**МІНІСТЕРСТВО ОСВІТИ І НАУКИ,
МОЛОДІ ТА СПОРТУ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
“ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ”**

С. В. Ольшанський, О. Д. Асютін

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ОБЧИСЛЮВАЛЬНИХ СИСТЕМ НА БАЗІ FREEMAT

Навчально-методичний посібник

Харків 2012

МІНІСТЕРСТВО ОСВІТИ І НАУКИ, МОЛОДІ ТА СПОРТУ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
“ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ”

С. В. Ольшанський, О. Д. Асютін

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ОБЧИСЛЮВАЛЬНИХ СИСТЕМ НА БАЗІ FREEMAT

Навчально-методичний посібник

Затверджено
редакційно-видавничою
радою університету,
протокол №1 від 20.06.2012 р.

Харків
НТУ “ХПІ”
2012

УДК 621.396.218
ББК 32.884.1
О 56

Рецензенти:
О. К. Морачковський, д-р техн. наук, проф., НТУ”ХПІ”;
К. В. Аврамов, д-р техн. наук, проф., ІПМАШ ім. А. М. Підгорного НАН України

О56 Ольшанський С. В. Програмне забезпечення обчислювальних систем на базі FreeMat : навч.-метод. посіб. / С. В. Ольшанський, О. Д. Асютін. – Х.: НТУ “ХПІ”, 2012. – 137 с.

Посібник присвячено вивченню вільного математичного пакета FreeMat. Наведено велику кількість прикладів програм та пояснень до них, містить лабораторні роботи.

Призначено для студентів спеціальностей “Прикладна математика” та “Інформатика” з курсів “Програмне забезпечення обчислювальних систем” і “Математичне моделювання”.

Іл. 74. Табл. 3. Бібліогр. 23 назви.

УДК 621.396.218
ББК 32.884.1

ISBN

ВСТУП

В останні двадцять років з'явився та набув розвитку новий фундаментальний науковий напрямок – комп'ютерна математика, яка зародилася на перетині математики та інформатики. Одними із перших засобів для автоматизованого виконання науково-технічних розрахунків стали програмовані калькулятори [1]. З виникненням персональних комп'ютерів їх стали широко застосовувати для числових розрахунків, програмованих на мовах високого рівня, наприклад на Сі, Бейсік, Фортран та ін.

Виявляється, що більшість наукових і навчальних розрахунків носять якщо не одноразовий, то досить рідкісний характер. Деякі класи розрахунків використовуються часто, але їх конкретні задачі повторюються рідко. За таких умов саме програмування обчислювальних задач стало займати незрівнянно більше часу, ніж їх розв'язання. При цьому воно найчастіше не має творчого характеру, є досить рутинним (хоча складним і трудомістким) та мало зрозумілим математикам, фізикам і студентам, які спеціалізуються в цих галузях знань. Для підготовки програм таких розрахунків ще зовсім недавно використовувалися різноманітні універсальні мови програмування [2–4]. Вихід із ситуації, яка склалася, з'явився зі створенням систем комп'ютерної математики.

Провісниками систем комп'ютерної математики були спеціалізовані програми для математичних числових розрахунків, які працювали в середовищі MS-DOS. Це Mercury [5], Eureca [6], перші версії систем Mathcad [7] і MATLAB [8] під MS-DOS. Пізніше на основі досягнень комп'ютерної математики з'явились найновіші програмні системи символьної математики або комп'ютерної алгебри. Серед них особливо відомими стали Mathcad під Windows [9], Mathematica [10–11], Maple [12–13].

Перші системи комп'ютерної алгебри були достатньо "повільними" програмами. Все ж таки вони помітно зменшили час загального розв'язання задач, перш за все, за рахунок спрощення етапу програмування. Але останнім часом у таких програмах різко зросла швидкість обчислень, що відкриває перед системами комп'ютерної математики значні перспективи застосування математичного моделювання різних фізичних явищ і технічних систем. Навряд чи є хоч один серйозний науковий проект, пов'язаний з математикою та в цілому з наукою й технікою, де б вони не застосовувалися. Лідером у галузі числових і матричних розрахунків, а також у реалізації техніки імітаційного та ситуаційного моделювання стала потужна система MATLAB із її численними пакетами розширення [14–20]. Однак у галузі аналітичних обчислень вона поступається таким системам, як Maple і Mathematica.

Особливо велике значення мають системи комп'ютерної математики в освіті, де вони стають не тільки зручним інструментальним засобом для виконання великої кількості навчальних розрахунків, але й засобом надання учням, а нерідко й викладачам, знань у галузі математики, фізики та інших наук, які використовують математичні методи. Це дозволяє віднести такі системи до інтелектуальних комп'ютерних систем у галузі математичних розрахунків. Важко переоцінити їх значення у підготовці високоякісних електронних уроків, навчальних курсів і книг.

Цей навчальний посібник присвячено системі FreeMat. FreeMat – вільне середовище для числових обчислень і мова програмування, подібна MATLAB и GNU Octave [21]. У доповнення до підтримки багатьох функцій MATLAB і деяких функціональних можливостей IDL, має інтерфейс до зовнішнього коду на C, C++ і Fortran, а також коннективні алгоритми (через MPI), побудови графіків і тривимірної візуалізації. FreeMat достатньо молода система порівняно з MATLAB і вона має значно менший набір функціональних можливостей. Але із завданнями, які ставляться в рамках навчального процесу, вона справляється на відмінно.

FreeMat як і MATLAB орієнтований на матричні операції. Матриці широко застосовуються в математичних розрахунках, наприклад, при

розв’язанні задач лінійної алгебри та математичного моделювання статичних, динамічних систем і об’єктів.

Одним із основних завдань системи FreeMat є надання користувачеві потужної мови програмування, орієнтованої на технічні та математичні розрахунки, що в змозі перевершити можливості традиційних мов програмування, які багато років використовувалися для реалізації числових методів. При цьому особлива увага приділялася як підвищенню швидкості обчислень, та і адаптації системи до розв’язання самих різноманітних задач користувачів.

Літератури з FreeMat українською або російською мовами під час написання посібника не було. Існувала тільки офіційна документація [22] та вільна література американських авторів [23].

Важливою перевагою системи FreeMat є її відкритість і розширюваність. Більшість команд і функцій системи реалізовані у вигляді m-файлів текстового формату (з розширенням .m), причому всі файли доступні для модифікації. Користувачеві надана можливість створювати не тільки окремі файли, але й бібліотеки файлів для реалізації специфічних задач. Такі файли можна готувати як в простому та зручному редакторі m-файлів системи FreeMat, так і в будь-якому іншому текстовому редакторі. Більше того, такі файли можна переносити за допомогою буфера в командний рядок FreeMat і тут же виконувати.

Матеріал посібника розбитий на шість розділів.

Перший розділ присвячено опису початку роботи в FreeMat. Тут розглянуто інсталяцію FreeMat на операційні системи Windows 7 і Ubuntu Linux. Запропоновано текст вашої першої програми. Описано вікно програми, тонкощі налаштування параметрів робочого середовища, установка робочої директорії, списку робочих шляхів.

У другому розділі увага приділена математичним функціям та операціям, а також їх виконанню. Зокрема, розглянуто функції суми, добутку, накопичення суми та накопичення добутку. Розглянуто тригонометричні функції, експоненти та логарифми.

Третій розділ посібника описує можливості роботи з матрицями та масивами. У підрозділах детально описано способи створення масивів і математичні операції над масивами. Важливою особливістю FreeMat є можливості поелементних операцій із масивами.

Четвертий розділ має назву “Програми та функції”. Тут описано процес створення та запуску програм, а також створення та запуску функцій. Розглянуто програмне введення та виведення, функції зчитування даних.

П’ятий розділ присвячено управлінню потоком команд. Розглянуто оператори порівняння, а також оператори циклів for, while, умовний оператор if-elseif-else.

Шостий розділ присвячено графічним можливостям FreeMat. Тут розглянуто питання створення графіків за допомогою різних команд, а також нстройки їх властивостей.

Викладення теоретичних основ супроводжується прикладами програм із числовими розрахунками. У посібнику запропоновано лабораторний практикум, який складається з восьми робіт. Автори сподіваються, що такий стиль подачі матеріалу допоможе спростити його засвоєння, особливо при самостійному вивченні.

При викладенні матеріалу були враховані зауваження рецензентів. Автори вдячні рецензентам, а також професорам В. П. Ольшанському та Д. В. Бреславському за зроблені зауваження, які сприяли поліпшенню змісту цього посібника.

1. РОБОТА У FREEMAT

1.1. Інсталяція FreeMat на OS Windows 7

Нижче наведено покрокові інструкції з установки FreeMat в OS Windows 7:

1. Скачайте установочні файли з офіційного сайту програми FreeMat – http://freemat.sourceforge.net/download.html.

Зверніть увагу (рис. 1.1), що для системи Windows можливі два варіанти скачування: реліз FreeMat (FreeMat-4.0-win32.exe) і версія програми із репозиторію (FreeMat-4.0-svn_rev4000-win32.exe). Необхідно наголосити, що на момент скачування вами програми номер її версії може відрізнятися від наведеного на рис. 1.1.

Рисунок 1.1 – Вікно скачування установочного файлу FreeMat

2. Запустіть скачаний установочний файл. Якщо операційна система поверне діалогове вікно з проханням отримати права адміністратора, натисніть «Да». На екрані з’явиться початкове вікно інсталятора FreeMat (рис. 1.2).

Рисунок 1.2 – Початкове вікно інсталятора FreeMat

3. Натисніть кнопку «Далее». На екрані з'явиться вікно з ліцензійною угодою програми FreeMat. Програма поширюється під ліцензією GNU і є абсолютно безкоштовною. Натискаємо кнопку «Принимаю».

4. У наступному діалоговому вікні пропонується обрати параметри установки (рис. 1.3). Тут ви можете додати програму FreeMat у список системних шляхів, а також додати ярлик програми на робочий стіл. Після установки всіх необхідних параметрів натискаємо на кнопку «Далее».

Рисунок 1.3 – Параметри установки FreeMat

5. З'явиться діалогове вікно, де необхідно вибрати папку для установки програми. За умовчанням пропонується встановлювати програму в папку «Program Files». Після вибору папки для установки натисніть на кнопку «Далее».

6. З'явиться діалогове вікно вибору папки в меню «Пуск» для створення ярлика програми FreeMat. Після вибору даної папки натискаємо кнопку «Установить» – починається установка програми. Після закінчення установки з'явиться вікно завершення роботи майстра установки. Натискаємо кнопку «Готово» – програма FreeMat встановлена.

7. Для запуску програми необхідно двічі натиснути кнопкою миші на ярлик на робочому столі (якщо ви обрали відповідну опцію) або в меню пуск у полі введення набрати FreeMat і натиснути Enter.

1.2. Інсталяція FreeMat на OS Ubuntu Linux

Нижче наведена покрокова інструкція з установки FreeMat на OS Ubuntu Linux:

1. У меню операційної системи обираємо Приложения → Центр приложений Ubuntu. На екрані з'явиться вікно програми Ubuntu Software Center.

2. Вводимо в поле введення для пошуку «FreeMat», обираємо відповідну програму та натискаємо «Установить» (рис. 1.4).

Рисунок 1.4 – Центр програм Ubuntu зі знайденою в ньому програмою FreeMat

3. Ubuntu запросить пароль поточного користувача. Після введення пароля почнеться установка в автоматичному режимі. Після закінчення установки програма FreeMat буде доступною через меню Ubuntu Приложения → Наука → FreeMat.

1.3. Перша програма на FreeMat

Запускаємо програму FreeMat (про те як це зробити можна прочитати в попередніх двох підрозділах залежно від вашої операційної системи). На екрані з'явиться таке вікно (рис. 1.5):

Рисунок 1.5 – Основне вікно програми FreeMat

Нарисуємо графік функції $\sin(x)$ на інтервалі від 0 до 2π. Для цього помістимо курсор у вікні FreeMat біля символу --> і набираємо такі три команди (завершуючи кожну з них натисканням Enter):

```
--> x = linspace(0,2*pi,50);
--> y = sin(x);
--> plot(x,y);
```

У результаті з'явиться вікно з графіком функції $\sin(x)$ (рис. 1.6).

Рисунок 1.6 – Результат виконання команд. Графік функції $\sin(x)$

Вітаємо! Ви тільки що написали вашу першу програму у FreeMat. Тепер давайте докладно вивчимо цей математичний пакет. Почнемо з інтерфейсу програми та її основних настройок.

1.4. Основне вікно програми (версія 4.0)

Основне вікно програми FreeMat виглядає як показано на рис. 1.5. Використовуючи командне вікно, ви можете реалізувати введення змінних, функцій програм і команд. Розділ History (Історія) включає всі команди, які вводилися користувачем у хронологічному порядку. Відразу після запуску FreeMat у командному вікні відображається базова інформація про програму: версія програми, інформація про авторські права та декілька корисних команд і поєднань клавіш.

Вікно програми FreeMat розділено на декілька секцій: розділ команд, файловий браузер, розділ «Історія», розділ «Змінні» та розділ «Настройка» (рис. 1.7).

Рисунок 1.7 – Вікно FreeMat з виділеними розділами

Основні дії з програмою FreeMat реалізуються у розділі команд. Розглянемо всі розділи вікна програмного пакета FreeMat.

1.4.1. Файловий браузер

Цей розділ відображає стандартне дерево каталогів робочої директорії (рис. 1.8).

Рисунок 1.8 – Файловий браузер

Описання робочої директорії та її відношення до оброблюваних FreeMat файлів буде описано в розд. 1.4.5 – Робоча директорія. У версії FreeMat 3.6 це вікно оновлювалося тільки після перезапуску програми. Тепер це вікно оновлюється щоразу, коли змінюється робоча директорія.

1.4.2. Розділ «Історія»

У розділі «Історія» відображається список введених команд. Останні відображаються у кінці списку, перші – на початку. У розділі команд у командному рядку ви можете повторно набрати команди із розділу «Історія», використовуючи клавіші ↑ і ↓ або за допомогою подвійного натискання мишкою на команду в розділі «Історія».

Історія введених команд може зберігатися від запуску до запуску програми FreeMat. Таким чином, при закритті та наступному відкритті програми в розділі «Історія» зберігаються виконані минулого запуску команди. Ви можете очистити розділ «Історія» за допомогою команди з меню Tools → Clear History Tool.

1.4.3. Розділ «Змінні»

Розділ «Змінні» відображає всі поточні глобальні змінні. Даний розділ також відображає змінні, які використовуються в функціях доти, поки йде виконання функції. Цей розділ використовується найбільш інтенсивно при роботі із FreeMat. Він дозволяє швидко продивлятися використовувані змінні.

1.4.4. Розділ «Настройка»

Розділ «Настройка» відображає будь-які помилки або попередження, генеровані програмою FreeMat. Зазвичай настроювальне повідомлення включає в себе час, коли воно було зареєстровано та сам текст помилки або попередження.

1.4.5. Робоча директорія

Робоча директорія визначає каталог за умовчанням, із якого FreeMat зчитує файли із даними та записує результати своєї роботи. Наприклад, якщо ви хочете зберегти графік функції у вигляді зображення, FreeMat запропонує вам зберегти його в робочу директорію. Ви можете при цьому вказати будь-яку іншу директорію для даного зображення. Визначити поточну директорію можна або за відповідним розділом на панелі інструментів, або за допомогою команди **pwd** (print working directory). Нижче наведено приклад результату роботи даної команди.

Windows 7

```
--> pwd
ans =
C:/Program Files/FreeMat
```

Ubuntu Linux

```
--> pwd
ans =
/home/lexicus/Freemat
```

1.5. Настройка FreeMat

Перш ніж розпочати використання, давайте спочатку настроїмо декілька параметрів середовища FreeMat. Під час запуску FreeMat використовує деякі директорії вашої файлової системи. FreeMat буде записувати дані в робочу директорію, шукати визначений набір файлів користувача в системних шляхах (це буде детально описано нижче) та буде здійснювати зчитування даних із визначених директорій.

Коли програма FreeMat намагається завантажити необхідні їй функції та програми, вона шукає їх в таких директоріях: поточна робоча директорія, список зареєстрованих шляхів, коренева директорія. FreeMat буде шукати необхідний йому файл доти, поки не найде перший файл, який задовольняє критеріям пошуку. У процесі роботи з FreeMat може виникнути така ситуація, коли користувач створює декілька версій файлу з одним і тим же ім'ям (файли розміщені в різних каталогах). Відповідно при запиті даного файлу FreeMat буде використовувати перший файл, який він знайде. «Перший» в даному випадку означає файл, який програма знаходить першим при перегляді структури директорій (це не пов'язано із часом створення та редагування файлу, відповідно не гарантує завантаження останньої версії файлу).

Деякі функції FreeMat, така як **wavread,** будуть працювати тільки з файлами із поточної робочої директорії або файлами, заданими за допомогою абсолютного шляху.

Коли FreeMat записує файл на жорсткий диск, запис здійснюється в поточну робочу директорію або в директорію, визначену користувачем (за допомогою абсолютного шляху).

Примітка! У випадку, якщо програма FreeMat встановлена на операційну систему Windows Vista або Windows 7, робоча директорія за умовчанням знаходиться в C:\Program Files\FreeMat. Дані операційні системи захищають всі файли, які знаходяться всередині директорії Program Files (включаючи вкладені директорії). Таким чином, FreeMat не може здійснити запис у дану директорію. На жаль, ні програма FreeMat, ні сама операційна система не надають ніякої інформації про те, що операція збереження даних не може бути виконана. Якщо ви використовуєте FreeMat у даних операційних системах, необхідно змінити робочу директорію на директорію всередині папки облікового запису користувача або будь-якої іншої папки, в яку дозволено запис.

1.5.1. Установка робочої директорії за допомогою команди cd

Робоча директорія – директорія за замовчуванням, із якої програма FreeMat зчитує дані та записує результати своєї роботи (у випадку, якщо явно не вказана інша директорія).

Команда **cd** змінює робочу директорію програми. Нижче наведено приклади використання даної команди на операційних системах Windows 7 и Ubuntu Linux:

Приклад використання команди **cd** на OS Windows 7

```
--> pwd
ans =
C:/Program Files/FreeMat
--> cd 'C:/Users/lexicus/Documents/FreeMat'
--> pwd
ans =
C:/Users/lexicus/Documents/FreeMat
```

Приклад використання команди **cd** на OS Ubuntu Linux

```
--> pwd
ans =
/home/lexicus
--> cd '/home/lexicus/FreeMat/'
--> pwd
ans =
/home/lexicus/FreeMat
```

Примітка! Зазвичай у системах Linux при задаванні шляху використовується прямий слеш «/», в той час як системи Windows іноді використовують прямий слеш «/», а іноді і зворотний слеш «\». У програмі FreeMat для версії Windows (Win XP, Vista, Win 7) немає ніякої різниці, чи використовуєте ви прямий слеш «/», або зворотний слеш «\». FreeMat автоматично змінює слеші на потрібні. Для операційної системи Linux ви повинні використовувати тільки прямий слеш «/», інакше програма згенерує повідомлення про помилку.

Для системи Windows стандартна робоча директорія розташована в C:\Program Files\FreeMat *X,* де *X* – спочатку встановлена версія програми FreeMat (поточна версія програми може відрізнятися від *X* у випадку, якщо ви виконували оновлення програми).

За умовчанням FreeMat буде зберігати всі ваші дані в робочу директорію. Зміна директорії для зберігання не викликає труднощів за допомогою команди «Save File» і використання діалогового вікна. Однак при програмному зчитуванні (за допомогою таких команд як **fread**, **wavread** або **dlmread**) і зберіганні (**wavwrite**, **print**) даних, вам, можливо, доведеться змінити робочу директорію за умовчанням на відмінну від «C:\Program Files\FreeMat *X*».

Приклад установки робочої директорії за умовчанням для систем Windows. Для цього треба виконати такі три кроки:

1. Створити нову папку для ваших файлів у FreeMat (зазвичай це робиться в робочій директорії поточного користувача операційної системи).

2. Додати цю папку у ваш список шляхів FreeMat.

3. Написати програму під назвою startup.m, яка буде включати в себе команду cd зі створеної новою директорією за замовчуванням.

Перший крок повинен бути знайомим усім користувачам персональних комп'ютерів. Для виконання другого кроку необхідно відкрити утиліту FreeMat Path Tool, додати в список шляхів вашу нову папку, зберегти зміни та закрити утиліту. Ви можете відкрити утиліту FreeMat Path Tool двома способами: або вибором у меню FreeMat Tool → Path Tool, або за допомогою команди **pathtool** у головному командному вікні FreeMat. Після відкриття утиліти Path Tool оберіть на лівій панелі створену вами папку та натисніть кнопку Add. Після цього натисніть Save і Done. Вікно утиліти Path Tool зображено на рис. 1.9.

