```
then begin
// Звичайна та усереднена квадратури Сімпсона
// на початкових ділянках
Result   :=   (RqArrP[IndB].Y   +   4*RqArrP[IndB+1].Y   +
RqArrP[IndB+2].Y);
Result := Result + (13*(RqArrP[IndB+2].Y+
RqArrP[IndB+3].Y)- (RqArrP[IndB+1].Y+ RqArrP[IndB+4].Y))/8;
Ind := IndB + 3;
end
else Ind := IndB;
// Звичайні квадратури Сімпсона на парних (парних) ділянках
while (Ind <= (IndE - 2))
do begin
Result := Result + (RqArrP[Ind].Y + 4 * RqArrP[Ind + 1].Y +
RqArrP[Ind + 2].Y);
Ind := Ind + 2;
end;
Result := Result * dX / 3;
end
// Застосувати альтернативний метод
else Result := Simpson2IntegralIndBE (RqArrP, IndB, IndE,
dX);
end; // of Simpson3IntegralIndBE
```

Так само як і раніше, для порівнянності, скористаємося оцінкою абсолютної помилки *AE*, розглянутої в попередньому прикладі:

$$AE(X) = S(X) - \int_{X_{beg}}^{X_{end}} X^3 dx.$$

Зазначимо, що точне значення інтеграла (як функції від *X*) і відповідно площа *S(X)*, змінюються в міру того, як аргумент *X* з кроком *h* зростає в межах від $X_{\text{beg}} = 0$ до $X_{\text{end}}=1$. Загальне число ділянок (або *n*) парне і дорівнює 62.

З наведеного графіка видно, що поки число ділянок для обчислення інтеграла менше п'яти, алгоритм переходить на менш точні поєднання методів, а, починаючи з п'яти ділянок, абсолютна помилка стає мінімальною, тобто, приблизно дорівнює значенню – 5*E*-20.