Рисунок 1.9 – Утиліта FreeMat Path Tool

Примітка! Всі зміни, які вносяться в настройки FreeMat за допомогою утиліти Path Tool вступають в дію після того, як ви закриєте і знову запустите FreeMat. Якщо ви використовуєте команди path або setpath, зміни вступають в дію негайно.

Останній крок – створити програму startup.m і помістити її в нову створену директорію. Для цього вам необхідно зробити так:

1. Відкрийте редактор FreeMat. Для цього введіть команду edit або оберіть в меню Tool → Editor.

2. У вікні редактора введіть команду **cd** і абсолютний шлях до директорії, яку ви збираєтесь використовувати за замовчуванням. Наприклад cd '/home/lexicus/Freemat'.

3. У вікні редактора натисніть File → Save, перейдіть у папку, яку ви вказали в команді **cd**, задайте ім'я програми startup.m і натисніть Save. Вікно редактора зображено на рис. 1.10.

4. Закрийте редактор FreeMat.

Рисунок 1.10 – Вікно редактора FreeMat

Тепер протестуємо нашу нову програму. Перезапустимо FreeMat. Якщо ваша поточна робоча директорія співпадає з тією, яку ви тільки що створили, то все пройшло успішно. Тепер всі дані, які ви будете зберігати в програмі FreeMat за замовчуванням, будуть розташовувати у даній директорії (поки ви явно не вкажете програмі іншу директорію).

Приклад. Зберігання файлу в робочу директорію. Даний приклад виконується на операційній системі Ubuntu Linux.

```
--> pwd
/home/lexicus/FreeMat
```

Нижче наведено короткий набір команд, які генерують множину випадкових чисел і записують їх у файл з іменем «Test.dat». Оскільки шлях до файлу не вказаний явно, він буде збережений у поточну робочу директорію – «/home/lexicus/FreeMat».

```
--> A=float(randn(512));
--> fp=fopen('test.dat','w');
--> fwrite(fp,A,'single');
--> fclose(fp);
```

1.5.2. Установка списку шляхів

Програмі FreeMat для роботи необхідно використовувати різні файли. У FreeMat є настройки під назвою **path**. Ця настройка містить в собі список шляхів, в яких ви можете зберігати дані, необхідні для роботи з FreeMat (програми та функції). FreeMat буде здійснювати пошук файлів у директоріях, які перелічені у змінній **path**.

Ви можете дізнатися перелік шляхів за допомогою команди **path:**

```
--> path
/home/lexicus/FreeMat
/home/lexicus/FreeMat/Data
/home/lexicus/FreeMat/FreeMat_Training
/home/lexicus/FreeMat/FreeMat_images
/home/lexicus/FreeMat/lexicus
/home/lexicus/FreeMat/myFunctions
/home/lexicus/FreeMat/myScripts
```

Path у програмі FreeMat – звичайна змінна, тобто ви можете, наприклад, присвоїти іншій змінній значення змінної path.

```
--> y=path
y=/home/lexicus/FreeMat:/home/lexicus/FreeMat/Data:/home/lexi
cus/FreeMat/FreeMat_Training:/home/lexicus/FreeMat/FreeMat_images
:/home/lexicus/FreeMat/lexicus:/home/lexicus/FreeMat/myFunctions:/ho
me/lexicus/FreeMat/myScripts
```

Попередження. Це стосується всіх операційних систем. Якщо у змінній path знаходиться лише один запис, FreeMat або видасть помилку при запиті списку шляхів (Linux), або поверне пустий рядок (Windows) при

виведенні команди path. Тим не менш FreeMat буде мати доступ до даної директорії при пошуку програм і функцій у даній директорії.

1.6. Вікно команд

Вікно команд є основним робочим вікном програми FreeMat. Користувач здійснює роботу з програмою FreeMat за допомогою командного рядка (поєднання символів «-->» у вікні команд визначає поточне положення введення). Ви будете використовувати вікно команд практично кругом: для базових операцій, таких як додавання та віднімання, або ж для складних обчислень, які потребують використання множини функцій. Програма FreeMat може використовуватися як калькулятор. Виконання будь-якої операції в командному вікні потребує введення самої команди та натискання на клавішу <Enter>.

Приклад введення команд.

Виконаємо додавання 3+2:

```
--> 3+2 <Enter>
ans =
 5
```

Помножимо 5 на 3.1415:

```
--> 5*3.1415
ans =
15.7075.
```

Коли ви натискаєте клавішу <Enter>, то FreeMat виконує обчислення, відображає результат (якщо ви спеціально не вказали програмі цього не робити, то як цього досягти буде описано нижче) і готує командний рядок для введення наступної операції.

1.6.1. Відображати або не відображати результат?

У кінці після введення команди можна ставити символ «;» (а можна и не ставити, FreeMat автоматично виконує команду після натискання на <Enter>). Таким чином, ви повідомляєте програмі FreeMat, що опис команди закінчено. Ви можете (за бажанням) розмістити декілька команд на одному рядку, розділяючи їх за допомогою символу «;».

Приклад. Розміщення декількох команд в одному рядку.

```
--> x=5.21; y=6.7; z=x*y
z =
34.9070
```

Хоч це не викличе ніяких технічних проблем, але писати декілька команд на одному рядку не рекомендується (це ускладнює процес читання і настройки програми, особливо якщо ваш програмний код буде використовуватись іншими користувачами).

За допомогою символу «;» ви також можете контролювати, показувати чи ні результат виконання команди. Якщо ви хочете побачити результат виконання команди миттєво, просто введіть команду без символу «;» наприкінці. Якщо ви не хочете миттєвого виведення результату, поставте після команди «;».

Приклад. Використання «;».

```
--> x=7*3
x =
21
--> x=7*3;
```

У будь-якому випадку результат обчислення один і той же. FreeMat проводить обчислення, зберігає результат у змінній *x*. Єдина відмінність, чи хочете ви бачити результат обчислень зараз. Використання «;» стає дійсно потрібним, коли ви починаєте писати свої власні програми, які складаються із декількох команд. У цьому випадку ви не будете завалені великим потоком результатів обчислень, які відображаються. Замість цього ви здійсните виконання сотні (або тисячі) команд, можливо з виведенням єдиного результату обчислень у самому кінці.

Навіть якщо ви використовуєте «;» в кінці команд, ви все ще можете побачити результат виконання команди. Цього можна досягти таким чином. Введіть назву змінної в командний рядок і натисніть на <Enter>. Використовуйте секцію Variables.

Приклад. Перегляд змінної з використанням командного рядка.

```
--> x=7*3;
--> x
ans =
21
```

1.6.2. Скільки значущих цифр після крапки необхідно виводити

FreeMat використовує команду **format** для настройки чисел при їх виведенні. Якщо ви хочете, щоб на екрані відображалось 14 цифр після коми, то введіть команду **format long**. Якщо потрібно відображати 4 значущих цифри після коми, введіть команду **format short**. За

замовчуванням FreeMat відображає 4 значущих цифри після коми (format short).

Приклад. Настройка форматування чисел із плаваючою крапкою.

```
--> format short
--> pi
ans =
3.1416
--> format long
--> pi
ans =
3.14159265358979
```

1.6.3. Змінні у FreeMat

Змінна у FreeMat – це символ або набір символів, який використовується для зберігання деяких даних. Ім'я змінної може складатися з латинських букв, цифр і/або символу підкреслення «_». Ім'я змінної повинно починатися з букви. Імена змінних у FreeMat чуттєві до регістру («a» і «A» – різні змінні).

У змінних можуть зберігатися:

– Числа;

– Рядки;

– Матриці із чисел, рядків і/або інших матриць. Матриці описані в розділі 3;

– Покажчик на функцію, який є анонімною функцією (детально описано далі в розділі 4.7).

Існує декілька типів для числових змінних. Є як знакові, так і без-знакові цілі числові типи змінних (беззнаковий тип може зберігати тільки додатні числові значення та значення 0), мовні змінні з одинарною та подвійною точністю, комплексні числа одинарної та подвійної точності. Для зберігання рядків передбачено один тип змінної – string.

Щоб привласнити змінній будь-яке значення, використовуйте знак рівності «=». Наприклад $x = 5$. У цьому випадку x – це змінна, а 5 – це значення, яке буде привласнене цій змінній.

Для перегляду використовуваних у даний момент змінних можна використати команду **who**. Просто наберіть у командному рядку **who** і натисніть на <Enter>.

```
--> x=1;
```

```
--> who
Variable Name      Type   Flags          Size
    x                  int32                  [1 1]
```

У даному випадку об’явлена тільки одна змінна. У випадку, якщо буде об’явлена більша кількість змінних, всі вони будуть перелічені нижче.

Приклад. Привласнення змінній значення. Щоб привласнити змінній значення, введіть ім’я змінної, знак рівності та значення, яке ви хочете привласнити змінній (число або рядок). Нижче наведено декілька прикладів привласнення числових і рядкових значень змінним із наступною їх візуалізацією за допомогою команди **who**:

```
--> x=5;
--> y=491.2768;
--> s='This is a string';
--> s2=['Part of a string ' num2str(2) ' part of another string']
s2 =
Part of a string 2 part of another string
--> who
 Variable Name      Type   Flags          Size
       ans          double                 [1 1]
        s            char                  [1 16]
       s2            char                  [1 41]
        x           double                 [1 1]
        y           double                 [1 1]
```

1.6.3.1. Типи змінних

У кожної змінної є свій тип. Тип змінної характеризує, що зберігає дана змінна, число або рядок та скільки біт відведено на зберігання даної змінної. Нижче наведено типи змінних, використовуваних FreeMat:

int8 – знакове 8-бітне ціле. Можливі значення від –128 до 127. Однойменна функція FreeMat є періодичною. Це означає, що якщо функції передається значення, яке виходить за межі (–128; 127), воно повертається до інтервалу. Наприклад int8(128) = 127 та int8(129) = 127.

Приклад. Тип змінної **int8**.

```
--> int8(53)
ans =
53
```

```
--> int8(130)
ans =
127
--> int8(33.98)
ans =
33
```

int16 – знакове 16-бітне число. Можливі значення від –32768 до 32767. Однойменна функція **int16** також є періодичною.

int32 – знакове 32-бітне число. Можливі значення від –2147483648 до 2147483647. Цей тип змінної є стандартним типом FreeMat для цілих чисел. Це означає, що якщо ви створюєте змінну числового типу, не вказуючи явно її тип, змінній буде привласнено тип **int32**.

Приклад. Створення цілочислової змінної без явної вказівки типу.

```
--> x=5;
--> who
Variable Name      Type   Flags          Size
        x           int32                [1 1]
```

int64 – знакове 64-бітне ціле. Можливі значення від -9223372036854775808 до 9223372036854775807.

uint8 – беззнакове 8-бітне ціле. Можливі значення від 0 до 255.

uint16 – беззнакове 16-бітне ціле. Можливі значення від 0 до 65535.

uint32 – беззнакове 32-бітне ціле. Можливі значення від 0 до 4294967295.

uint64 – беззнакове 64-бітне ціле. Можливі значення від 0 до 9223372036854775808.

float – знакове 32-бітне дійсне число. Можливі значення від 3.4e1038 до 3.4e1038.

double – знакове 64-бітне дійсне число. Можливі значення від -1.79e10308 до 1.79e10308. Цей тип є типом за умовчанням для дійсних чисел FreeMat.

complex – знакове 32-бітне комплексне дійсне число. Дійсна та уявна частини комплексного числа є дійсними числами одиничної точності.

dcomplex – знакове 64-бітне комплексне дійсне число. Також як і для типу complex дійсна та уявна частини комплексного числа типу dcomplex представляють собою дійсні числа подвійної точності.

string – будь-яка комбінація букв, чисел і/або спеціальних символів. Рядкова змінна може бути довжиною лише в один символ, максимальна довжина 65535 символів. Значення для рядкової змінної вводиться за допомогою одинарних лапок. Наприклад, a = 'hello' присвоює змінній “a” значення “hello”.

1.6.3.2. Бінарні типи

FreeMat може оперувати двійковими числами, однак це не може виконуватись. Двійкове число перетворюється в десяткове ціле подання, далі з цим числом можуть проводитися будь-які обчислення, для відображення результату число перетворюється в двійкове у вигляді рядка.

1.6.3.3. Відображення двійкових чисел

Двійкові числа можуть бути подані у вигляді вектора або рядка. Для подання цілих або десяткових чисел у двійковій системі обчислення FreeMat надає дві функції **int2bin** и **dec2bin**.

int2bin(x,n) – створює двійкове число. Проводиться розрахунок двійкового значення цілого числа *x* з використанням *n* розрядів для відображення. Значення, що повертається, являє собою вектор довжиною *n*.

dec2bin(x) – функція, що створює рядок, в якому міститься значення числа *x* у двійковому вигляді. Значення, яке повертається, не є двійковим числом, бо замість цього повертається рядок з відповідним значенням.

1.6.3.4. Операції з двійковими даними

FreeMat підтримує три операції з двійковими даними:

bitor

bitand

bitxor

1.7. Рядки

У програмі FreeMat рядком називають вектор, який складається з букв, цифр і спеціальних символів. Для того щоб створити рядок, помістіть любу комбінацію вище перелічених символів між одинарними лапками. Пробіл також є символом. FreeMat подає рядок у вигляді вектора [1 *n*], де *n* – це кількість символів у рядку. Одинарні лапки, в які укладено рядок, не є частиною рядка.

Приклад. Рядок у FreeMat.

```
--> clear all
--> a='here is a string';
--> b='yet another string to ponder';
--> c='hello';
--> who
Variable Name      Type   Flags          Size
        a           string                [1 16]
        b           string                [1 28]
        c           string                 [1 5]
```

1.7.1. Створення рядка

Ви можете створити рядок таким чином:

– помістити потрібні символи між одинарними лапками;

– за допомогою функцій **string** або **char**, для конвертування символів за їх ASCII кодами. ASCII коди будуть описані в підрозд. 4.8.4;

– за допомогою функції **num2str**, яка перетворює число в рядок;

– створити однорядкову матрицю, в яку помістити один або декілька рядків за допомогою одинарних лапок.

Приклад. Створення рядка.

```
--> x='It was a dark and stormy night.'
x =
It was a dark and stormy night.
--> y='To be or not to be.  That is the question.'
y =
To be or not to be.  That is the question.
--> x=[70 114 101 101 109 97 116];
--> string(x)
ans =
Freemat
--> x=['The temperature yesterday was 105' string(176) 'F.']
x =
The temperature yesterday was 105°F.
--> temp=105;
--> x=['The temperature yesterday was' num2str(temp) string(176)
'F.']
x =
```

```
The temperature yesterday was 105°F.
```

1.7.2. Конкатенація рядків

Щоб здійснити конкатенацію (об'єднання) двох і більше рядків, то помістіть їх в однорядкову матрицю.

Приклад. Конкатенація рядків.

```
--> s1='To be or not to be.';
--> s2='That is the question.';
--> s=[s1 s2]
s =
To be or not to be.That is the question.
--> s1='To be or not to be.';
--> s2='That is the question.';
--> s=[s1 ' ' s2]
s =
To be or not to be. That is the question.
--> s1='20';
--> s2='4';
--> s3='5';
--> s=[s1 string(247) s2 '=' s3]
s =
20ч4=5--> s1=20;
--> s2=4;
--> s3=s1/s2;
--> s=[num2str(s1) string(247) num2str(s2) '=' num2str(s3)]
s =
20ч4=5
```

1.8. Вбудовані змінні

Програма FreeMat зберігає значення деяких визначених змінних і констант. Найбільш корисні з них:

e – експонента, основа натурального логарифма. Значення змінної приблизно 2.718281828459045;

i – уявний оператор (використовується для комплексних чисел);

inf – нескінченність. Використовується тільки з дійсними типами даних. Якщо ви намагатиметесь використати **inf** з цілочисловими типами даних, то ви можете отримати непередбачувані результати;

pi – співвідношення довжини кола до його діаметра. Значення дорівнює 3.1415926...

ans – ця змінна використовується програмою FreeMat для відображення результатів, якщо ви не привласнюєте результат виконання команди якій-небудь змінній. Тобто, якщо ви вкажете змінну для присвоєння, наприклад "*x* = 1+1", результат розрахунку буде привласнений змінній *x* (а не ans).

Приклад. Вбудовані змінні.

```
--> format long
--> e
ans =
2.71828182845905
--> i
ans =
0.0000000 + 1.0000000i
--> inf
ans =
1.#INF0000000000
--> pi
ans =
3.14159265358979 --> 5*pi
ans =
   15.7080
--> 1:10
ans =
  1  2  3  4  5  6  7  8  9 10
--> 1:3:13
ans =
  1  4  7 10 13
--> 9/11
ans =
    0.8182
-->  --> 9/11
ans =
    0.8182
--> ans*4.5
```

```
ans =
   3.6818
```

1.9. Використання вбудованих функцій

Програма FreeMat надає велику кількість вбудованих функцій. У FreeMat функція являє собою алгоритм або процес, який перетворює вхідні параметри у визначений результат. Наприклад, функція **sin**, яка приймає як аргумент один параметр – *x* і обчислює значення синуса кута *x*. Після чого функція повертає результат своєї роботи. Даний результат може бути числом, як у випадку з функцією синус, або це може бути зображення, як наприклад з функцією **plot**. Нижче перелічені основні моменти для правильної роботи з функціями.

Які аргументи потрібні функції та який порядок їх задання? Наприклад візьмемо функцію **cumsum(x,d)**. Аргумент *x* повинен бути масивом, *d* – розмірність масиву. При введенні функції дуже важливо ввести правильні аргументи у правильному порядку. У випадку, якщо ви введете функцію у вигляді **cumsum(d,x)**, FreeMat згенерує повідомлення про помилку або ви отримаєте неправильний результат.

Чи повинні аргументи функції бути визначеного типу? Деякі функції приймають тільки цілі, інші приймають числа з плаваючою точкою або рядки. Наприклад, якщо ви намагатиметесь назвати функцію sin('string'), то ви обов'язково отримаєте повідомлення про помилку. Функція sin як аргумент приймає дійсне число, а не рядок. Для уточнення типів аргументів функції звертайтеся до документації програми FreeMat.

Чи повинні аргументи функції відповідати визначеній розмірності? Наприклад, тригонометричні функції **sin**, **cos** і **tan** потребують, щоб їх аргументи (кути) були задані в радіанах. Якщо ви введете функцію із значенням аргументу 30 °, ви не отримаєте очікувану відповідь 0.5.

Якого типу буде значення, яке повертається функцією? Наприклад, якщо ви виконуєте такі команди:

```
t=1:128;
x=sin(2*pi*t/32);
y=plot(x)
```

Ви отримаєте графік функції *y* і таке повідомлення в командному вікні:

```
y =
```

100002

Що означає *y* = 100002 у документації не описується. Однак функція **plot** повертає графічне представлення, вона не повинна повертати число (ймовірно, що це число показує нам ідентифікатор графічного подання).

Чи існує така функція? Для перевірки необхідно використовувати команду **which**. Дана команда повертає або шлях до функції, або повідомлення, що такої функції немає. Нижче наведено декілька прикладів. У першому прикладі ми перевіряємо функцію FreeMat, другий приклад перевіряє функцію, якої не існує. Третій приклад перевіряє створену користувачем функцію.

```
--> which diff
Function diff, M-File function in file
'/usr/share/freemat/toolbox/general/diff.m'
--> which myFunc
Function myFunc is unknown!
--> which labelSet
Function labelSet, M-File function in file
'/home/lexicus/FreeMat/myScripts/labelSet.m'
```

Для ознайомлення із вбудованими функціями зверніться до офіційної документації FreeMat.

2. МАТЕМАТИЧНІ ОПЕРАЦІЇ

2.1. Базові математичні операції

Базові математичні операції включають в себе додавання (+), віднімання (–), множення (*), ділення (/) і піднесення до степеня (^). Операція піднесення до степеня більш детально викладена в розд. 2.4.

Приклади базових математичних операцій:

```
--> 5.4+3.8
ans =
9.2000
--> 5.4–3.8
ans =
1.6000
--> 5.4*3.8
ans =
20.5200
--> 5.4/3.8
ans =
1.4211
--> 5.4^3.8
ans =
606.8709
```

Зверніть увагу, що при виконанні математичних операцій ми не використовуємо знак рівності. Єдиний випадок, коли вам треба використовувати знак рівності, це коли ви зберігаєте результат виконання у змінній.

2.2. Порядок виконання операцій

При виконанні математичних операцій програма FreeMat використовує стандартну для математики систему пріоритету операцій. Це значить, що FreeMat виконує спочатку одні математичні операції, а потім – інші. Система пріоритету така:

1. Виконання дій в круглих дужках – (…);
2. Піднесення до степеня;
3. Множення та ділення;
4. Додавання та віднімання.

Це означає, що залежно від способу запису виразу ви можете отримати різні результати. Наприклад:

```
--> 4+5*3
ans =
 19--> (4+5)*3
ans =
 27
```

У першому випадку програма FreeMat згідно з правилами пріоритетів обчислення виконує множення (5*3) до додавання. Тобто, здійснюючи перемноження та отримуючи результат 15, ми потім додаємо до нього 4. У результаті в першому випадку отримуємо відповідь 19.

У другому випадку програма FreeMat спочатку виконує складання (через наявність круглих дужок). Таким чином, спочатку ми знаходимо суму 4 і 5, в результаті отримуємо 9 і результат множимо на 3. Кінцевий результат у даному випадку – 27.

Приклад. Пріоритети операцій та додатні числа.

Відомо, що від'ємне число, будучи помноженим на від'ємне число, дає додатній результат. Однак необхідно пам'ятати, що саме програма FreeMat розцінює як від'ємне число. Ось декілька прикладів:

```
--> –1*–5
ans =
 5
```

У даному випадку ми помножили –1 на –5 і отримали 5. Те, що ми очікували. У наступному прикладі давайте візьмемо –5 в квадрат.

```
--> –5^2
ans =
 –25
```

У результаті отримали –25, а не 25, як очікували. Це відбувається тому, що програма FreeMat спочатку здійснює піднесення до степеня (5^2), а потім віднімання (операція піднесення до степеня має більший пріоритет ніж віднімання). Щоб це виправити, необхідно скористатися круглими дужками, тоді програма FreeMat зведе в квадрат число –5.

```
--> (–5)^2
ans =
 25
```

Щоб уникнути подібних неприємних помилок, рекомендується використовувати круглі дужки зі всіма від'ємними числами. Таким чином, ви будете впевнені, що FreeMat правильно виконує операції над від'ємними числами (так, як ви плануєте).

2.3. Сума, добуток, накопичення суми та добутку, факторіали

У програмі FreeMat є декілька функцій, які дозволяють вам ефективно обчислювати суму, добуток, накопичення суми та добутку масиву чисел, а також факторіал числа. Нижче перелічені дані функції:

sum(x) – повертає суму всіх елементів масиву *x*. Результатом виконання функції є скалярне число;

prod(x) – повертає добуток всіх елементів масиву *x*. Результатом виконання функції є скалярне число;

cumsum(x) – повертає накопичену суму масиву *x*. Результатом виконання даної функції є масив, кожен елемент якого є сумою всіх елементів, які стоять перед поточним і самим поточним елементом в оригінальному масиві (далі буде наведено приклад);

cumprod(x) – повертає накопичений добуток масиву *x*. Результатом виконання даної функції є масив, кожен елемент якого дорівнює добутку всіх елементів, розташованих до нього, помноженого на поточний елемент оригінального масиву (далі буде показано приклад);

gamma(x) – інтегральна функція. Ми будемо обговорювати дану функцію трохи нижче. Тим не менш, вона може застосовуватись для обчислення факторіала цілого числа. З допомогою цієї функції факторіал обчислюється таким чином: $x! = \text{gamma}(x+1)$.

Приклад. Накопичена сума, добуток і функція факторіала:

```
x =
 1 2 3 4 5
--> y = cumsum(x)
y =
  1  3  6 10 15
--> y = cumprod(x)
y =
   1   2   6  24 120
--> gamma(5+1)
ans =
```

```
 120.0000
```

2.4. Експоненти та логарифми

З числами можна проводити операції піднесення до степеня. Наприклад, для x^2, 2 визначає степінь, а x є основою. Є два способи піднесення числа до степеня. Перший спосіб складається з використання символу «^». Другий спосіб – використання функцій піднесення до степеня. Використовувати символ «^», очевидно, простіше. Просто пишіть число, яке хочете піднести до степеня, потім символ «^», а слідом за ним число – степінь. Якщо у вас є масив і ви хочете кожен з його елементів піднести до степеня, використовуйте спеціальне поєднання символів – «.^» (поставте перед «^» символ точку). Нижче наведений синтаксис функції піднесення до степеня:

power(x,y),

де x – основа, яка буде піднесена до степеня; y – степінь.

Приклад. Піднесення числа або змінної до степеня.

```
--> x=5.6;
--> x^2.5
ans =
74.2113
--> 8^3
ans =
512
--> power(8,3)
ans =
512
--> power(2,5)
ans =
 32
--> power(1:10,2)
ans =
   1   4   9  16  25  36  49  64  81 100
```

Остання операція також може бути виконана за допомогою оператора «.^».

```
--> (1:10).^2
ans =
```

```
  1   4   9  16  25  36  49  64  81 100
```

Програма FreeMat надає такі функції для здійснення операцій піднесення до степеня числа *e* (експонента):

exp(x) – ця функція виконує теж саме, що і операція *e^x*. У чому ви запитаєте різниця? У ранніх версіях FreeMat дані операції видавали різницю в 14 знаку після крапки. У більш пізніх версіях програми дана відмінність відсутня. Використовуйте форму запису, яка більш прийнятна для вас;

expm1(x) – ця функція дуже корисна при обчисленні exp(x)–1, коли x – мала величина. Це один із випадків, коли краще використовувати вбудовану функцію, ніж писати простий вираз.

Приклад. Функції для роботи з експонентою.

```
--> exp(4)
ans =
54.5982
--> e^4
ans =
54.5982
--> expm1(0.0001)
ans =
1.0001e-004
```

Логарифми. Логарифмічна функція – це спосіб обчислити степінь, до якого необхідно піднести число (основу логарифма), щоб отримати інше задане число (аргумент логарифма). Найбільш часто як основу логарифма використовують числа 10 і *e*. Нижче подані функції FreeMat для знаходження логарифмів:

log10(x) – обчислює логарифм числа *x* за основою 10;

log2(x) – обчислює логарифм числа *x* за основою 2;

log(x) – обчислює логарифм числа *x* за основою *e*. Даний логарифм дуже часто називають натуральним логарифмом числа *x*;

log1p(x) – обчислює логарифм числа *x*+1 за основою *e*.

Приклад. Логарифмічні функції.

```
--> log10(10)
ans =
1
--> log10(25)
```

```
ans =
1.3979
--> log(10)
ans =
2.3026
--> log(e)
ans =
1
--> log2(2)
ans =
1
--> log2(16)
ans =
4
--> log2(10)
ans =
3.3219
```

Обчислення логарифма числа за довільної основи. Для обчислення логарифма числа за довільної основи у математиці є спеціальний спосіб. Логарифм числа a за основою b обчислюється за такою формулою:

$$\log_b a = \frac{\ln a}{\ln b}.$$

Приклад обчислення логарифма за довільною основою. Обчислимо логарифм числа 5 за основою 3.

```
--> format long
--> log(5)/log(3)
ans =
1.46497352071793
```

Обчислимо логарифм числа 16 за основою 6.2.

```
--> log(16)/log(6.2)
ans =
1.51960198297685
--> log10(16)/log10(6.2)
ans =
1.51960198297685
```

Тепер покажемо, що даний алгоритм дає той же результат, що і функції FreeMat для стандартних основ.

```
--> log(5)
ans =
1.60943791243410
--> log10(5)/log10(e)
ans =
1.60943791243410
```

2.5. Тригонометричні функції

Програма FreeMat має повний набір тригонометричних функцій: стандартні та гіперболічні тригонометричні функції, зворотні стандартні та зворотні гіперболічні тригонометричні функції. Також програма FreeMat містить функції, які як вхідний аргумент приймають значення в градусах, а не радіанах.

Таблиця 2.1 – Тригонометричні функції

Стандартні (радіани)	Зворотні стандартні (радіани)	Гіперболічні	Зворотні гіперболічні	Стандартні (градуси)	Зворотні стандартні (градуси)
sin()	asin()	sinh()	asinh()	sind()	asind()
cos()	acos()	cosh()	acosh()	cosd()	acosd()
tan()	atan()або atan2()	tanh()	atanh()	tand()	atand()
sec()	asec()	sech()	asech()	secd()	asecd()
csc()	acsc()	csch()	acsch()	cscd()	acscd()
cot()	acot()	coth()	acoth()	cotd()	acotd()

Відповідно, значення кутів для стандартних функцій необхідно задавати в радіанах. Зворотні стандартні функції повертають значення кутів у радіанах. Гіперболічні функції оперують з безрозмірними величинами. Набір стандартних тригонометричних функцій для роботи з градусами відповідно оперує градусами.

3. МАТРИЦІ ТА МАСИВИ

Матриця – це прямокутний масив, який складається із чисел та рядків. У FreeMat кожне число – матриця. Матриця самий основний елемент FreeMat. (Див. FD, с. 49). Одне число, наприклад 5.342, матриця розміром [1 1], де перше число – кількість рядків, а друге число – кількість стовпців. Тому, якщо ви зберігаєте одну змінну, вона буде збережена розміром [1 1]. Це скаляр. Ось приклад використання команди **who:**

```
--> y=5.342;
--> who
Variable Name Type Flags Size
        y       double      [1 1]
```

Або ви можете використати команду **size:**

```
--> y=5.342;
--> size(y)
ans =
1 1
```

Ось ще один спосіб подивитись на це. Ми просто привласнюємо значення 5.342 змінній "y". Якщо ми хочемо перевірити її значення, ми просто набираємо "y" і натискаємо на кнопку введення, а саме:

```
--> y
ans =
5.3420
```

Ви можете використовувати індекси для перегляду кожного елемента матриці. Індекси розміщують у дужках ( ) після змінної, яка містить матрицю. З нашим поточним значенням «*y*» ми можемо використовувати індекс масиву, щоб проглядати його значення, а саме:

```
--> y(1)
ans =
5.3420
```

Як відзначалося раніше, одне значення змінної – це матриця розміром [1 1]. Таким чином, ми можемо також використовувати подвійне значення індексу, а саме:

```
--> y=5.342;
--> y(1,1)
```

```
ans =
5.3420
```

Зверніть увагу, що коли ви хочете подивитись на елемент багатовимірної матриці, ви повинні поставити кому між кожним виміром.

Двовимірний масив подається у форматі [рядок стовпець]. Така матриця являє собою масив, тобто вона містить елементи, відсортовані за рядками та стовпцями. Опис матриці, масиву, вектора і скаляра наведено в таблиці 3.1.

Таблиця 3.1 – Опис матриці, масиву, вектора та скаляра

Тип	Опис
Матриця	Пронумерований прямокутний масив
Масив	Згруповані елементи (числа, рядки, інші масиви або покажчики на функції), які можуть мати одне або декілька вимірів. Двовимірні масиви – матриці
Вектор	Вектор складається з одного рядка або одного стовпця та двох або більше елементів різних розмірів. Наприклад, рядок із 10 значень буде розглядатися як вектор довжини 10
Скаляр	Це одиничне число тільки з одним стовпцем і одним рядком

Таким чином, матриця показана розміром [2 3] означає, що у неї є 2 рядки та 3 стовпці. Простіше щоб було зрозуміло, рядок означає “елементи за горизонталлю”, як показано на рис. 3.1.

Рисунок 3.1 – Вигляд матриці. У даному конкретному прикладі матрицярозміром [4 3], таким чином, вона має 4 рядки і 3 стовпці. Точка, в якій перетинаються виділені рядок і стовпець (2,1), що означає 2-й рядок і 1-й стовбець

3.1. Матриці, вектори та індексування

У багатьох математичних мовах програмування, наприклад FreeMat, існують різні способи для зберігання значень змінних. Як ми вже обговорювали раніше, змінна може містити числа, рядки або покажчик на функцію. У FreeMat є два різних способи подивитися на те, як ці змінні організовані для зберігання значень. Перший – це матриці. Його використовують для зберігання букв або груп букв (тобто рядків), або набору групи чисел. Разом з тим матриці не можуть містити те й інше одночасно. Другий – клітинки (cell). Клітинка може складатися або із рядка, числа, або того та іншого одночасно. Зараз настав час, коли нам треба обговорити різні типи дужок (Табл. 3.2).

Таблиця 3.2 – Види дужок та їх тлумачення

Тлумачення дужок	
[ ] (квадратні дужки)	Набір квадратних дужок використовують для оголошення матриці або вектора. Наприклад: --> x=[4 5 2 1 6 0] x = 4 5 2 1 6 0 Другий приклад для декількох рядків. Ми використовуємо крапку з комою (;) для розділення рядків. --> x=[5 1 8 3 4 7 6;3 6 1 9 7 5 0] x = 5 1 8 3 4 7 6 3 6 1 9 7 5 0
( ) (круглі дужки)	Набір круглих дужок ми використовуємо для індексування матриці або вектора. Наприклад --> x=[4 5 2 1 6 0] x = 4 5 2 1 6 0 --> x(2) ans = 5 Індекс операторів не потребує використання змінної. Хоча ви не можете помістити індекс безпосередньо з самого масиву. Ви можете помістити індекс безпосередньо на

Продовження табл. 3.2

	використовувану функцію, створивши масив. --> x=linspace(0,2*pi,10); --> sin(x)(4) ans = 0.8660 Знову ж таки, ви не можете використовувати індекс оператора безпосередньо на масив, як показано в прикладі --> [7 1 8 5 2 3 9](3) Error: Unexpected input [7 1 8 5 2 3 9](3) ^
{} (фігурні дужки)	Фігурні дужки означають масив клітинок або можуть бути використані для індексації масиву клітинок. Наприклад: --> y={'this is a test' [4 5 2 1 6 0] rand(4,4)} y = [this is a test] [1 6 double array] [4 4 double array] --> y{2} ans = 4 5 2 1 6 0
; (точка з комою)	У масиві клітинок або матричному масиві точка з комою використовується для оголошення різних рядків. Ось приклад: --> y={'this is a test';[4 5 2 1 6 0];rand(4,4)} y = [this is a test] [1 6 double array] [4 4 double array] --> y{2} ans = 4 5 2 1 6 0

Продовження табл. 3.2

, (кома)	Існує два способи для позначення стовпців у масиві. Ви можете поставити пробіл між елементами або кому. Залежно від ваших уподобань
: (двокрапка)	Двокрапка має спеціальне позначення. Вона може бути використана для вказівки розмірів матриць або масивів клітинок
end	Слово "end" має спеціальне значення. Воно може бути використано для вказівки останнього елемента масиву. Наприклад, використаємо **cumsum** (кумулятивну суму), як показано нижче: --> x=1:50; --> y=cumsum(x); --> y(end) ans = 1275 Це можна записати коротше: --> x=1:100; --> y=cumsum(x)(end) y = 5050 Або ще коротше: --> cumsum(1:100)(end) ans = 5050

3.2. Створення масиву послідовності

Матриці або масиви послідовностей можуть бути дуже корисними при програмуванні. Ви можете використовувати їх як лічильник або для зберігання послідовності змінних в одновимірному масиві.

Щоб створити простий масив можна використовувати двокрапку (:). За замовчуванням оператор двокрапки використовує розмір кроку 1 між початком і кінцем масиву. Наприклад, щоб створити послідовний масив, який йде від 1 до 10 з кроком 1, використовуйте таку команду:

t=1:10.

Примітка. Це раціональна ідея використовувати крапку з комою у кінці рядка при створенні масиву. Якщо ви створюєте великий масив і не використовуєте його, то FreeMat відображає змінну. Ви отримаєте повний екран непотрібних номерів.

Ви також можете створити масив із вказаним розміром кроку. Наприклад, якщо ви хочете створити послідовність від 1 до 10 з кроком 3, використовуйте такий формат:

```
t=1:3:10.
```

Є дві функції, які можна використовувати для створення масивів. Ця функція **linspace** і функція **logspace**. **Linspace** створює масив між двома числами.

Приклад. Створення масиву послідовності.

Перший приклад використання оператора двокрапки для створення масиву від 1 до 10.

```
t=1:10
t =
1 2 3 4 5 6 7 8 9 10
```

Тепер подивимось на аналогічну матрицю, але створену за допомогою команди **linspace**.

```
t=linspace(1,10,7)
t =
1.0000 2.5000 4.0000 5.5000 7.0000 8.5000 10.0000
```

Це означає, що ми створюємо лінійний масив, який пробігає значення від 1 до 10 у 7 кроків. Якщо ви не вкажете кількість кроків, **linspace** використовує значення за замовчуванням 100 кроків.

Функція **logspace** дещо інша. Вона створює лінійний масив між мінімальною та максимальною точками, визначеними у функції. Створюється масив від 10^(мінімум) до 10^(максимум). Це краще пояснює приклад.

Приклад створення спеціального логарифмічного масиву за допомогою функції **logspace**.

```
logspace(1,10,10)
ans =
10 100 1000 10000 100000 1000000
10000000 100000000 1000000000 10000000000
```

Зверніть увагу, що розрахунок масиву починається з 10^1 і закінчується 10^{10}. Якщо ми хочемо створити масив із логарифмічним розташуванням величин між двома значеннями конкретних цифр, ми повинні об'єднати **logspace** із функцією **log10**, як показано нижче:

```
logspace(log10(1),log10(10),10)
ans =
1.0000  1.2915  1.6681  2.1544  2.7826  3.5938  4.6416  5.9948
7.7426  10.000
```

3.3. Створення масиву випадкових величин

FreeMat має багато функцій для створення матриць, заповнених випадковими величинами. До них належать:

rand – створює випадкову величину, рівномірно розташовану в діапазоні від 0 до –1;

randbeta(alpha, beta) – створює випадкову величину з бета розподіленням. Потребує параметри, **alpha** и **beta**, встановлює вид функції розподілення ймовірності;

randi(low, high) – створює масив випадкових цілих чисел між значеннями **low** і **high** (включно);

randn – створює масив випадкових величин з нормальним (Гауссовим) розподіленням.

Це лише деякі із множини способів створення випадкового масиву.

При створенні випадкового масиву ви повинні визначитись із кількістю рядків і стовпців. Причина в тому, що за замовчуванням FreeMat буде використовувати одне значення, внесене до функції для рядків і стовпців, як показано тут:

```
x=rand(10000);
size(x)
ans =
10000 10000
```

Візьміть до уваги, якщо ви хочете створити масив із одного рядка та 10000 стовпців, використовуйте такий формат:

```
x=rand(1,10000);
size(x)
ans =
1 10000
```

Ми також можемо використати гістограму, щоб побачити розподілення значень.

*Примітка! Помилка функції **hist**. На момент написання посібника існувала проблема з функцією **hist.** Вона не працювала правильно. Поки це не буде виправлено ви можете використовувати функцію, розташовану в Додатку А, зберігши його як **hist.m** в одній із папок на вашому шляху. Вона повинна забезпечити правильне функціонування, таке ж як у FreeMat документації.*

Використовуючи функцію **hist**, ми бачимо розподілення значень, створених за допомогою функції **rand** і **randn**:

```
x=rand(1,10000);
y=hist(x)
y =
1042  1020  1019  1001  977  1022  914  1005  1041  957
y=hist(x,10,1)
y =
0.1042  0.1020  0.1019  0.1001  0.0977  0.1022  0.0914
0.1005 0.1041 0.0957
plot(y)
ylim([0,0.2])
```

Результат роботи програми показано на рис. 3.2.

Рисунок 3.2 – Гістограма значень, побудованих за допомогою функції **rand**

3.4. Перегляд значень функцій

Щоб побачити значення окремого елемента матриці, використовуйте такий формат:

x(d1,d2,...,dn)

де: *x* = ім'я змінної; *d1* = перша розмірність матриці; *d2* = друга розмірність матриці; *dn* = *n*-а розмірність матриці.

Приклад. Перегляд одного елемента матриці.

```
x=rand(3,3)
x =
    0.4421    0.9413    0.5842
    0.3209    0.1061    0.7719
    0.2505    0.3724    0.7086
x(1,2)
ans =
        0.9413
x(2,1)
ans =
        0.3209
```

Щоб побачити повний розмір матриці, використовуйте двокрапку (:). Формат:

x(:,:)

де x = ім'я змінної; : = розмірність, яку ви хочете бачити.