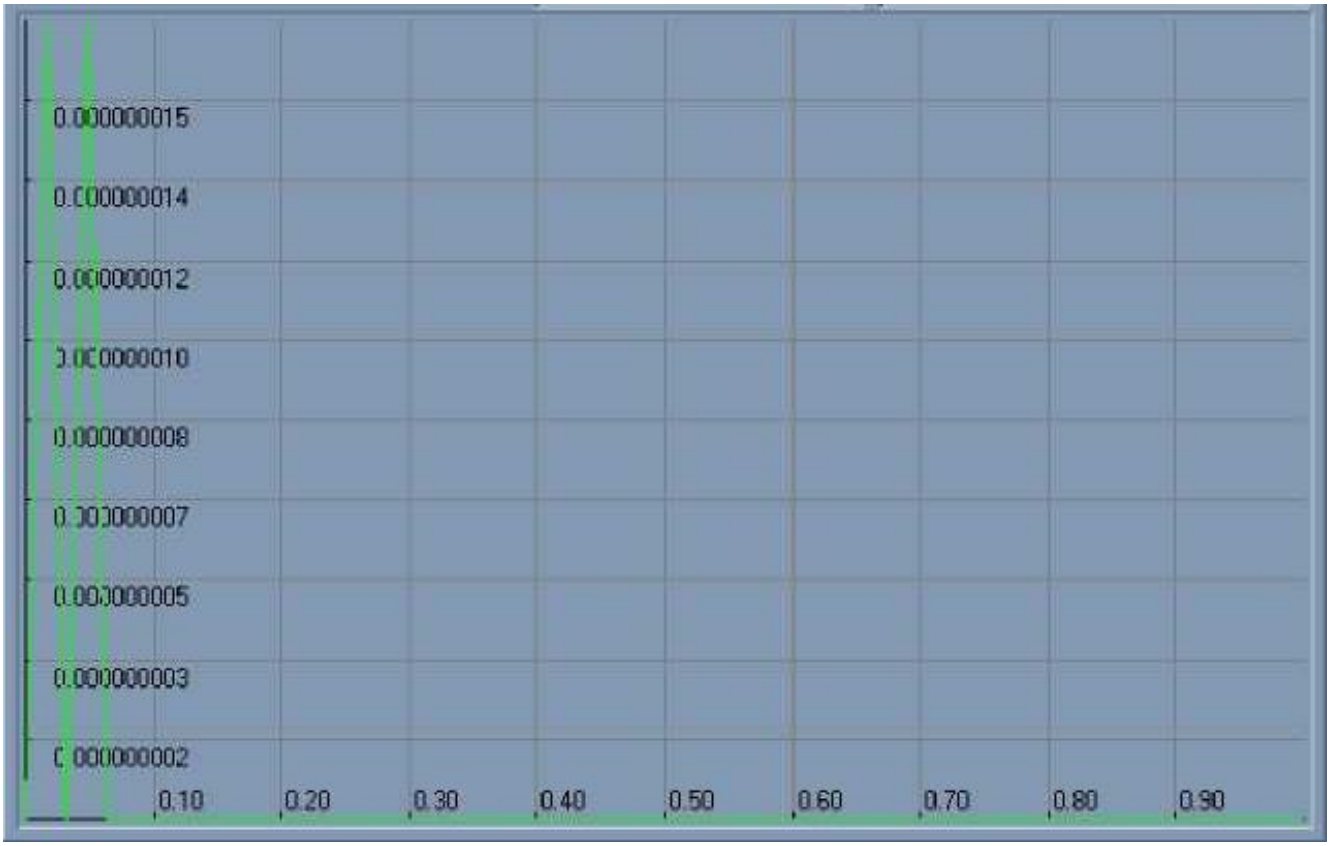


Рис. 14.14. Графік абсолютної помилки при використанні для початкових непарних ділянок квадратур $S^{(M)}$

14.4. Індивідуальні завдання

1. Для проведення чисельних експериментів з обчислення інтеграла функції використовувати програму *Integral*.

2. Відповідно до номера *N* за списком у журналі групи, записаного у вигляді $N = CM$, де C – старша цифра, M – молодша цифра, виберіть функцію (і відповідні їй коефіцієнти рядів Маклорена) з табл. 14.1. і методи обчислення інтеграла з табл. 14.1. та табл. 14.2.

3. За допомогою програми *Integral*, двома заданими методами обчислення інтеграла здійсніть визначення коефіцієнтів полінома та побудуйте графіки: полінома, інтеграла та абсолютної помилки між точним і квадратурним способом обчислення.

Таблиця 14.1

M	**Функція**
1	sin(*x*)
2	si(*x*) – інтегральний sin
3	sin(*x*) / *x*
4	sh(*x*) – гіперболічний sin
5	cos(*x*)
6	ch(*x*) – гіперболічний cos
7	exp(*x*)
8	arcsin(*x*), |*x*| <=1
9	arctg(*x*)
0	cos(*x*) / *x*

Таблиця 14.2

C	**Методи обчислення інтегралу**	
0	Метод лівих прямокутників	Сімпсон і трапеція
1	Метод правих прямокутників	Сімпсон і Сімпсон R
2	Метод трапецій	Сімпсон M і Сімпсон

4. Оформіть звіт з лабораторної роботи.

ВИМОГИ ДО ОФОРМЛЕННЯ ЗВІТІВ

Звіти оформлюються українською мовою у редакторі *Microsoft Word* шрифтом *Times New Roman* без перенесення. Міжрядковий інтервал – 1,5 рядки, абзацний відступ – 1,25 см. Обсяг звіту – не більше ніж 10 друкованих сторінок. Звіт готується на аркушах формату А4, орієнтація – книжкова. Поля: верхнє, нижнє та праве – по 2,0 см, ліве – 3,0 см. Розміри колонтитулів: верхнього – 0 см, нижнього – 2,0 см. Нумерація виконується внизу сторінки, розмір шрифту 14 pt, вирівнювання по центру.

У першому рядку друкується прізвище та ініціали студента, а також номер академічної групи. Розмір шрифту 12 pt, напівжирний, курсив, вирівнювання праворуч.

На другому терміні друкується номер роботи великими літерами (шрифт – прямий, напівжирний, 14 *pt,* вирівнювання по центру).

Далі друкується тема роботи великими літерами через один порожній рядок після номера роботи (шрифт – прямий, жирний, 14 *pt*, вирівнювання по центру).

Потім друкується мета роботи через один порожній рядок після теми роботи. Розмір шрифту 14 *pt*, відступ 1,25 пт, вирівнювання – по ширині.