Приклад перегляду повного розміру матриці.

```
x=rand(3,3)
x =
    0.4421    0.9413    0.5842
    0.3209    0.1061    0.7719
    0.2505    0.3724    0.7086
x(:,1)
ans =
    0.4421
    0.3209
    0.2505
x(1,:)
ans =
    0.4421    0.9413    0.5842
```

3.5. Матриці

У даному підрозділі ми зупинимось на тому, як FreeMat виконує різні операції із будь-якими матрицями.

3.5.1. Додавання матриць

При додаванні матриць результатом буде матриця, яка є сумою елементів матриць на поелементній основі. Це означає, що елементи першої матриці будуть підсумовуватись із тими ж елементами другої матриці (Рис. 3.3).

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} + \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} = \begin{pmatrix} a_{11}+b_{11} & a_{12}+b_{12} \\ a_{21}+b_{21} & a_{22}+b_{22} \end{pmatrix}$$

Рисунок 3.3 – Додавання матриць

Примітка! Додавання матриць потребує, щоб матриці були однакового розміру. Це означає, що кожна матриця повинна мати однакову кількість елементів в одному й тому вимірі.

Приклад додавання матриць.

```
x=rand(2,2)
x =
    0.3759    0.9134
    0.0183    0.3580
y=rand(2,2)
y =
    0.7604    0.0990
    0.8077    0.4972
x+y
ans =
    1.1363    1.0124
    0.8260    0.8552
```

3.5.2. Віднімання матриць

При відніманні аналогічно, як і при додаванні, матриці повинні мати однаковий розмір і однакову кількість елементів у кожному вимірі. Кожен елемент другої матриці віднімається із відповідного елемента в першій матриці.

Приклад віднімання матриць.

```
x=rand(3,3)
```

```
x =
    0.7457    0.9862    0.9445
    0.3774    0.0484    0.4942
    0.7623    0.0395    0.0089
y=rand(3,3)
y =
    0.4399    0.3236    0.9954
    0.9112    0.5196    0.5407
    0.6565    0.1273    0.0840
x–y
ans =
    0.3058    0.6626   –0.0509
   –0.5338   –0.4713   –0.0465
    0.1058   –0.0878   –0.0751
```

3.5.3. Множення матриць

Є два шляхи множення матриць у FreeMat. Перший стандартний, шлях лінійної алгебри. У цьому методі двовимірна матриця (x, y) множиться на іншу двовимірну матрицю (y, z), у результаті буде створена матриця (x, z). Наприклад, якщо перша матриця (10,3), а друга (3,7), в результаті отримаємо матрицю (10,7).

При виконанні стандартного множення множаться значення двох матриць так, як заведено в алгебрі. Пам'ятайте, що другий розмір першої матриці повинен мати таку ж кількість елементів, що й перший розмір другої матриці.

Приклад стандартного множення матриць.

```
x=rand(3,5);
y=rand(5,6);
z=x*y;
who
Variable   Name     Type     Flags     Size
                x     double              [3 5]
                y     double              [5 6]
                z     double             [3 6]
```

Другий метод дозволяє множити матриці на поелементній основі так, як при додаванні та відніманні. Це потребує додаткових символів у

командному рядку. Замість * вам потрібно використовувати .* як оператора. Таким чином, операція буде виглядати таким чином:

```
x .* y
```

Ця операція помножить кожен елемент матриці x на відповідний елемент матриці *y*. Зверніть увагу на те, що цю операцію можна виконувати на матрицях із будь-яким розміром, тобто вони мають однакову кількість вимірів і ту ж кількість елементів у кожному вимірі.

Приклад поелементного множення матриць.

```
x=rand(1,5)
x =
    0.7202    0.1487    0.7576    0.6396    0.2645
y=rand(1,5)
y =
    0.5725    0.5640    0.4085    0.8969    0.4665
x .* y
ans =
    0.4123    0.0839    0.3095    0.5737    0.1234
```

3.5.4. Ділення матриць

3.5.4.1. Ділення матриці на число

Ви можете розділити матрицю на число дійсне або уявне. Тут просто ділиться кожен елемент матриці на число.

Приклад ділення матриці на число.

```
y
ans =
    0.9132    0.0756    0.6215
    0.0204    0.3774    0.4662
    0.2899    0.4398    0.1450
y/5
ans =
    0.1826    0.0151    0.1243
    0.0041    0.0755    0.0932
    0.0580    0.0880    0.0290
```

3.5.4.2. Ділення матриці на іншу матрицю

Ви знаєте із курсу математики, що не існує такого поняття, як ділення матриць. Це означає, що ви не можете фактично розділити одну матрицю на іншу. Але ви можете сказати: “Але в FreeMat, я можу створити дві матриці, а потім розділити одну на іншу”. Так. Це правильно. Але це не випадок поділу однієї матриці на іншу. Це той випадок, коли одна матриця множиться на зворотну іншої матриці. Для того щоб це було здійснено, матриці повинні мати певні властивості:

1. Матриці повинні бути квадратними. Це означає, що вони повинні мати однакову кількість рядків і стовпців. Це означає, що допускаються матриці тільки 2x2, 3x3, 4x4 і тому подібні, а матриці 2x3 або 3x2 не допускаються.

2. Матриця повинна бути оборотною. Метод ділення однієї матриці на іншу вимагає обертання матриць. Але якщо матриця необоротна, операція не вдасться. Для того щоб вона було оборотною, визначник матриці не повинен дорівнювати нулю.

Якщо матриці відповідають цим двом критеріям, то FreeMat виконує операцію обертання матриці. Давайте розглянемо приклад.

Приклад множення матриці на зворотну матрицю.

```
x=rand(3,3)
x =
    0.0603    0.4750    0.1794
    0.5541    0.6298    0.6902
    0.9326    0.4791    0.8172
y=rand(3,3)
y =
    0.3334    0.7200    0.3153
    0.5322    0.0361    0.1913
    0.2064    0.1283    0.9270
x/y
ans =
     0.6701   –0.3186    0.0313
     0.7843    0.3961    0.3961
     0.5233    1.2519    0.4452
```

Здається, наче розділили одну матрицю *x*, на іншу *y*. Але це не так. Замість цього FreeMat матрицю оборотну *y* помножив на *x*. У цьому

можна переконатись показуючи проміжний крок розрахунку зворотної матриці *y*:

```
z=y^–1
z =
    –0.0294     2.0584     –0.4148
     1.4898    –0.8009     –0.3415
    –0.1996    –0.3476      1.2184
x*z
ans =
     0.6701    –0.3186      0.0313
     0.7843     0.3961      0.3961
     0.5233     1.2519      0.4452
```

3.5.4.3. Обчислення зворотної матриці

Для обчислення зворотної матриці просто використовуйте показник функції (^) та привласніть матриці степінь –1.

Приклад обчислення зворотної матриці. У цьому прикладі створюється квадратна матриця, а далі знаходиться зворотна їй.

```
y=rand(3,3)
y =
    0.9132    0.0756    0.6215
    0.0204    0.3774    0.4662
    0.2899    0.4398    0.1450
y^–1
ans =
     0.7923   –1.3832    1.0506
    –0.6969    0.2520    2.1773
     0.5295    2.0016   –1.8086
```

3.5.5. Поелементні операції з матрицями

У FreeMat є можливість виконувати множення, ділення та зведення в ступінь функцій матриць на поелементній основі. Для виконання таких поелементних операцій кожна матриця повинна бути одного розміру. Наприклад, матриця 2x3 потребує матрицю розміром 2x3.

Для перевірки розміру матриці ви можете:

1. Використовувати команду **who** або команду **size**, щоб побачити розмір змінних;

2. Використовувати вкладку "**Workspace**" біля командного вікна, щоб побачити розмір змінних.

Приклад. Помилка при виконанні поелементних матричних операцій. Цей приклад показує, що відбувається, коли ви намагатиметеся виконати поелементні операції з двома матрицями невідповідних розмірів.

```
x=rand(100,1);
t=1:100;
who
Variable Name      Type      Flags      Size
                t     int32              [1 100]
                x    double              [100 1]
x .* t
Error: Size mismatch on arguments to arithmetic operator .*
```

У цьому випадку матриці мають однакову кількість елементів, але змінна t має розмір [1 100], а змінна *x* розмір [100 1]. Щоб зробити їх однакового розміру, ви можете використовувати команду **transpose,** транспонувати одну із них та вирівняти їх належним чином.

```
x=transpose(x);
who
Variable Name      Type      Flags      Size
             ans     double               [0 0]
               t      int32              [1 100]
               x     double              [1 100]
```

Зверніть увагу, що розмірність та розмір однакові. Таким чином, можна переходити до обчислень

```
z=x .* t;
who
Variable Name      Type      Flags      Size
             ans     double               [0 0]
               t      int32              [1 100]
               x     double              [1 100]
               z     double              [1 100]
```

Зверніть увагу, що такі операції завжди виконуються поелементно:

– множення, ділення, додавання і віднімання матриць одного розміру;

– додавання і віднімання двох матриць.

При виконанні поелементного множення або поділу двох матриць перед математичним оператором ставиться крапка. Зокрема:

– множення: .*;

– поділ: ./;

– піднесення до степеня .^.

Приклад поелементного множення двох матриць:

```
x
ans =
      0.0603    0.4750    0.1794
      0.5541    0.6298    0.6902
      0.9326    0.4791    0.8172
y
ans =
      0.3334    0.7200    0.3153
      0.5322    0.0361    0.1913
      0.2064    0.1283    0.9270
x .* y
ans =
      0.0201    0.3420    0.0566
      0.2949    0.0228    0.1320
      0.1925    0.0615    0.7576
```

Приклад поелементного поділу двох матриць:

```
x
ans =
      0.0603    0.4750    0.1794
      0.5541    0.6298    0.6902
      0.9326    0.4791    0.8172
y
ans =
      0.3334    0.7200    0.3153
      0.5322    0.0361    0.1913
      0.2064    0.1283    0.9270
```

```
--> x ./ y
ans =
        0.1809    0.6597    0.5688
        1.0411   17.4284    3.6083
        4.5176    3.7355    0.8816
```

Приклад поелементного піднесення до степеня:

```
x=rand(1,10)
x =
  0.7314    0.9302    0.5387    0.0270    0.5100    0.8202
0.7037    0.7831    0.1695 0.3414
x.^2
ans =
  0.5349    0.8653    0.2902   0.0007    0.2601    0.6727
0.4952    0.6132    0.0287    0.1165
```

4. ПРОГРАМИ ТА ФУНКЦІЇ

Програми та функції у FreeMat є набором послідовних команд. Ви створюєте файл, в якому міститься набір команд, які виконуються послідовно одна за одною. Виконання програми або функції починається з першої команди в першому рядку та проходить через кожен рядок з командами в ньому. По закінченню виконання останньої команди інтерпретатор FreeMat зупиняється. Програми та функції є сценаріями FreeMat для виконання. Різниця між програмою та функцією полягає в тому, що виконання програми еквівалентно виконанню цих же команд в основному командному вікні FreeMat, в той же час функція має свої локальні змінні.

Ви можете використовувати ті ж імена змінних у функції, які ви використовували в командному вікні FreeMat або програмі, або навіть в іншій функції. Змінні, які використовуються в функції, є локальними для цієї функції. Змінні, які використовуються в програмі, це глобальні змінні. Це означає, що вони доступні для інших програм або команд, які набираються в основному вікні FreeMat. Введення команд у файл із розширенням .m еквівалентно набору цих же інструкцій у командному вікні FreeMat. При роботі з великою кількістю команд вам буде легше оперувати із програмами, ніж працювати в основному вікні FreeMat. Таким чином, ви можете виконувати всі команди із програми, набравши в FreeMat лише одну інструкцію – ім'я файлу-програми.

Щоб створити програму вам потрібен звичайний текстовий редактор. В операційній системі Windows є програма блокнот. Ви можете використовувати текстові процесори, такі як MS Word або OpenOffice Writer, але ви повинні зберегти набрані вами команди як звичайний текст у форматі (ASCII). Якщо ви збережете ваші команди як стандартний файл MS Word.doc, FreeMat видасть помилку. Таким чином, краще використовувати простий текстовий редактор або, що зручніше, улаштований редактор FreeMat.

4.1. Редактор FreeMat

Важливою якістю FreeMat є те, що він постачається із улаштованим редактором (рис. 4.1). Ми наполегливо рекомендуємо вам

використовувати улаштований редактор. Щоб відкрити редактор FreeMat, ви можете використовувати один із таких варіантів:

– просто введіть edit у командному вікні та натисніть <Enter>;

– використовуйте поєднання клавіш Ctrl+E;

– виберіть у меню програми Tools->Editor.

Даний редактор забезпечує нумерацію рядків (корисно, коли ви отримуєте повідомлення про помилку для програми або функції, відповідно вказується номер рядка, де знаходиться помилка), підсвічування коду та автоматичне форматування для різних функцій (**for** і **while**), а також автоматичне розширення файлу «.m».

Рисунок 4.1 – Редактор

4.2. Створення програми

Давайте розглянемо процес створення програми у FreeMat. Введення команд, функцій, змінних і чисел у вікні редактора FreeMat здійснюється аналогічним чином, як і в командному вікні. Вводити команди необхідно в тій же послідовності, в якій ви хотіли б, щоб вони були виконані. Різниця між вікном редактора та командним у тому, що у вікні редактора нічого не буде виконано безпосередньо після введення. Після введення команди збережіть сценарій як файл із розширенням **.m**. Для цього у редакторі FreeMat виберіть у меню File –> Save або Save As. Або натисніть кнопку «Save». З'явиться вікно, яке повинно бути знайомим будь-кому, хто раніше використовував Windows, Linux або MacOS. Це стандартна команда "Save file". Програма запитає у вас, де ви

хочете помістити файл і його ім'я. Рекомендується зберігати програми в директоріях, вказаних у змінній оточення FreeMat (можна подивитись за допомогою утиліти Path Tool). Ім'я файлу повинно містити допустимі символи для імен файлів Windows (Linux, MacOS). Нижче перелічено вимоги до імен файлу для програм у FreeMat:

– ім'я файлу програми повинно починатися з літери;

– ім'я файлу програми може містити букви, цифри та підкреслювання;

– ім'я файлу програми не може містити пробілів.

Примітка. Якщо ви використовуєте FreeMat редактор, він буде автоматично привласнювати розширення .m. Але ми рекомендуємо взяти це за звичку та робити це самостійно. Таким чином, якщо ви створюєте файл в іншому редакторі (Notepad, VI в Linux, або будь-який інший), ви завжди будете мати правильне розширення і у вас буде менше труднощів у майбутньому.

Нумерація рядків. На рис. 4.2 показана програма з нумерацією рядків у вікні редактора. Номер рядка автоматично генерується. При виникненні помилки в програмі або функції FreeMat вказує номер рядка, де виникла помилка.

Рисунок 4.2 – Вікно редактора з нумерацією рядків

Виділення кольором і відступи. На рис. 4.3 показано вікно редактора з циклом **while**. Зверніть увагу на підсвічування коду, яке використовується для циклу. Також зверніть увагу на відступ операторів всередині циклу.

Рисунок 4.3 – Вікно редактора, на якому показне кольорове кодування та відступ

Ім'я файлу. Рис. 4.4 показує ім'я файлу в верхній частині вікна редактора. Ім'я файлу відображається або (а) після того, як ви зберегли файл, або (б), якщо ви завантажили файл.

Рисунок 4.4 – Вікно редактора, який відображає ім'я файлу

Приклад створення програми. Нижче наведено приклад програми. Введіть такі команди у вікно редактора.

```
% Створення затухаючої синусоїдоподібної хвилі.
% Даний приклад був взятий із офіційної документації FreeMat (стр 466)
x=linspace(–3,3,1024);
y=(exp(–(x.^2))).*cos(8*pi*x);
plot(x,y);
```

На рис. 4.5 зображено редактор із набраною програмою.

Рисунок 4.5 – Редактор з тестовою програмою

Коли ви закінчите введення, збережіть файл сценарію. У вікні "SaveFile" введіть ім’я файлу. Ми використовували **myScript.m** як ім’я файлу. Тепер повернемося до командного вікна. Введіть ім’я без розширення:

```
myScript
```

FreeMat виконує кожну команду в програмі у тій послідовності, в якій ви її ввели. Ця проста програма створює змінну масиву від 1 до 128, розраховує 8 періодів коливань синусоїди, а потім видає графік синусоїди (рис. 4.6).

Рисунок 4.6 – Графік затухаючої синусоїди, створений на основі програми

Саме примітне тут те, що програми FreeMat створювати дуже легко, а також легко їх повторювати. Ви можете використовувати стрілки вверх і вниз у командному вікні для переключення між командами із розділу «Історія». Таким чином, ви можете запустити програму, внести будь-які зміни в програму, повторно зберегти програму, а потім повторити її запуск із командного вікна. Це дуже зручно для роботи із великими програмами.

4.3. Запуск програми

Коли ви створили програму та зберегли її як .m файл, ви можете здійснити її запуск, просто ввівши ім'я файлу без розширення .m. Потім натисніть <Enter>. Наприклад, якщо ви створили файл із назвою "myscript.m", у командному вікні потрібно ввести на виконання таку програму

```
-->myscript
```

4.4. Створення та використання функції

Функція – це коротка програма, яка виконує будь-які дії. Різниця між програмою та функцією полягає в тому, що змінні, використовувані у програмі, є глобальними, змінні в функції – локальні. Це означає, що ви можете використовувати ті ж імена змінних у функції, які ви використовували за її межами. Функції можуть виконувати ряд розрахунків, забезпечувати введення, візуалізацію або їх комбінацію. Зазвичай ви створюєте функцію для якоїсь дії. Функція – це програма, яка починається з ключового слова функції та набуває одне (або більше) вхідних значень. Функції FreeMat мають такий синтаксис:

function [returnValue1, returnValue2, ... , returnValueM] =
function_name (parameter1, parameter2, ... , parameterN)
statements

Функція починається із слова function. Якщо вона повертає значення, необхідно об'явити змінну, яка буде використовуватись для повернення значень (або значення). Це значення, яке повертається, привласнюється імені функції та може приймати будь-яку величину. Нижче наведено загальний синтаксис функції, яка приймає один вхідний параметр.

function rv=funcName(x)

(різноманітні команди)

rv=<яке-небудь значення, яке повертається>.

Зверніть увагу, що функція не завершується словом end. Якщо функція визивається, FreeMat компілює функцію, поки не досягає останнього рядка. Після виконання всіх команд функції, програма повертає потрібне значення і повертається туди, де функція була викликана. Коли ви створюєте функцію, ви повинні привласнити файлу функції ім'я, яке співпадає з іменем функції. Наприклад, якщо ви створюєте функцію, яка має назву MyFunc, вона повинна бути збережена в файлі з ім'ям myFunc.m в одній із ваших робочих директорій.

Для використання функції введіть ім'я функції та потрібні параметри. Нижче наведено декілька прикладів.

Приклад функції, яка обчислює факторіал числа. FreeMat не має улаштованої функції факторіала, яка, як правило, позначається знаком оклику (!). Однак не важко її створити. Ця функція обчислює факторіал значення вхідної змінної n.

```
function return_value=fact(n)
n=floor(n);
return_value=gamma(n+1);
```

На рис. 4.7 наведено код програми у вікні редактора FreeMat.

Рисунок 4.7 – Вікно редактора з функцією

Оскільки функція повинна бути збережена з тим же ім'ям, ім'я файлу для даної функції – fact.m. Нижче показано результат розрахунку 5!.

```
fact(5)
```

```
ans =
120.0000
```

Ми можемо також привласнити значення функції змінній.

```
y=fact(5)
y =
120.0000
```

Є можливість створити функцію, яка потребує більш ніж один вхідний параметр. Щоб створити таку функцію просто, вкажіть змінні для всіх необхідних параметрів. Тобто, функція, яка потребує три вхідних параметри, буде виглядати таким чином:

exampleFunction(x,y,z)

де *x*, *y*, *z* – вхідні параметри.

Приклад функції з декількома параметрами. У теорії ймовірностей комбінаторним називається число із заміною *r* об'єктів у наборі з *n* об'єктів. Рівняння для розрахунку кількості комбінацій:

$$C_{n,r} = \frac{n!}{(n-r)!r!}.$$

Спосіб Бернуллі дозволяє розрахувати ймовірність виконання події *k* в *n* випробуваннях, з імовірністю *p*. Саме рівняння використовує комбінаторні функції і виглядає таким чином:

$$P_n(k) = C_{n,k} p^k (1-p)^{n-k}.$$

Всі ці рівняння з кількома вхідними параметрами. Ми можемо створити функції для кожного з них. Почнемо з комбінаторної функції.

```
function rv=comb(n,r)
rv=gamma(n+1)./(gamma(n-r+1).*gamma(r+1));
```

Зберігши цей код під ім’ям comb.m, ми можемо спробувати цю функцію. Подивимося скільки буде комбінацій із 3 об'єктів у наборі із 5.

```
comb(5,3)
ans =
10.0000
```

Це говорить про те, що 3 об'єкти в набір із 5 можуть бути об'єднані в 10 комбінацій. Ось приклад:

0XXX0
0XX0X

0X0XX
00XXX
X0XX0
X0X0X
X00XX
XX0X0
XX00X
XXX00

Далі ми розглянемо функції для розрахунку ймовірності способом Бернуллі. Ця функція вимагає три вхідні параметри: кількість випробувань *n*, кількість успіхів *k*, та ймовірність успіху *p*.

```
function return_value = bern(n,k,p)
x1 = comb(n,k);
x2 = p.^k;
x3 = (1-p).^(n-k);
return_value = x1.*x2.*x3;
```

4.5. Удосконалення функції

Є декілька прийомів, які допоможуть покращити продуктивність ваших власних функцій, це перевірка вхідних параметрів і використання поелементного множення та поділу (коли це можливо).

4.5.1. Перевірка вхідних параметрів

У FreeMat існує команда, яка має назву **nargin**, що розшифровується як "кількість аргументів, які вводяться". Ця команда при виклику всередині функції підраховує кількість вхідних аргументів, переданих функції. Ви можете використовувати її, щоб визначити: чи всі необхідні параметри передані вашій функції або встановити деякі значення за замовчуванням для параметрів.

Приклад підрахунку кількості переданих у функцію аргументів. У цьому першому прикладі ми подивимося на кількість вхідних параметрів для визначеної функції. Якщо в функцію не передано жодного аргументу, ми будемо виводити повідомлення про помилку. Використаємо функцію Бернуллі як приклад, але з деяким додаванням коду для перевірки вхідних аргументів.

```
function return_value = bern(n,k,p)
if(nargin<3);
printf('This function requires three inputs, which are (in order):\n');
```

```
printf('the number of trials, the number of successes and the
probability\n');
printf('of success of each trial.\n');
return
end
x1 = comb(n,k);
x2 = p.^k;
x3 = (1-p).^(n-k);
return_value = x1.*x2.*x3;
```

Запустивши цю функцію тільки з двома вхідними параметрами, ми отримуємо:

```
bern(10,0)
ERROR: This function requires three inputs, which are (in order):
the number of trials, the number of successes and the probability
of success of each trial.
```

Ми також можемо зробити так, що у випадку, якщо ні один аргумент не задано, для них будуть встановлені значення за замовчуванням. Наприклад, ми можемо встановити значення за замовчуванням для третього аргументу, яке дорівнює 0.5.

```
function return_value = bern(n,k,p)
if(nargin<2);
printf('ERROR: This function requires at least two inputs, which
are (in
order):\n');
printf('the number of trials and the number of successes.\n');
return
end
if(nargin<3);
p=0.5;
end
x1 = comb(n,k);
x2 = p.^k;
x3 = (1-p).^(n-k);
return_value = x1.*x2.*x3;
```

4.5.2. Використання поелементного обчислення функцій

Наша первинна функція має вигляд:

```
function rv=comb(n,r)
rv=gamma(n+1)./(gamma(n-r+1).*gamma(r+1));
```

Зазначимо, що для множення ми використовуємо ".*", а для поділу використовуємо "./". Поелементне множення і поділ обговорювались раніше. Поелементні математичні операції для функцій дозволяють використовувати їх як з масивами, так і скалярами.

Приклад. Особливості поелементної математики функцій. Давайте повернемось до нашої комбінаторної функції **comb** і встановимо її для нормального, а не поелементного поділу та множення:

```
function rv=comb(n,r)
rv=gamma(n+1)/(gamma(n-r+1)*gamma(r+1));
```

Якщо ми спробуємо передати в неї масив як вхідні дані, то ось що виходить:

```
comb(10,0:3)
In /home/gary/Freemat/myFunctions/comb.m(comb) at line 21
In docli(builtin) at line 1
In base(base)
In base()
In global()
```

Error: Requested matrix multiplication requires arguments to be conformant.

Використовуючи поелементні математичні операції для множення та поділу функцій, ми можемо не турбуватися про те чи є аргумент скаляром, чи масивом.

Ми рекомендуємо вам використовувати поелементні операції в програмному коді функції. Це зробить ваші функції більш корисними.

4.6. Коментарі до ваших програм і функцій

Програма або функція, яку ви пишете сьогодні, може здатися вам інтуїтивно зрозумілою для будь-якого починаючого користувача. Але завтра або через місяць ви подивитесь на цю функцію та запитаєте себе: «Про що я думав, коли писав це?». Ось тут вам знадобляться коментарі. Коментар – це не більше ніж текст, введений в код вашої функції або програми, який передбачає якесь пояснення або контекст. Він просто

роз'яснює що робить цей код. Всі добре знають, що в середньому ваш код, будь-то функція чи програма, буде мати більше коментарів ніж рядків із командами.

Щоб помістити коментар у код, використовуйте символ процента (%). Якщо ви використовуєте редактор FreeMat, ви помітите, що редактор автоматично виділяє коментарі світло-коричневим кольором, щоб зробити їх такими, що легко виділяються із самого коду (це ще одна причина, за якою ми рекомендуємо користуватись вбудованим редактором FreeMat). Коментар може бути на окремому рядку або написаним після команди у тому ж рядку. Все, що написано після знаку (%), вважається коментарем і не буде мати впливу на виконання програм або функцій.

```
% Це приклад коментаря на окремому рядку;
n=5; % Це приклад коментаря після команди.
```

Ви побачите в цьому посібнику різні програми та функції із коментарями в них. Це не тільки дозволить вам, читачеві, краще їх зрозуміти, але й допоможе в майбутньому при написанні нових.

4.7. Анонімні функції

У змінних FreeMat можна зберігати не тільки звичайні значення, такі як числа та рядки, але й функції. Ця можливість дає FreeMat велику гнучкість. Такий вид функцій називається анонімними. Визначення анонімної функції складається з імені змінної, оператора «=@», списку аргументів функції (у круглих дужках) та самого виразу функції (рис. 4.8). По суті, анонімна функція – функція, яка зберігається в змінній.

Рисунок 4.8 – Описання анонімної функції

Нижче наведено невеликий приклад, який демонструє використання анонімної функції. Так само, як і для стандартних функцій, тут слід застосувати поелементне множення та поділ для того, щоб ваші анонімні функції працювали з масивом так само, як і зі скаляром:

```
f=@(x) (exp(–x.^2));
x=linspace(–1,3,10);
f(x)
ans =
0.3679  0.7344  0.9877  0.8948  0.5461  0.2245  0.0622
0.0116  0.0015  0.0001
```

Давайте проаналізуємо, те що ми зробили – ім'я нашої функції (*f*) можна використовувати як і будь-яку іншу змінну. Коли ми набираємо *f* = @ (*x*), символ @ означає, що ця функція (не вектор або клітинка іншої змінної); (*x*) – позначає, що ця функція залежить від змінної x. Якщо б вона залежала від *x*, *y* і *z*, ми повинні були написати *f* =@(*x*, *y*, *z*). Решта виразу просто набір команд, який буде виконуватись в даному випадку:

$$f(x) = e^{-x^2}.$$

Коли ми викликаємо f(3), буде обчислено значення *exp*(–3^2).

4.8. Програмне введення і виведення

Функція **input** відповідає за введення даних у програму. Функція **printf** використовує спеціальні заповнювачі для форматування даних і подальшому їх виведенні.

4.8.1. Функція printf

Функція **printf** використовується для виведення рядків. Нижче наведено формат функції **printf:**

printf(string, n1,n2,…)

де string – рядок, який буде роздруковано; *n*1,*n*2,… – числа або рядки, які будуть вставлені замість заповнювачів string.

Приклад використання функції **printf**.

```
--> printf('This is a string with the newline character at the end.\n');
This is a string with the newline character at the end.
--> printf('This is a test.\n');
This is a test.
```

```
--> printf('How now brown cow?\n');
How now brown cow?
```

У наступному прикладі ми роздрукуємо декілька рядків тексту, використовуючи одну функцію **printf** та спеціальний символ «\n»:

```
--> printf('This is line 1.\nThis is line 2.\n');
This is line 1.
This is line 2.
```

Рядок може складатись із будь-якого набору символів в одинарних лапках. Нижче наведено декілька прикладів задавання рядка:

```
'hello'
'Hello'
'my name is Joe.'
'This is yet another example of a string.'
```

Якщо ви не вкажете символ «\n» в кінці рядка, FreeMat перезапише ваш рядок рядком командного введення або такою функцією **printf**.

Приклад використання символу «\n» в кінці рядка **printf**:

```
printf('intvalue is %d, floatvalue is %f',3,1.53);
printf('intvalue is %d, floatvalue is %f\n',3,1.53);
intvalue is 3, floatvalue is 1.530000
```

4.8.2. Виведення чисел

Ви не можете вивести число просто підставивши його в функцію **printf**. Якщо ви намагатиметесь зробити це, FreeMat видасть вам повідомлення про помилку.

```
printf(6);
Error: printf format argument must be a string
```

Для того щоб роздрукувати число, вам необхідно як перший аргумент функції **printf** вказати рядок, який містить спеціальний заповнювач. Число може бути надруковано у вигляді цілого або дробового. Нижче перелічені спеціальні заповнювачі для виведення на екран чисел:

%f: – число з плаваючою крапкою, за замовчуванням виводиться 6 знаків після крапки;

%+f: – число з плаваючою крапкою, перед яким виводиться знак «+» для всіх додатних чисел і знак «–» для всіх від’ємних.

%e: – число з плаваючою крапкою в експоненціальному поданні;

%n.mf: – число з плаваючою крапкою, для відображення якого використовується n символів (точка також є символом) і m знаків після крапки;

%0n.mf: – число з плаваючою крапкою, зліва доповнено нулями;

%d: – ціле число.

Приклад виведення чисел у різних форматах.

```
printf('The number is %f.\n',pi/10);
The number is 0.314159.
printf('The number is %f\n',pi);
The number is 3.141593
```

4.8.3. Виведення спеціальних символів

Ви можете друкувати символи, використовуючи їх ASCII коди (ціле число між 0 і 255). Наприклад, заголовна буква «A» має ASCII код 65, цифра «7» – код 55, а знак питання «?» – код 63. Коди ASCII розділені на дві групи: стандартні та спеціальні. Діапазон стандартних кодів 0–127, символи для даного діапазону стандартизовані. Символи із кодами із діапазону 128–255 можуть відрізнятись в деяких реалізаціях.

Для виведення на друк символів FreeMat подає спеціальний заповнювач – **%c**, який дозволяє вам додавати спеціальні символи в ваші рядки. Деякі спеціальні символи можуть бути виведені за допомогою клавіатури ($, #, @, &, !).

```
printf('%c',code)
```

4.9. Функції для зчитування даних

FreeMat подає можливість введення даних із клавіатури. Для цього в нього є спеціальна функція **input**. Нижче наведено її синтаксис для введення чисел:

r=input('Prompt text')

Синтаксис для введення рядка показаний нижче:

r=input('Prompt text','s')

Приклад введення числа та обчислення його квадрата.

```
x=input('Enter a number');
x^2
```

4.10. Зчитування даних із текстового файлу в кодуванні ASCII

Функція **dlmread** забезпечує можливість зчитування даних із файлу в кодуванні ASCII. Функція має такий синтаксис:

data = dlmread(filename,<separator>,<range>)

де separator – це символ дільник даних (,;:<tab>); range – діапазон стовбців для зчитування.

Приклад зчитування даних із 2-х колонок даного текстового файлу (стовпці розділені символом табуляції).

```
clear all;
close('all');
data=real(dlmread('supermanAccelY1.txt',char(9)));
t=data(:1,);
accel=data(:,2);
```

4.11. Функції зчитування/запису в файл

Роботу функції зчитування/запису в файл продемонструємо на прикладі.

```
% Read in ASCII text from standard text file.
clear all;
fp=fopen('Dol.txt','r');
charVals=fread(fp,[1,inf],'int8');
doiText=string(charVals);
printf([charVals '\n']);
```

Зберігши даний файл під ім’ям **readDol.m** і вибравши в командному вікні FreeMat

```
readDol
```

ми виведемо на екран вміст файлу Dol.txt.

5. УПРАВЛІННЯ ПОТОКОМ КОМАНД

У даному розділі описуються різні оператори управління потоком виконання програм у FreeMat. Тут будуть показані різні способи, за допомогою яких потік виконання команд у програмі або функції може бути зміненим відповідно із заданими вами умовами. Розглянуто оператори циклів **for** і **while**, а також умовний оператор **if-elseif-else**.

5.1. Цикл for

Цикл **for** використовується в тих випадках, коли вам необхідно виконати блок команд деяку визначену кількість раз. Оператори циклу мають такий синтаксис:

for (variable = expression);
statements
end

Частина в круглих дужках буде лічильником циклу. Взагалі FreeMat не потребує тут використання круглих дужок, але ми наполегливо рекомендуємо вам їх використовувати, оскільки код стає легким для читання.

Приклад використання циклу **for** для розрахунку висоти снаряда. Дана програма (збережіть її під ім'ям **projectile.m**) розраховує та відображає висоту снаряда, запущеного вертикально вверх. Для генерації часу з інтервалом 1 с використано цикл **for**. Для кожного моменту часу проводиться розрахунок висоти:

```
velocity = 100; % вертикальна початкова швидкість
zeroTime = velocity/9.8; % час необхідний для досягнення максимальної висоти
maxHeight = 4.9*(zeroTime)^2; % максимальна висота снаряда
printf('The projectile initial velocity is %.2f meters/second.\n',velocity);
printf('The maximum height is %.3f meters.\n',maxHeight);
printf('The time to maximum height is %.2f seconds\n\n',zeroTime);
totalTime=ceil(2*zeroTime);  % Час польоту снаряда до його зупинки
```

```
for (k = 1:totalTime); % Даний цикл for використовує крок, який дорівнює 1
distance = velocity*k-(4.9*k^2);
printf('Height at %d seconds is %.4f meters.\n',k,distance);
end
```

Після запуску даної програми на виконання отримуємо такий результат:

```
projectile
The projectile initial velocity is 100.00 meters/second.
The maximum height is 510.204 meters.
The time to maximum height is 10.20 seconds
Height at 1 seconds is 95.1000 meters.
Height at 2 seconds is 180.4000 meters.
Height at 3 seconds is 255.9000 meters.
Height at 4 seconds is 321.6000 meters.
Height at 5 seconds is 377.5000 meters.
Height at 6 seconds is 423.6000 meters.
Height at 7 seconds is 459.9000 meters.
Height at 8 seconds is 486.4000 meters.
Height at 9 seconds is 503.1000 meters.
Height at 10 seconds is 510.0000 meters.
Height at 11 seconds is 507.1000 meters.
Height at 12 seconds is 494.4000 meters.
Height at 13 seconds is 471.9000 meters.
Height at 14 seconds is 439.6000 meters.
Height at 15 seconds is 397.5000 meters.
Height at 16 seconds is 345.6000 meters.
Height at 17 seconds is 283.9000 meters.
Height at 18 seconds is 212.4000 meters.
Height at 19 seconds is 131.1000 meters.
Height at 20 seconds is 40.0000 meters.
Height at 21 seconds is -60.9000 meters.
```

Приклад розрахунку значень функцій синуса за допомогою циклу **for**. Якщо ви новачок в інженерній справі, фізиці або математиці, вам може буде цікаво, чому величини кутів розраховуються в радіанах (в той час як більшість людей звикли рахувати величини кутів у градусах). Це

частково зумовлено тим, як програмно розраховується значення синуса (sin), косинуса (cos) і тангенса (tan). Розрахунок проводиться за такими формулами:

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} \ldots,$$

$$\cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} \ldots,$$

$$\tan(x) = \frac{\sin(x)}{\cos(x)}.$$

де x – кут у радіанах, тобто є безрозмірною величиною.

Даний приклад показує як використовувати цикл **for** для отримання значення функцій синуса та косинуса. Навіть, якщо ряд розкладання теоретично можна продовжити до нескінченності, ми не будемо висувати вимогу досягнення надзвичайної точності. У прикладі, показаному нижче, ми будемо використовувати тільки 8 членів ряду.

```
% Обчислення  sin(x) за допомогою ряду Маклорена та циклу for
clear all;
x = input('Angle = '); % введіть кут
c = 1;  % Змінна, яка використовується для чергування додавання та віднімання.
totalSum = 0; % Змінна, яка використовується для зберігання суми ряду.
for(ii = 1:2:15);
totalSum = totalSum+(x^ii)/fact(ii)*c; % Обчислюємий доданок ряду
c=c*(–1); % Чергуємо додавання та віднімання
end
totalSum % Відображаємо скінчену суму ряду
```

Збережемо дану програму під ім’ям **mySin.m**, запустимо її і виведемо результат для x = 3:

```
mySin
Angle = 3
ans =
0.1411
```

Давайте тепер порівняємо результат із значенням, отриманим за допомогою функції **sin** програми FreeMat:

```
sin(3)
ans =
   0.1411
```

У підсумку отримаємо однакові відповіді. Для точнішого порівняння використаємо команду **format**

```
format long
mySin
Angle = 3
ans =
  0.14111965434119
sin(3)
ans =
  0.14112000805987
ans-totalSum
ans =
 3.53718673018477e-07
```

Таким чином, обчислення функції **sin** за допомогою ряду Тейлора проводиться з точністю до 5-го знака після крапки. Шляхом додавання членів у ряд Маклорена ми можемо добитися більшої точності. Аналогічна методика застосовується при розрахунку функції косинуса за допомогою розкладання в ряд Маклорена.

```
% розрахунок функції cos(x) з використанням ряду Маклорена
clear all;
x = input('Angle = '); % Вводимо кут в радіанах
c = 1; % Змінна, яка використовується для чергування додавання та віднімання
totalSum = 0; % Змінна, яка використовується для зберігання суми ряду.
for(ii = 0:2:14);
totalSum = totalSum+(x^ii)/fact(ii)*c; % Обчислення доданку ряду
c = c*(–1); % Чергуємо додавання та віднімання
end
totalSum % Відображаємо підсумкову суму ряду
```

Збережемо файл із ім'ям **myCos.m**, запустимо його, введемо значення для змінної x = 3, отримаємо такий результат:

```
myCos
Angle = 3
ans =
 – 0.98999449490242
```

Тепер порівняємо отриманий результат із значенням, отриманим за допомогою функції **cos** програми FreeMat:

```
cos(3)
ans =
 –0.98999249660045
```

Дані значення також є близькими. Треба відзначити, що програма FreeMat використовує більш складні алгоритми розрахунку для тригонометричних функцій, ніж ті, які використовували ми для написання програм за допомогою циклів. Є спеціальні прийоми, які допомагають обчислити значення функцій для великих кутів (наприклад 1000 радіан, при таких кутах використання методики розкладання в ряд не дасть точного результату).

Для розрахунку значення функції тангенса використовуються значення функцій синуса та косинуса. Результат отримуємо за допомогою поділу.

5.2. Оператори порівняння

Перед тим як розглянути цикл **while** і **if-elseif-else**, давайте розглянемо оператори порівняння. Дані оператори використовуються для перевірки двох величин на рівність (або нерівність). У програмі FreeMat є 6 операторів порівняння:

– оператор дорівнює (==);

– оператор не дорівнює (~=);

– оператор менше (<);

– оператор більше (>);

– оператор менше або дорівнює (<=);

– оператор більше або дорівнює (>=).

У той час як людина думає про результат операцій порівняння істина чи брехня, комп'ютер сприймає дані значення як 1 і 0 відповідно. Вище перелічені оператори порівняння повертають значення 0, якщо

результат операції порівняння брехня та 1, якщо результат операції порівняння істина.

```
5>1
ans =
 1
5<1
ans =
 0
6*5==30
ans =
 1
x=1:10
x =
  1  2  3  4  5  6  7  8  9 10
x<=3
ans =
 1 1 1 0 0 0 0 0 0 0
```

Ми також можемо використовувати операції І та АБО для обчислення додатних умов. Операція І здійснюється шляхом перемноження двох операцій порівняння. Операція АБО – шляхом їх складання.