Далі друкуються результати виконання всіх пунктів індивідуального завдання. Текст друкується шрифтом – 14 *pt,* абзацний відступ – 1,25 см, вирівнювання по ширині.

Наприкінці наводяться висновки щодо роботи. Вони друкуються через один порожній рядок після результатів виконання роботи. Розмір шрифту 14 *pt*, відступ 1,25 пт, вирівнювання – по ширині.

СПИСОК ЛІТЕРАТУРИ

1. Machine and Deep Learning Using MATLAB. Algorithms and Tools for Scientists and Engineers / *Kamal I. M. Al-Malah* // Wiley, 2023. 592 p.

2. Introduction to Intelligent Systems, Control, and Machine Learning Using MATLAB / *Marco Schoen* // Cambridge University Press, 2023. 450 p.

3. MATLAB Applications in Chemical Engineering / *Chyi-Tsong Chen* // Unknown Publisher, 2022. 576 p.

4. MATLAB and Simulink Crash Course for Engineers / *Eklas Hossain* // Springer International Publishing, 2022. 657 p.

5. Matlab for Beginners / *Peter Kattan* // PETRA BOOKS, 2022. 474 p.

6. Distribution System Modeling and Analysis with MATLAB and WindMil / *William H. Kersting, Robert Kerestes* // CRC Press, 2022. 496 p.

7. Fundamentals of Computational Intelligence / *O. Zakovorotniy, O. Lipchanska* // Laboratory workshop. Part 1. Kharkiv: NTU "KhPI", 2022. 160 p.

8. Fundamentals of Computational Intelligence / *O. Zakovorotniy, O. Lipchanska* // Laboratory workshop. Part 2. Kharkiv: NTU "KhPI", 2022. 152 p.

9. Visible Light Communication. Comprehensive Theory and Applications with MATLAB / *Suseela Vappangi, Vakamulla Venkata Mani, Mathini Sellathurai* // CRC Press, 2021. 502 p.

10. Programming and Engineering Computing with MATLAB 2021 / *Huei-Huang Lee* // SDC Publications, 2021. 532 p.

11. Condition Monitoring Algorithms in MATLAB / *Adam Jablonski* // Springer International Publishing, 2021. 527 p.

12. Multiphysics Modeling Using COMSOL 5 and MATLAB / *Roger W. Pryor* // Mercury Learning and Information, 2021. 626 p.

13. Fluid Mechanics. Problem Solving Using Matlab / *Raju, K. Srinivasa, Kumar, D. Nagesh* // PHI Learning Pvt. Ltd., 2020. 368 p.

14. Programming and Engineering Computing with MATLAB 2020 / *Huei-Huang Lee* // SDC Publications, 2020. 532 p.

15. Basics of MATLAB Programming / *R. Balaji* // Notion Press, 2020. 390 p.

16. MATLAB Fast Automation. Automate Your Work With MATLAB / *Jacob Sapir* // Amazon Digital Services LLC – Kdp, 2020. 85 p.

17. MATLAB Programming. Mathematical Problem Solutions / *Dingyü Xue* // De Gruyter,·2020. 318 p.

18. Differential Equation Solutions with MATLAB / *Dingyü Xue* // De Gruyter, 2020. 451 p.

Навчальне видання

ЗАКОВОРОТНИЙ Олександр Юрійович
ОРЛОВА Тетяна Олександрівна
ГРИНЬОВ Денис Валерійович
СЕРГІЄНКО Віталіна Миколаївна

ОСНОВИ КОМП'ЮТЕРНОЇ МАТЕМАТИКИ

Лабораторний практикум
для студентів денної та заочної форм навчання за спеціальністю
123 "Комп’ютерна інженерія"

Роботу до видання рекомендував *М. Й. Заполовський*
Відповідальний за випуск *С. Ю. Леонов*
У авторській редакції.

План 2022 р.

Підп. до друку 25.01.2023. Формат 60 × 84. Гарнітура Times New Roman. Обл.-вид. арк. 7,5.

Видавничий центр НТУ «ХПІ»
Свідоцтво про державну реєстрацію ДК № 5478 від 21.08.2017 р.
61002, Харків, вул. Кирпичова, 2.

Електронне видання