І: <порівняння #1>*< порівняння #2>;

АБО: < порівняння #1>+< порівняння #2>.

Нижче наведено декілька прикладів:

```
(125^(1/3)<7)*(24/3==8)
ans =
 1
x=1:5;
y=1:5;
(x<2).*(y>4)
ans =
 0 0 0 0 0
(x<=2).*(y<4)
ans =
 1 1 0 0 0
```

5.3. Цикл while

Цикл **while** здійснює виконання блока команд доти, поки значення умови циклу **while** повертає додатне число. Не дивлячись на той факт, що результатом операції порівняння, які повертають істину, є число 1, для циклу **while** (аналогічно **if-elseif-else**) достатньо лише, щоб число було додатним. Цикл **for** використовується для ітерацій масивів. Коли досягнуто останнього елемента масиву, цикл завершує свою роботу. Цикл **while**, з іншого боку, буде виконуватись доти, поки значення його умови не стане рівним 0. Стандартне використання циклу **while** потребує наявності в ньому блока порівняння:

```
while (comparison operation);
  statements
end
```

Приклад пошуку найбільшого загального дільника. У даному прикладі використовується цикл **while**, визначається функція для розрахунку найбільшого загального дільника двох цілих чисел:

```
function gcd_value=gcd(n,m)
while (m ~= 0);
temp = mod(n,m);
n = m;
m = temp;
end
gcd_value=n;
```

Найбільший загальний дільник знаходиться за допомогою розрахунку залишку від поділу n на m за допомогою функції **mod**. Даний залишок використовується потім для привласнення його одному із вхідних параметрів. Цикл **while** продовжує свою роботу доти, поки залишок від поділу не стане рівним 0. Даний алгоритм має назву евклідової версії для знаходження загального дільника двох чисел. Збережемо дану функцію в файл **gcd.m**. Таким чином, якщо вам потрібно буде розрахувати найбільший дільник чисел 100 та 36, ви напишете таким чином:

```
gcd(36,100)
ans =
4
```

Найбільший загальний дільник чисел 100 і 36 – число 4.

Приклад обчислення кубічного кореня числа. У даному прикладі розглянемо функцію, яка використовує цикл **while** для обчислення кубічного кореня числа. Дані функції використовують ітераційний цикл для отримання наближеної відповіді. Задаючи початкове наближення, яке дорівнює s(1), ця функція обчислює такі значення за допомогою співвідношення:

$$s(n+1) = \frac{1}{2}(s(n) + \frac{x}{s(n)^2})$$

Нижче наведено реалізацію даної функції.

```
% cubeRoot.m
function returnValue=cubeRoot(x)
n=1;
s(n)=1; % Початкове наближення
diffS=1; % Різниця між двома наближеннями
epsilon=1e-8; % Потрібна точність відповіді
while(diffS>abs(epsilon*s(n)));
s(n+1)=(1/2)*(s(n)+x/(s(n)^2));
n=n+1;
diffS=abs(s(n)–s(n–1));
end
returnValue=s;
```

Дана функція повертає послідовність наближень, починаючи з початкового (s(1)). Цикл **while** перевіряє, чи досягли ми потрібної точності.

Знайдемо за допомогою даної функції кубічний корінь числа 2:

```
format long
y=cubeRoot(2);
y
ans =
1.00000000000000
1.50000000000000
1.19444444444444
1.29814163812271
1.24248215661613
1.26900936034694
```

```
1.25547429372185
1.26216808077850
1.25880353145720
1.26048129769035
1.25964129946292
1.26006101831131
1.25985108900747
1.25995603616619
1.25990355821644
1.25992979609835
1.25991667688420
1.25992323642298
1.25991995663651
1.25992159652548
1.25992077657993
1.25992118655243
1.25992098156611
1.25992108405926
1.25992103281268
1.25992105843597
1.25992104562433
1.25992105203015
```

Для знаходження даного значення було потрібно багато ітерацій циклу. Ми можемо настроїти нашу функцію таким чином, щоб вона повертала нам тільки останнє значення послідовності. Для цього необхідно змінити функцію таким чином:

```
returnValue=s(n);
```

Запустивши цю змінену функцію для знаходження кубічного кореня 5, отримуємо такий вираз:

```
cubeRoot(5)
ans =
1.70997595038430
```

Нижче подана невелика програма, яка демонструє той факт, що цикл **while** приймає як результат умови додатне число.

```
% testWhile.m n=5;
while(n);
```

```
printf('The number is %d.\n',n);
n=n–1;
end
```

Запустивши дану програми, ми отримаємо такий вираз:

```
--> testWhile
The number is 5.
The number is 4.
The number is 3.
The number is 2.
The number is 1.
```

5.4. Цикл if-elseif-else

Даний оператор схожий на аналогічний оператор порівняння в мовах програмування загального призначення (C++, Java). Оператор **if** використовує операцію порівняння. Якщо операція порівняння повертає додатне значення (істину), будуть виконані команди, які випливають за оператором **if**, поки не буде зустрінутий оператор **end**. Нижче наведено повний формат умовного оператора:

```
if (comparison operation);
  statement
elseif (comparison operation)
  statement
else (comparison operation)
  statement
end
```

Найпростіша версія оператора if має такий вигляд:

```
if (comparison operation)
  statement
end
```

Приклад використання оператора **if**. Тут наведено простий приклад програми, яка порівнює два числа та виводить більше з них.

```
x=5;
y=3;
greaterNumber=x;
if (x<y);
  greaterNumber=y;
```

```
end
greaterNumber
```

У даному прикладі задаються дві змінні – *x* і *y*. Перед виконанням операції порівняння ми робимо припущення, що *x* є велике число. Оператор **if** перевіряє, чи є у більшим за *x*. Якщо це так, то виконується команда, яка првласнює змінній greaterNumber значення змінної y. Збережемо програму під ім'ям **greaterNumber.m**. Для запуску запишемо її ім'я у командному рядку:

```
greaterNumber
ans =
5
```

6. ГРАФІКИ ТА РИСУНКИ

FreeMat має дві команди для створення 2D (двовимірних) графіків. Це команда **plot** (FD, стр. 478) та команда **line** (FD, стр. 474). Є різниця між цими двома командами, яку ми стисло опишемо.

Команда **plot** тільки для 2D-графіки. (Існує окрема команда **plot3** (FD, стр. 482) для тривимірних графіків). **Plot** створює вікно (якщо воно ще не відкрито) або переписує поточне вікно рисунку (якщо воно вже відкрито). Ця команда дозволяє використовувати один, два або три вектори вихідних даних, пов'язаних з *x*, *y* і *z* відповідно. Якщо використовувати тільки один вектор, то команда буде будувати тільки двовимірний графік, використовуючи вектор значень для осі *y* і лічильник для осі *x*. Команда **plot** може рисувати декілька кривих різним кольором залежно від властивостей настройки **colororder**;

Командою **line** можна створювати як 2D, так і 3D-графіки. Насправді, завжди необхідно задати, принаймні, вектори значень для осей *x* і *y*. Далі, якщо ви хочете встановити різні властивості (колір лінії, ширину лінії, стиль лінії), ви повинні подати набір даних для всіх трьох осей – *x*, *y*, *z*. Команда **line** створює вікно рисунку (якщо воно ще не відкрито) або додає ще одну лінію на існуючому рисунку. Нею можна побудувати тільки один графік, але вона забезпечує більшу гнучкість у настройки кольору ліній, ніж команда **plot**.

На наш погляд, велика різниця між цими двома командами в тому, що команда **plot** перезаписує існуючі графіки, в той час як команда **line** просто додає ще один графік до вже існуючого. Команда **line** є дещо складнішою у використанні через те, що необхідно задавати всі три вектори даних для контролювання налаштувань, таких як ширина та колір. Ви можете використовувати команду **line** тільки с *x*-масивом і *y*-масивом, але це дасть нам просто тонку чорну лінію. При першому виклику команд **plot** або **line** створюється вікно фігури, яке містить 600x400 пікселів. Якщо обчислити площу вікна, то зона дорівнює 600x335 (для Linux версії Freemat) або 600x344 (для версії Windows). Фактична площа накреслення графіків приблизно 480x268 або приблизно 80 % від повного розміру вікна.

6.1. Створення графіка або рисунка

Ви можете створити декілька різних типів графіків у FreeMat. Залежно від конкретної команди, яку ви використовуєте, можна створити графік однієї, двох або трьох змінних, як *y*, *xy* (рис. 6.1), *xyz* відповідно. Використання однієї змінної дає вам графік цієї змінної вздовж прямої лінії. Дві змінні забезпечують графік *xy*, на якому ви можете створити всі види двовимірних форм. Графік *xyz* дозволяє створювати тривимірні форми. У цьому посібнику ми розглянемо графіки однієї та двох змінних, а 3D графіки розглянемо у майбутньому.

Рисунок 6.1 – Двовимірний рисунок

Незалежно від типу створюваного графіка основний принцип той же. FreeMat використовує кожну точку вектора осі абсцис і осі ординат, і розміщує її на графіку. Якщо в **linestyle** вказана форма лінії (суцільна лінія, пунктир або точка пунктир), команда буде створювати цей тип лінії між кожною із точок. Якщо **linestyle** визначається як маркер (точка, квадрат або ромб і т.ін.), буде просто вказаний маркер у кожній точці.

Наприклад, давайте розглянемо два векторам з п'ятьма значеннями. Будемо використовувати такий масив:

Якщо ми подивимося на кожну точку пари *xy*, то побачимо такий рисунок (рис. 6.2).

Рисунок 6.2 – Формування точок графіка

Зверніть увагу, що у нас є однакова кількість елементів в обох масивах. Якщо ми спробуємо побудувати рисунок за допомогою масивів різного розміру, FreeMat просто поверне пусте вікно. Ви не отримаєте помилки відносно того, чому поле пусте.

Графік, побудований за точками, показано на рис. 6.3.

Рисунок 6.3 – П'ятиточковий графік, який показує як точки побудовані та зв'язані

FreeMat наносить точки *xy*, потім рисує лінії сполучення між точками.

6.1.1. Створення графіка функції однієї змінної

Для базового графіка використовується команда **plot**, а саме:

plot(y)

де *y* – ім'я змінної, яка містить масив для побудови.

Ви також можете використовувати команду **line**, хоча її використання потребує набору двох векторів, для осі абсцис і осі ординат, наприклад

line(x,y)

де *x* – змінна, яка містить значення по осі абсцис; *y* – змінна яка, містить значення осі ординат.

При створенні рисунка він поміщається в окреме вікно, яке має номер. За замовчуванням перший рисунок буде **Figure 1**. Будь-які наступні рисунки будуть зроблені у цьому ж вікні, якщо ви явно не об'явите FreeMat помістити їх у різні вікна. Використовуючи команду **hold** (FD, стр. 466), можна помістити декілька графіків в одному вікні. У зворотному випадку вихідний графік буде стертий, а новий запишеться на його місце. Детальніше про це розповімо пізніше.

Давайте почнемо з графіка однієї змінної.

Приклад створення графіка однієї змінної. У першому прикладі будемо використовувати команду **plot.**

```
t = linspace(0,8*pi,256);
y = sin(t);
plot(y)
```

Отриманий графік показано на рис. 6.4. Зверніть увагу, що вісь *ox* (так звана "незалежна вісь") є лічильником індексу, яка використовується для змінної *y*. FreeMat при побудові однієї змінної створює масив осі абсцис із використанням лічильника індексу, відповідно, значення по осі *x* пробігають від 1 до 256.

Рисунок 6.4 – Графік однієї змінної, побудований за допомогою команди plot

Для того щоб створити графік *xy*, вам потрібно мати два вектори, які мають однакову довжину, а після того використовувати такий формат для команди **plot**:

plot(x,y)

де *x* – змінна, яка вміщує дані у напрямі вісі *x* (горизонтальна); *y* – змінна, яка вміщує дані у напрямі вісі *y* (вертикальна).

Команда **line** аналогічно:

line(x,y)

де *x* – змінна, яка вміщує дані у напрямі вісі *x* (горизонтальна); *y* – змінна, яка вміщує дані у напрямі вісі *y* (вертикальна).

Приклади створення *xy* рисунка. Перший приклад використовує команду **plot**. Створюється відрізок лінії, з використанням координат початку та кінця:

```
clear all;
close('all')
line([–2,2.5],[–0.3,1])
```

Результат показаний на рис. 6.5. Можливість будувати лінії між точками дозволяє створювати вашу власну вісь в будь-якому місці, де ви хочете.

Рисунок 6.5 – Відрізок лінії від точки (–2; –0,3) до точки (2,5; 1)

Другий приклад показує косинусоїду, нанесену на горизонтальній вісі навпроти синусоїди на вертикальній вісі. У результаті отримуємо коло. Результат показано на рис. 6.6.

```
theta = linspace(0,2*pi,500);
x = cos(theta);
y = sin(theta);
plot(x,y)
```

Рисунок 6.6 – Графік кола

Ось ще один приклад, який використовує команду **line** для створення п'ятикутної зірки (рис. 6.7).

Рисунок 6.7 – Зірка, яка побудована за точками

```
theta = 0:144:720; % Зміна кута в градусах між точками
x = sind(theta); % Використовуємо функцію sin, яка приймає значення в градусах
y = cosd(theta); % Використовуємо функцію cos, яка приймає значення в градусах
clf; % Очистка попереднього рисунку, якщо він є
line(x,y); % Побудова точок для створення зірки
```

Існують особливості при побудові двовимірних графіків, які можуть зробити їх кращими. Це використання команди **axis equal**. Дана команда робить довжини обох осей *x* і *y* одного масштабу. Простіше кажучи, це означає що рис. 6.6 буде більше схожий на коло, а не на еліпс.

Зірка на рис. 6.8 стане більш пропорційною. Ми використовуємо команду **axis equal** для демонстрації.

```
theta = 0:144:720;
x = sind(theta);
y = cosd(theta);
clf; %
line(x,y); %
axis equal;
```

Далі результат.

Рисунок 6.8 – П'ятикутна зірка, створена за допомогою команди **line** в осях рівного масштабу. Зверніть увагу, що вона виглядає більш пропорційно, ніж оригінальна зірка

6.1.2. Створення декількох графіків на одному рисунку

Є три способи створення декількох графіків на тому ж рисунку. До них належать:

1. використання команди **plot** з декількома змінними;
2. кількаразове використання **plot** разом із командою **hold**;
3. використання команди **line** декілька раз.

Приклад створення графіка з декількома змінними. У цьому прикладі ми будемо використовувати, один раз команду **plot** для декількох кривих. Для того щоб це зробити ми повинні використовувати пари векторів значень *xy*, тому використовуємо *"x, y1, x, y2, x, y3"*, а не просто *"y1, y2, y3"*.

```
x = linspace(0,6*pi,600);
y1 = sin(x);
y2 = sin(x)–1;
y3 = sin(x)–2;
```

```
plot(x,y1,x,y2,x,y3)
```

Результат показано на рис. 6.9. Зверніть увагу, що ми не визначаємо кольори, за замовчуванням FreeMat присвоює кольори із списку, який складається з сімох кольорів для команди **plot**. При використанні команди **line** всі лінії завжди будуть чорними, якщо іншого кольору не буде вказано. Ми це розглянемо далі, коли будемо обговорювати кольори графіків.

Рисунок 6.9 – Графіки трьох функцій, нарисовані за допомогою команди **plot**. Зверніть увагу на різноманітні кольори, використані для кожного з них.

Для другого прикладу ми будемо використовувати декілька раз команду **plot**. При цьому ми повинні використовувати команду **hold**, у протилежному випадку кожне використання команди **plot** буде просто стирати та перезаписувати графіки, зроблені до цього.

```
x = linspace(0,6*pi,600);
y1 = sin(x);
y2 = sin(x)–1;
y3 = sin(x)–2;
plot(x,y1);
hold on;
plot(x,y2);
plot(x,y3);
hold off;
```

Результат показаний на рис. 6.10. Зверніть увагу, що він ідентичний рис. 6.9. Перевагою цього методу є те, що кожна крива може мати індивідуальні товщину ліній, кольорів і тип, що не можливо, якщо одноразово використовувати команду **plot**.

Рисунок 6.10 – Декілька графіків, створених за допомогою багаторазового використання команди **plot**

У наступному прикладі ми будемо використовувати команду **line**. Зверніть увагу, що команда **line** дозволяє побудувати тільки одну залежність за один раз. Якщо вам потрібно побудувати декілька кривих, ми повинні використовувати дану команду декілька разів.

```
clf;
x = linspace(0,6*pi,600);
y1 = sin(x);
y2 = sin(x)–1;
y3 = sin(x)–2;
line(x,y1);
line(x,y2);
line(x,y3);
```

Результат показаний на рис. 6.11. Зверніть увагу, що цього разу всі лінії одного кольору. Команда **line** будує за замовчуванням 1-піксельну чорну лінію. Також відзначимо, що команда **line** завжди потребує два вектори *xy*, на відміну від команди **plot**.

Рисунок 6.11 – Декілька залежностей, побудованих за допомогою команди **line**

6.2. Властивості графіків

Команди **plot** і **line** мають різноманітні властивості, які можна встановити при виклику. Це включає в себе такі атрибути, як колір лінії, ширина лінії та стиль лінії. Зверніть увагу, що тільки завчасно можна встановити колір графіка, тип лінії та точки маркерів при виклику функції **plot**. Неможливо змінити ці дані після того, як графік буде створено.

6.2.1. Кольори графіка

FreeMat подає декілька механізмів для зміни кольору лінії на графіку. По-перше, забезпечується набір із семи основних кольорів, які можуть бути використані для команд **plot** і **line**. Ними є:

– 'b' – синій;

– 'g' – зелений;

– 'r' – червоний;

– 'c' – блакитний;

– 'm' – пурпурний;

– 'y' – жовтий;

– 'k' – чорний.

Ось декілька прикладів, які показують використання стандартних функцій. Перший приклад використовує команду **plot**.

```
x = linspace(–3,3,1000);
y = exp(–x.^2).*sin(10*pi*x);
plot(x,y,'r')
```

Рисунок 6.12 – Графік із використанням червоного кольору, створений командою **plot**

Ось ще один приклад, який використовує команду **line**.

```
close('all')
x = linspace(–3,3,1000);
```

```
line(x,y,zeros(1,length(x)),'color','y');
```

Відзначимо, що в даному випадку ми додали осі масиву (по суті масив нулів), оскільки команда **line** потребує масив *x*, *y* і *z* для того, щоб додати будь-який рядок властивостей (ширина, колір та т.ін.).

Рисунок 6.13 – Графік затухаючої синусоїди, створений за допомогою жовтого кольору та команди **line**

Далі ми будемо об’єднувати декілька графіків у одному вікні, а також використовувати різні кольори. Напишемо програму, яка дозволяє створювати графік, а потім виділяти його секцію певним кольором. Це буде реалізовано шляхом створення зигзагоподібної лінії, яка виділяє сектор, як показано на рис. 6.14 (Оригінальний метод використовує цикл **for**. Він працює дуже повільно. Запропонований метод працює швидко та ефективно).

Рисунок 6.14 – Виділення сектора графіка

У прикладі ми будемо використовувати метод вказаний вище. Створюємо криву Гаусса, а потім визначаємо ділянку від –1 до 0,3 для виділення:

```
clear all;
close('all');
% Створення кривої Гаусса, як анонімної функції
fcn =@(x) ((1/sqrt(2*pi))*exp((–x.^2)/2));
% Встановлення різних обмежень для графіка та виділеної частини
start_x = –3; % Нижня межа графіка
stop_x = 3; % Верхня межа графіка
start_point=–1; % Встановлення нижньої межі для підсвічування
stop_point = 0.3; % Встановлення верхньої межі для підсвічування
steps = 600;
% Створення x і y масивів для самої кривої.
xx = linspace(start_x,stop_x,steps);
yy = fcn(xx);
% Побудова кривої Гаусса
plot(xx,yy);
% Розрахунок значень по осі абсцис і та осі ординат для зигзагоподібної лінії.
x = linspace(start_point,stop_point,steps);
y = fcn(x); % Розрахунок початку виділеного розділу
c = 1:(2*steps); % Лічильник для створення значень по осі абсцис
c2 = 2:2:(2*steps); % Лічильник для створення значень по осі ординат
xx(c) = x(ceil(c/2)); %
yy = zeros(1,length(xx)); % Створення нульового вектора для осі ординат.
yy(c2) = y(c2/2);
line(xx,yy,zeros(1,length(xx)),'color','y'); % Ділянка виділеної частини
```

Запустивши програму, ми отримаємо рис. 6.15.

Рисунок 6.15 – Графік кривої Гаусса з виділеним сектором від –1 до 0,3, створений за допомогою команди **line**

Коли ви рисуєте декілька графіків у тому ж вікні за допомогою однієї команди **plot,** вона обирає колір в такому порядку: синій, зелений, червоний, блакитний, пурпурний, жовтий, чорний. Приклад з використанням трьох кольорів (синій, зелений, червоний) показано на рис. 6.9. Кольори визначаються за допомогою масиву 7x3, який має кольоровий баланс RGB (червоний / зелений / синій). Ви можете побачити це, використовуючи команду **get** з опцією **colororder**:

```
close('all')
figure(1)
get(gca,'colororder')
ans =
0          0          1.0000
0          0.5000     0
1.0000     0          0
0          0.7500     0.7500
0.7500     0          0.7500
0.7500     0.7500     0
0.2500     0.2500     0.2500
```

Кожен рядок являє собою набір із трьох змінних, причому кожен стовпець визначає кількість червоного, зеленого та синього відповідно, 0 означає відсутність кольору, а 1 означає повний колір.

При виведенні на друк однієї змінної, використовуючи команду **plot**, ви завжди будете за замовчуванням використовувати перший колір у масиві, це буде синій ([0 0 1.0]). Як змінити цей масив ми розкажемо дещо пізніше.

Можна використовувати нестандартні кольори з командами **plot** і **line**. Ми вважаємо, що команда **line** простіша для встановлення нестандартних кольорів. Це пов'язано з тим, що для того, щоб використовувати нестандартні кольори з командою **plot**, ми повинні відкрити вікно фігури, відключити використання команди **hold**, потім за допомогою команди **set** разом із опцією **coloroder** створити колір, який ви хочете. Тільки потім ви можете використовувати команду **plot**, щоб створити графік. З командою **line**, з іншого боку, ви просто використовуєте установки кольору в самій команді за допомогою трьох-елементного масиву для установки бажаного кольору шляхом використання стандартної шкали RGB (червоний/зелений/синій). З командою **plot** необхідно використовувати команду **hold**, перш ніж змінювати набір вектора кольорів. Зауважимо, що вектор **colororder** належить тільки до команди **plot**, він не має відношення до команди **line**. Крім того, незалежно від того яка кольорова палітра використовується в **colororder**, якщо один із конкретних основних кольорів указаний в команді **plot**, він буде використовуватися як основний колір, а не один із кольорів вектора.

Приклад створення графіків нестандартних кольорів. Методи, які використовуються для нестандартної розмальовки, різняться залежно від того, яку команду ви використовуєте. Методи для обох команд показані нижче. Для кожного прикладу ми створимо основні форми хвилі та використаємо коричневий колір лінії. Коричневий колір ми отримаємо поєднанням 144/256 червоного, 88/256 зеленого і 0 синього. Як перший приклад ми будемо використовувати команду **plot**.

```
clear all;
close('all')
x = linspace(0,6,600);
y = x.*sin(10*x).*exp(–x);
figure(1);
hold on;
set(gca,'colororder',[(144/256) (88/256) 0]);
```

```
plot(x,y);
hold off;
```

Результати показані на рис. 6.16.

Рисунок 6.16 – Приклад графіка коричневого кольору, отриманого за допомогою команди plot

Ось подібний приклад, але з використанням команди line:

```
clear all;
close('all')
x = linspace(0,6,600);
y = x.*sin(10*x).*exp(-x);
z = zeros(1,length(x));
line(x,y,z,'color',[(144/256) (88/256) 0]);
```

Результати цієї програми ідентичні з результатами команди **plot**, як показано на рис. 6.17. Єдина відмінність у тому, що ми повинні були включати вісь *z*, разом з тим ми не використовували команди **hold** і **set**.

Рисунок 6.17 – Приклад графіка коричневого кольору, отриманого за допомогою команди **line**

6.2.2. Настройки ширини лінії

За замовчуванням FreeMat рисує тонку лінію. Можна, однак, встановити ширину лінії більшу, використовуючи властивості **linewidth**. Синтаксис для команди **plot** виглядає таким чином:

plot(x,'linewidth',n),

де *x* – змінна графіка; *n* – ціле число від 1–32, яке регулює товщину лінії. Чим вище число, тим товстіша лінія.

Синтаксис команди **line** виглядає таким чином:

line(x,y,z,'linewidth',n) де: *x*, *y*, *z* – відповідно масив осей *X*, *Y* і *Z*; *n* – ціле число, яке подає ширину лінії від 1–32.

Відзначимо, що FreeMat при використанні команди **plot** незалежно від кількості ліній дозволяє встановити тільки одну ширину лінії. Ви можете встановити різні ширини ліній для різних кривих, але вам прийдеться використовувати окрему команду **plot** у поєднанні з командою **hold**, як це зробити (див. нижче).

Приклад настройки ширини лінії. Цей приклад покаже різницю в ширині ліній для нормального відображення (рис. 6.18–6.21).

```
x = linspace(0,4*pi,600);
y = cos(x);
plot(x,y,'linewidth',1);
```

Рисунок 6.18 – Ділянка синусоїди із шириною лінії 1

```
x = linspace(0,4*pi,600);
y = cos(x);
plot(x,y,'linewidth',2);
```

Рисунок 6.19 – Ділянка синусоїди із шириною лінії 2

```
x = linspace(0,4*pi,600);
y = cos(x);
plot(x,y,'linewidth',6);
```

Рисунок 6.20 – Ділянка синусоїди із шириною лінії 6

```
x = linspace(0,4*pi,600);
y = cos(x);
plot(x,y,'linewidth',32);
```

Рисунок 6.20 – Ділянка синусоїди із шириною лінії 32

Нарешті, приклад побудови за допомогою команди **line**. Зверніть увагу, що команда **line** вимагає змінну осі *z* (у даному випадку просто масив нулів), якщо ви хочете змінити ширину лінії, колір чи щось інше, використовуйте властивості команди **line**.

```
clear all;
close('all');
x = linspace(0,4*pi,600);
y = cos(x);
z = zeros(1,length(x));
line(x,y,z,'linewidth',10);
```

Рисунок 6.21 – Графік синусоїди з шириною лінії 10, побудований за допомогою команди **line**

Наступний приклад демонструє, як змінюється ширина лінії в команді **plot**, яка будує кілька залежностей.

```
x = linspace(0,6*pi,600);
y1 = sin(x);
y2 = sin(x)–1;
y3 = sin(x)–2;
plot(x,y1,x,y2,x,y3,'linewidth',6)
```

Результат показано на рис. 6.22.

Рисунок 6.22 – Декілька кривих на одному графіку, побудованих за допомогою команди **plot**

Ось приклад, який використовує дві команди **line** з різною шириною ліній та різними кольорами. Спочатку будується жовта лінія товщиною 6, а потім чорна товщиною 2.

```
clear all;
close('all');
x = linspace(0,6,600);
y = x.*sin(2*pi*x).*exp(-x);
z = zeros(1,length(x)); % Нульовий вектор для осі z
line(x,y,z,'color','y','linewidth',6);
line(x,y,z,'linewidth',2);
```

Як результат отримано такий графік (рис. 6.23).

Рисунок 6.23 – Графік лінії, яка"світиться", створений з використанням ліній різної ширини та різних кольорів

Наступна програма схожа на попередню, різниця полягає в тому, що перша лінія (товста) буде чорною, а друга (тонка) буде жовтою.

```
clear all;
close('all');
x = linspace(0,6,600);
y = x.*sin(2*pi*x).*exp(-x);
z = zeros(1,length(x));
line(x,y,z,'linewidth',6);
line(x,y,z,'color','y','linewidth',2);
```

Отримуємо такий графік (рис. 6.24).

Рисунок 6.24 – Графік складається із двох ліній різної ширини та різних кольорів

6.2.3. Типи ліній графіків

FreeMat надає декілька різних типів ліній. До них нележать:

'-' – (дефіс) суцільний стиль лінії;

':' – (двокрапка) точковий стиль лінії;

'-.' – (дефіс точка) стиль лінії крапка-тире-крапка-тире;

'--' – (два дефіси підряд) пунктирний стиль лінії.

Приклад побудови графіків із використанням різних типів ліній. В першому прикладі використовуємо команду **line**. Використання таких команд:

```
close('all')
t = linspace(0,8*pi,256);
y = sin(t)–(1/3)*sin(3*t)+(1/5)*sin(5*t);
line(t,y,zeros(1,length(t)),'color','g','linestyle',':');
```

дає графік, зображений на рис. 6.25. Цей графік зеленого кольору та побудований пунктирною лінією. Зелений колір привласнюється при

написанні літери "*g*" в настройках кольору; пунктирна лінія з'являється після того, як ви поставите двокрапку у властивостях стилю ліній **LineStyle**. Зверніть увагу, що ми використовували нульовий вектор для створення осі *z*.

Рисунок 6.25 – Графік побудований з використанням пунктирної лінії

Змінимо команду, щоб створити суцільну чорну лінію, як показано на рис. 6.26.

```
close('all')
t = linspace(0,8*pi,256);
y = sin(t)–(1/3)*sin(3*t)+(1/5)*sin(5*t);
line(t,y,zeros(1,length(t)),'color','k','linestyle','–');
```

Рисунок 6.26 – Графік побудований з використанням чорної суцільної лінії

6.2.4. Точки маркера

FreeMat дозволяє показати фактичні точки як маркери. Маркери можуть бути таких типів:

'.' – (період) символ точка;

'o' – (мала літера "o") символ коло;

'x' – (мала літера "x") символ помножити;

'+' – (очевидно) символ плюс;

'*' – (очевидно) символ Asterisk;

's' – (мала літера "s") символ площі;

'd' – (мала літера "d") символ діамант;

'v' – (мала літера "v") символ, що вказує вниз трикутний;

'^' – (очевидно) символ, що вказує вгору трикутний;

'<' – (очевидно) символ менше;

'>' – (очевидно) символ більше.

Символи маркера можуть знаходитись на лінії або бути використані окремо.

Приклад використання маркерів. Ось коротка програма, яка створить рисунок, схожий на рис. 6.23 і рис. 6.24, але з точками маркера (квадрати у даному випадку), а не суцільною лінією.

```
clear all;
close('all');
x = linspace(0,6,57);
y = x.*sin(2*pi*x).*exp(–x);
plot(x,y,'ks');
```

Результат на рис. 6.27.

Рисунок 6.27 – Графік кривої з використанням квадратних маркерів точок

Тепер ми до вказаного вище рисунка додамо лінію

```
clear all;
close('all');
x = linspace(0,6,57);
y = x.*sin(2*pi*x).*exp(–x);
plot(x,y,'ks');
```

Результат на рис. 6.28.

Рисунок 6.28 – Графік кривої, проведеної через точки квадратних маркерів

Ми можемо використовувати інші форми точок маркера. Наприклад, замість квадратів можемо використовувати кола (мала літера "o"). Використовуємо також кольори за замовчуванням.

```
clear all;
close('all');
x = linspace(0,6,57);
y = x.*sin(2*pi*x).*exp(–x);
plot(x,y,'o–');
```

Результат на рис. 6.29.

Рисунок 6.29 – Графік кривої з використанням малих точок кола маркера, а також безперервної лінії

Далі продемонструємо використання команди **line** для створення точок маркера.

```
clear all;
close('all');
x = linspace(0,6,57);
y = x.*sin(2*pi*x).*exp(–x);
z = zeros(1,length(x));
line(x,y,z,'marker','o');
```

Результат на рис. 6.30.

Рисунок 6.30 – Графік кривої з використанням суцільної лінії, а також невеликих круглих маркерів

Зверніть увагу, що команда **line** за замовчуванням рисує суцільну лінію, якщо не вказано інше. Таким чином, при використанні команди **line**, якщо ви не хочете, щоб лінія була нарисована, вкажіть "none" в опції **LineStyle**. Ось приклад.

```
clear all;
close('all');
x = linspace(0,6,57);
y = x.*sin(2*pi*x).*exp(–x);
z = zeros(1,length(x));
line(x,y,z,'linestyle','none','marker','o');
```

Результат на рис. 6.31.

Рисунок 6.31 – Графік кривої з використанням малих маркерів точок кола

Ця крива була створена за допомогою команди **line**. У настройках **LineStyle** було встановлено "none".

Ось приклад точкової діаграми, тобто графік наборів точок *x* і *y*. При цьому використовується команда **line**:

```
clear all;
close('all');
x = randn(1,2000);
y = randn(1,length(x));
z = zeros(1,length(x));
line(x,y,z,'linestyle','none','marker','.');
```

Рисунок 6.32 – Точкова діаграма, створена командою **line**

6.3. Покращення вигляду графіків

Є декілька способів для покращення зовнішнього вигляду і наочності вашого графіка. До них належать настройки меж *x* і *y* осей,

назви осей, додавання легенди і зміна розміру графіка. Важливою особливістю цих команд є те, що вони повинні бути викликані після того, як графік було побудовано (як командою **plot,** так і **line**).

6.3.1. Настройка горизонтальних і вертикальних меж графіка

Можливо, ви помітили, що у деяких випадках графік не повністю заповнює площу рисунка. Це тому, що FreeMat сам встановлює межі. Ви можете "вручну" встановити межі. При цьому використовуються функції **xlim** і **ylim** (FD, с. 508 і 510, відповідно). Для початку після того, як побудовано ваш графік і ви хочете побачити межі даних (як на осі абсцис, так і осі ординат), ви можете використовувати команду **get** з опцією **datalimits**.

Для установки горизонтальних меж синтаксис

xlim([lo, hi]),

де lo і hi є початком (крайній ліворуч) і кінцем (крайній праворуч), які обмежують графік.

Для установки вертикальних меж синтаксис

ylim([lo, hi]),

де lo і hi саме нижнє (внизу) і саме верхнє (вверху) обмеження, відповідно.

Приклад установки горизонтальних та вертикальних границь графіка. Ми почнемо з графіка, який має кілька відносних максимумів і мінімумів.

```
clear all;
close('all');
x = linspace(–2,3,600);
y = x.*cos(x.^2);
plot(x,y);
```

Рисунок 6.33 – Графік кривої, яка має декілька максимумів і мінімумів

По-перше, ми подивимося на діапазон даних:

```
get(gca,'datalimits')
ans =
    –2.0000   3.0000   –2.7334   2.5143   –0.5000   0.5000
```

Це шестиелементний вектор. Перші два елементи є мінімальними і максимальними по осі абсцис, другий і третій елементи – мінімум і максимум по осі ординат, а останні – два мінімум і максимум по осі аплікат. Зверніть увагу, що дані по осі аплікат не використовуються.

Ми можемо задати межі по осі *x* від 0 до – 2:

```
xlim([0,2])
```

Результат показаний на рис. 6.34. Зауважимо, що зміна меж візуалізації графіка не змінює даних, як це передбачено в настройках **datalimits**:

```
get(gca,'datalimits')
ans =
    –2.0000   3.0000   –2.7334   2.5143   –0.5000   0.5000
```

Рисунок 6.34 – Графік зі встановленими горизонтальними межами побудови

Зверніть увагу, що у нас є великий простір між максимальною точкою на графіку і верхньою частиною рисунка. Ми можемо відрегулювати вертикальні межі так, щоб графік заповнював велику площу рисунка:

```
ylim([–2,1])
```

Результати показані на рис. 6.35.

Рисунок 6.35 – Графік зі встановленими як горизонтальними, так і вертикальними обмеженнями

6.3.2. Зміна розмірів графіка

Ви можете змінити розмір всього вікна рисунка за допомогою функції **sizefig** (FD, с. 487) або за допомогою миші, захопивши сторону або кут, натиснувши та перетягнувши їх для потрібних розмірів вікна. У рамках самого вікна ви можете настроїти взаємне розташування ділянок.

На рис. 6.36 зображено структурні елементи рисунка

Рисунок 6.36 – Структурні елементи рисунка

Для того щоб знайти розмір рисунка ви повинні зрозуміти як сам рисунок і вікно співвідносяться один до одного. Вікно bar (на самому верху) та різні меню інструментів (текстові, іконки) не враховуються в розмір рисунка. Розмір ділянки рисунка складається з вікна, зазначеного як "outerposition". Вікно, зазначене як "position", є ділянкою, в якій графік буде побудований.

Щоб побачити розмір поточного рисунка використовуйте:

get(gcf,'figsize').

Це одна із властивостей рисунка (FD, с. 461). Зверніть увагу, що коли ви хочете побачити властивості рисунка, використовуйте команду **gcf** (отримати поточну фігуру) на початку команди **get**. Якщо ви хочете побачити властивості осей (FD, с. 441), використовуєте **gca** (отримати поточні осі) на початку команди **get**. Щоб змінити розмір фігури використовуйте команду **sizefig**, яка використовує такий синтаксис:

sizefig(x,y),

де *x* – загальний розмір вікна по горизонталі; *y* – розмір сірої ділянки по вертикалі. Вертикальний розмір вікна буде цей номер плюс 56 пікселів.

Ця функція також викликається окремо від функції **plot** і повинна бути запущена після неї. Розмір фактичної площі побудови, тобто "позиції" коробки, як показано на рис. 6.36 буде меншим, ніж зазначено в **sizefig**. Команда **sizefig** встановлює розмір вікна зазначеного як "outerposition" на рис. 6.36. Ви можете побачити відносний розмір ділянки побудови за допомогою такої команди:

get(gca,'position')

Ви можете регулювати відносний розмір ділянки у вікні, використовуючи таку команду:

set(gca,'position',[x_start y_start x_size y_size])

Приклад. Перегляд розміру та зміна розміру ділянки побудови. У цьому прикладі ми будемо дивитися на поточний розмір, а також зміну розміру ділянки:

```
clear all;
close('all');
x = linspace(-2,3,600);
y = x.*cos(x.^2);
plot(x,y);
```

Ці команди створять ділянку, показану на рис. 6.37.

Рисунок 6.37 – Рисунок з певними розмірами вікна

Для перегляду поточного розміру ми використовуємо команду **get** з опцією **figsize**. Оскільки це властивість рисунка, ми повинні використовувати номер рисунка (команду **gcf**), а не номер осі (команду **gca**).

```
get(gcf,'figsize')
ans =
600   335
```

Це означає, що розмір "outerposition" – це вікно 600 пікселів в ширину і 335 пікселів у висоту. Ми можемо також проглянути відносний розмір ділянки побудови, використовуючи команду **get** з опцією **position**. Ця властивість належить до осі, тому ми повинні використовувати **gca**. Так різниця між "віссю" і "фігурою" може викликати плутанину.

```
get(gca,'position')
ans =
0.1000   0.1000   0.8000    0.8000
```

Опція **position** повертає чотирьохелементний вектор. Перші два розміри є відстанями від нижнього лівого кута ділянки рисування до лівого кута ділянки всієї фігури. Першим елементом відображається відстань по осі *x*, а другим по осі *y*. Третій і четвертий елементи відносна ширина і висота прямокутника, в якому знаходиться крива. У цьому прикладі вікно "outerposition" 600 пікселів в ширину і 335 пікселів у висоту. Відносний розмір вікна кривої 0,8x0,8, або 600x0,8 = 480 пікселів у ширину і 335x0,8 = 268 пікселів у висоту. Ми можемо об'єднати опції **figsize** і **position** для створення набору команд, який дасть нам абсолютну ширину і висоту поточного рисунка, а саме:

```
get(gca,'position')(3)*get(gcf,'figsize')(1)
```

```
ans =
480
get(gca,'position')(4)*get(gcf,'figsize')(2)
ans =
268
```

Рисунок 6.38 – Опція **position** повертає відносне розміщення, яке нижче лівого кутка ділянки, а також відносний розмір (опція **figsize**) ділянки

Тепер ми змінимо розмір рисунка на менший, хоча можна залишати будь-який розмір.

```
sizefig(300,200)
```

Результати показано на рис. 6.39.

Рисунок 6.39 – Зменшений рисунок

6.3.3. Додавання підписів до рисунка

Після створення рисунка ви можете додати підписи. До них належать назви осей *x* і *y*, текстові мітки.

6.3.3.1. Додавання назви рисунка

Ви можете додати назву на рисунок, яка буде розташовано трохи вище графіка, використовуючи команду **title** (FD, с. 498). Загальний синтаксис:

title('Put Your Title Here')

Функція **title** включає кілька опцій: положення, розмір шрифту, колір фону (коло заголовка, а не на весь графік) та інші властивості. Властивості назви включають в себе все, що може бути віднесене до **textproperties** (FD, с. 497).

Приклад додавання назви рисунка. У цьому першому прикладі ми додамо назву рисунку за замовчуванням, а потім використаємо більший 20 шрифт (який вимагає переробки назви). За замовчуванням використовується 8 шрифт (шрифт 8 пікселів у висоту):

```
clear all;
close('all');
x = linspace(–2,3,600);
y = x.*cos(x.^2);
plot(x,y);
title('Plot of Curves')
```

Ось результат.

Рисунок 6.40 – Рисунок з назвою, створений за допомогою 8 шрифту

Тепер ми зробимо назву 20 шрифтом:

```
title('Plot of Curves','fontsize',20);
```

Результат на рис. 6.41.

Рисунок 6.41 – Рисунок з назвою створений за допомогою 20 шрифту

Надалі ми розглянемо ще деякі властивості для назви, у тому числі конкретне положення, вирівнювання тексту та інший колір фону. Назва може бути розташована як всередині вікна "outerposition", а також відповідно до вирівнювання тексту. Рядок заголовка може бути вирівняний за дев'яти різними точками, як показано на рис. 6.42. За замовчуванням вирівнювання підпису буде зверху по центру. Ви можете встановити вирівнювання, використовуючи дві різні властивості тексту, horizontalalignment і verticalalignment.

Опція **position** буде визначати, де рядок заголовка буде поміщатися в ділянку. Позицією за замовчуванням є [0,5 1], яка зосереджена у верхній частині дисплея.

Рисунок 6.42 – Вирівнювання точок тексту, показаних *X*-ми. Є три за горизонтальною і вертикальною віссю, що становить дев'ять позицій

Приклад позиціонування назви рисунка та настройки вирівнювання. Ми продовжимо попередній приклад, але тут будемо задавати положення заголовка і вирівнювати його текст.

```
title('Plot of Curves','fontsize',20,'position',[0.250.9],
'verticalalignment','middle')
```

Результат показний на рис. 6.43.

Рисунок 6.43 – Позиціонування та вирівнювання назви рисунка. У цьому випадку *x*-position було встановлено на 0,25 зліва і 0,9 знизу загального розташування рисунка

6.3.3.2. Настройка підписів осей x (по горизонталі) та y (по вертикалі)

Тепер ми додамо мітки для горизонтальної та вертикальної осей. Це робиться за допомогою команд **xlabel** (FD, с. 507) та **ylabel** (FD, с. 509). Деякі з властивостей тексту застосовуються до **xlabel** і **ylabel**, а деякі ні. Наприклад, опція розмір шрифту може бути застосована до цих двох команд, але властивості вирівнювання і положення не можуть бути застосовані.

Примітка. Положення тексту при використанні команди xlabel. На момент написання посібника існувала помилка в FreeMat 4.0, нижній підпис рисунка знаходиться за межами видимості. Це проблема для розміру вікон (600 x 335 пікселів для Linux; 600 x 344 для Windows). Обійти цю проблему можна за допомогою використання опції ***position*** *команди* ***get****, змінюючи розміри внутрішнього розташування рамки таким чином, щоб текст став повністю видимим. Існує спеціальна функція, яка називається* ***labelSet*** *(додаток Б): вона складається з команд для зміни розміру ділянки з підписом, які будуть використовувати цю команду, і деяких спеціальних розрахунків, щоб змінити розмір вашої ділянки так, щоб весь текст (підписи осей xlabel, ylabel) було нормально видно.*

Синтаксис команд:

xlabel('Put the horizontal axis label here')

ylabel('Put the vertical axis label here ')

Приклад введення підписів горизонтальної осі. Ми знову використовуємо попередній приклад (мінус назва) і додаємо підпис горизонтальній осі.

```
clear all;
close('all');
x = linspace(-2,3,600);
y = x.*cos(x.^2);
plot(x,y);
```

Цей код забезпечує основну побудову (без назви). Далі ми додамо підпис осі *x*.

```
xlabel('Put the horizontal axis label here')
```

Результат показано на рис. 6.44. У цьому випадку ми не використовували всі доступні опції. Тут напис обрізаний і видно тільки верхню частину. Це помилка FreeMat (яка, мабуть, буде виправлена в майбутньому релізі).

Рисунок 6.44 – Додавання горизонтального підпису осі

У цьому прикладі команда **xlabel** створює підпис на горизонтальній осі, але наполовину обрізаний. У той же час ми створили функцію, яка має назву **labelSet** (лістинг у додатку Б: спеціальна функція для зміни розміру ділянки підпису осей), яку можна використовувати для вирішення цієї проблеми. Ця функція перевіряє наявність підписів **xlabel** або **ylabel**, дивиться на їх розміри і загальний розмір рисунка, а потім обчислює відповідний розмір площі і положення.

Результатом роботи цієї функції є те, що графік змінено таким чином, щоб горизонтальний підпис було видно (рис. 6.45).

```
labelSet
```

Рисунок 6.45 – Графік, але після запуску функції labelSet, описаної вище. Ця функція змінила розмір графіка так, щоб підпис горизонтальної осі було видно

Давайте повторно виконаємо команду **xlabel**, але з більшим шрифтом. Ми будемо також використовувати функцію **labelSet**, яку ми створили для позиціонування підпису.

```
xlabel('Here Is A Label for the X-Axis','fontsize',16)
labelSet
```

Результат тут.

Рисунок 6.46 – Інший підпис осі *X*, тепер 16 шрифтом. Примітно, що функція **labelSet** (лістинг в додатку Б) використовувалася для позиціонування напису

Тепер ми додамо вертикальну вісь (осі ординат), використовуючи 16 шрифт для підпису, результат на рис. 6.47:

```
labelSet function.
ylabel('Here is a Label for the Y-Axis','fontsize',16)
labelSet
```

Рисунок 6.47 – Підписи до обох осей зроблені 16 шрифтом

6.3.3.3. Додавання легенд

Ви можете додати легенду, яка опише, що кожна лінія графіка означає. При цьому використовується команда **legend** (FD, с. 473), яка вимагає, щоб одному рядку відповідала одна крива в межах графіка. Синтаксис виглядає таким чином:

legend('String for trace 1', 'String for trace 1')

Зверніть увагу, що ви використовуєте лише одну команду **legend** для кожного рисунка. Не намагайтеся використовувати одну команду **legend** для кожної кривої, ви просто перезапишете попередню команду **legend**.

*Примітка. Додавання легенди і команда **hold**. Ми помітили деякі проблеми при використанні функції **legend** після використання команди **hold on**. Тому, якщо ви використовуєте команду **hold on**, ми не рекомендуємо вам додавати легенду, поки команда **hold off** не була видана.*

Команда **legend** має один аргумент, який може бути скоректований. Це опція місця. Вона може бути вибраною із восьми напрямів: північ, південь, схід і захід, а інші чотири напрями між ними (північно-східний, південно-східний, південно-західний, північно-західний). За замовчуванням ця властивість має значення "північний-схід" (мається на увазі верхній правий кут графіка).

Ось декілька прикладів.

Приклад додавання легенди на рисунок. Ми почнемо з однієї кривої на рисунку.

```
clear all;
close('all');
x = linspace(-2,3,600);
y = x.*cos(x.^2);
plot(x,y);
legend('This is a basic curve')
```

Цей код створює рисунок 6.48.

Рисунок 6.48 – Базовий графік з однією кривою та легендою

Ви можете регулювати розташування легенди в лівий нижній кут ("південно-захід") таким чином:

```
legend('This is a basic curve','location','southwest')
```

Як результат отримуємо графік (рис. 6.49).

Рисунок 6.49 – Графік з легендою, що “переїхала” в нижній лівий (південний захід) кут

Ми створимо іншу криву, яка буде диференціалом залежності, показаної вище. Потім додамо легенди для кожної кривої (оригінал і диференціал). При цьому використовуємо команду **diff** (FD, с. 202).

```
clear all;
close('all');
x = linspace(-2,3,600);
y = x.*cos(x.^2);
plot(x,y);
dy = diff(y);
dx = diff(x);
ds = dy./dx;
w=dx(1);
xx=linspace(–2,(3-w),length(dy));
line(xx,ds);
legend('This is a basic curve','Differential of curve','location',
'southwest');
```

Результат програми показаний на рис. 6.50.

Рисунок 6.50 – Рисунок із зображенням двох кривих з легендами

Ми можемо використовувати ще один приклад з легендою для того, щоб показати різницю між гіперболічними функціями синус і косинус (рис. 6.51).

```
x = linspace(-5,5,600);
y1 = sinh(x);
y2 = cosh(x);
plot(x,y1);
line(x,y2,zeros(1,length(x)),'linewidth',2);
```

```
legend('Sinh(x)','Cosh(x)','location','southeast');
```

Рисунок 6.51 – Порівняння гіперболічних синуса та косинуса від –5 до +5

6.3.3.4. Додавання пояснюючого тексту

Є можливість також додати текст безпосередньо до графіка за допомогою команди **text** (FD, с. 496). Ця команда має синтаксис:

text(x,y,<string>),

де *x* – точка на осі абсцис, з якої буде починатися лівий край тексту; *y* – точка на осі ординат, де буде знаходитися середина тексту; <string> – це текст, введений як стандартний рядок, який буде відображатися на рисунку.

Приклад додавання пояснюючого тексту. У цьому прикладі ми будемо використовувати рисунок із попередніх прикладів. Різниця в тому, що замість використання команди **legend** для підписання різних кривих на графіку ми будемо використовувати текст поруч з кожною кривою (рис. 6.52).

```
x = linspace(–5,5,600);
y1 = sinh(x);
y2 = cosh(x);
plot(x,y1,'linewidth',2);
line(x,y2,zeros(1,length(x)),'linewidth',2);
text(–4,–40,'Hyperbolic sine');
text(–4,40,'Hyperbolic cosine');
```

Рисунок 6.52 – Використовуючи команду **text**, ми можемо написати текст прямо на графіку. У цьому випадку ми використовуємо його для описання різних кривих і не використовуємо команду **legend**

6.3.4. Додавання сітки

Щоб додати сітку на графік потрібно набрати:

```
grid on
```

Щоб прибрати сітку використовуйте команду:

```
grid off
```

6.4. Робота з декількома графіками

Коли ви створюєте графік, він отримає номер. Зверніть увагу, як у вікні рисунка пишеться у верхній частині: "Figure *X*". *X* є номером рисунка. Перший за замовчуванням дорівнює 1. Поки ви не зміните номер рисунка, будь-яка інша побудова буде спрямована в це ж вікно. Таким чином, ви можете випадково перезаписати рисунок, коли ви хотіли б мати два (або більше) рисунків.

FreeMat забезпечує простий спосіб управління кількома рисунками за допомогою команди **figure(x)** (FD, с. 461). Ви можете використовувати цю команду, щоб зробити рисунок *x* активним. Якщо рисунка *X* ще не існує, FreeMat створює пусте вікно. Це вікно залишиться активним доти, поки ви не зміните його. Є три способи, щоб змінити активний рисунок. Перший – це використання команди **figure**. Другий, якщо ви закриєте вікно рисунка. Наприклад, якщо у вас є два відкритих рисунками (1 і 2), і ви закриваєте 1, то 2 стає активним рисунком. Якщо закрити обидва, то рисунок 1 стане активним знову. Третій спосіб полягає у використанні миші для натискання на той рисунок, який ви хочете зробити активним.

На жаль, немає ніякого способу, щоб визначити загальну кількість рисунків, відкритих у будь-який момент часу. Тим не менш, ви можете використовувати команду **gcf** (отримати поточний рисунок) (FD, с. 462) для визначення поточного активного рисунка.

Приклад програми створення декількох рисунків.

```
t = 1:128;
x = sin(2*pi*t/32);
plot(x) % Створюється Рисунок 1
y = cos(2*pi*t/16);
figure(2) % Створюється Рисунок 2
plot(y)
```

6.5. Збереження вашого рисунка

Після обробки деяких даних, що створюють зображення (при установці кольору), зміни його розміру, установки вертикальних і горизонтальних меж і додавання описових міток, є можливість швидко, безболісно і легко зберігати його у вигляді графічного файлу.

Щоб зберегти активну ділянку рисунка, використовуйте функцію **print** (FD, с. 483). Вона має такий синтаксис:

print('filename.png')

Ця функція зберігає файли як PNG (Portable Network Graphics) зображення. Інші допустимі розширення JPG,. PDF і SVG. Власно кажучи, на наш погляд, PNG виглядає найкраще.

Приклад збереження зображення.

```
t = linspace(0,2,1000);
f = 3; % Частота в Гц.
y = cos(2*pi*f*t);
plot(t,y)
print('testfile.png')
```

Отримане зображення показано на рис. 6.53. Зазначимо, що оскільки повністю шлях не був вказаний, то файл був збережений у робочому каталозі.

Рисунок 6.53 – Рисунок, збережений у вигляді зображення

Якщо ви не прописали повністю шлях у команді **print**, то зображення буде збережено в свій робочий каталог. Для підтвердження того, що ваш файл був збережений правильно, ви можете використовувати стандартні інструменти файлів (Провідник Windows, Linux Наутілус і т.ін.) або роздрукувати вміст поточного каталогу за допомогою команди **ls** (FD, с. 419).

7. ЛАБОРАТОРНИЙ ПРАКТИКУМ

У даному розділі подано лабораторні роботи для практичного засвоєння курсу. Вони викладені у тій послідовності, в якій написано посібник, тобто завдання першої лабораторної роботи відповідають матеріалу першого розділу. Відповіді на всі запитання та завдання можна знайти в посібнику або довідці FreeMat. Розділу 6 “Графіки та рисунки” присвячено дві лабораторні роботи, відповідно 6 і 7, це зумовлено чималим обсягом цього розділу. Остання лабораторна робота “Розв’язання СЛАР” дещо виходить за межі матеріалу посібника, тому нами наведено приклад її виконання.

7.1. Лабораторна робота 1. Перші кроки у FreeMat

1. Визначте поточну директорію за допомогою команди **pwd**.

2. Змініть робочу директорію за допомогою команди **cd**.

3. Дізнайтеся перелік шляхів за допомогою команди **path**.

4. Задайте декілька значень x, y, z. Подивіться на запис цих змінних у робочій ділянці.

5. Подивіться перелік елементарних функцій за допомогою довідки.

6. Обчисліть $\sin(1)$, $\cos(x)$, $\mathrm{tg}(yz)$.

7. Запишіть число Pi/2 у різних форматах.

8. Створіть три рядки, які будуть включати ваше ім’я, призвіще та по батькові. Далі проведіть конкатенацію цих рядків.

9. Запишіть комплексні числа $X = 5 + 7i$ і $Y = 3 + 2i$. Обчисліть $|X||Y|$, $\mathrm{Re}(X) / \mathrm{Im}(Y)$.

7.2. Лабораторна робота 2. Математичні операції

1. Проведіть операції додавання, віднімання, множення та ділення чисел 9 і 3.

2. Задайте вектор, який буде складатись із чисел дати вашого народження (наприклад 01.01.01) та знайдіть суму, добуток, накопичену суму та накопичений добуток цих чисел.

3. Обчисліть факторіал числа 10.

4. Піднести до степеня 5 вектор вашого народження двома способами.

5. Проведіть операцію експоненціювання з вектором народження.

6. Обчисліть логарифм вектора народження за основами 10, 2 та *e*.

7. Обчисліть всі стандартні тригонометричні функції для кута Pi/2.

8. Витріть вектор вашого народження.

9. Збережіть текст робочого вікна.

7.3. Лабораторна робота 3. Матриці та масиви

1. Задайте тривимірний масив $B(3*3*3)$ з випадковими елементами.

2. Подивіться та змініть значення елементів $B(2,2,1)$ та $B(3,2,3)$.

3. Видаліть з масиву B другий рядок та перший стовпець.

4. Створіть двома способами масив послідовності від 1 до 100 з кроком 5.

5. Створіть логарифмічний масив від 1 до 10 з кроком 2.

6. Подивіться як застосовуються функції **ones**, **zeros** до багатовимірних масивів.

7. Створіть матриці A і B розміром $4*4$. Проведіть операції $A+B$, $A-B$, $A/B+2*|A|$, A^{-1}.

8. Проведіть поелементні операції додавання, віднімання, множення та поділу над матрицями A і B.

7.4. Лабораторна робота 4. Програми та функції

1. Створіть та запустіть m-файли, які будуть рисувати графіки функцій: $y_1(x) = \exp(-x^2\cos(8\pi x))$; $y_2(x) = x^2 + \frac{1}{x^3}$;

$y_3(x) = 2\cos x + \sin^2 3x$.

2. Створіть функцію для обчислення факторіала.

3. Створіть функції для обчислення таких виразів: $f_1(x) = 3\cos x + 5tgx$; $f_2(x) = 5shx + 5th^2x$; $f_3(x) = \cos 2x*(1-\frac{1}{4}\sin^2 2x)$.

4. Створіть анонімні функції $f_1(x) = \exp(x^2)$; $f_2(x) = x\sin^2 x$ та розрахуйте їх значення у декількох точках.

5. Введіть за допомогою функції **printf** назву вашого університету з переносом кожного слова на новий рядок.

6. Введіть за допомогою функції **printf** вашу дату народження.

7. Введіть довільні дані з клавіатури за допомогою функції **input**.

7.5. Лабораторна робота 5. Цикли та оператори порівняння

1. Створіть функції для обчислення sin , cos та tg з використанням ряду Маклорена. Порівняйте результати отримані за допомогою створених функцій з улаштованими у FreeMat.

2. Створіть дві послідовності даних та порівняйте їх між собою всіма операторами порівняння.

3. Задайте два масиви послідовності та порівняйте їх за допомогою операцій І та АБО.

4. Створіть функцію для обчислення найбільшого загального дільника двох чисел. Для перевірки візьміть числа 720 і 36.

5. За допомогою оператора **if** порівняйте два числа з видачею найменшого із них.

7.6. Лабораторна робота 6. Графіка

1. Побудуйте коло за допомогою команди **plot** у рівних осях.

2. Побудуйте графік функції $y = \sin(\ln(x^2))$ зеленою штрих-пунктирною лінією.

3. Побудуйте зірку в рівних осях.

4. На одному графіку побудуйте 4 криві $y = \cos^2 x, y = e^x$, $y = x\sin x, y = e^{-0.2x}\sin^2 x$ різним кольором і типом з різними вузловими точками.

5. Створіть на побудованому вище рисунку титульний напис та підписи осей.

6. Побудуйте криві пункту 4 на чотирьох різних рисунках.

7.7. Лабораторна робота 7. Графіка

1. Побудуйте залежності: $y = \sin(x), y = \sin(x) - 1, y = \sin(x) - 2$ різними кольорами та різною товщиною ліній.

2. Створіть графік $y = \sin(x)$, змініть його горизонтальні та вертикальні межі і нанесіть сітку.

3. Побудуйте графіки функцій: $y_1 = \sin x, y_2 = \cos 2x$ з нанесенням пояснюючого тексту на рисунку.

4. Розберіть функцію **plot3** та побудуйте параметрично задану криву $x = \cos^2(t); y = \sin^2(t); z = t^2$.

5. Розберіть функцію surf та побудуйте довільну поверхню.

6. З'ясуйте призначення функцій **tubeplot** та **view.**

7.8. Лабораторна робота 8. Розв'язання СЛАР

Створіть програми розв'язання СЛАР методами Крамера та Гаусса, використовуючи свій варіант рівнянь. Порівняйте результати, отримані двома методами.

Приклад виконання роботи. Розглянемо систему рівнянь:

$$x_1 + 2x_2 + 3x_3 + 4x_4 = 30;$$
$$-x_1 + 2x_2 - 3x_3 + 4x_4 = 10;$$
$$x_2 - x_3 + x_4 = 3;$$
$$x_1 + x_2 + x_3 + x_4 = 10.$$

```
% Знайдемо її розв'язок методом Крамера
A=[1 2 3 4; -1 2 -3 4; 0 1 -1 1; 1 1 1 1];
b=[30;10;3;10];
% Перевіримо невирожденість системи
rank(A)
% За правилом Крамера
A1 = A;
A2 = A;
A3 = A;
A4 = A;
A1(:,1) = b;
A2(:,2) = b;
A3(:,3) = b;
A4(:,4) = b;
x1 = det(A1)/det(A);
x2 = det(A2)/det(A);
x3 = det(A3)/det(A);
x4 = det(A4)/det(A);
x = [x1;x2;x3;x4]
x=
    1.0000
    2.0000
    3.0000
    4.0000
% Перевіримо розв'язок
A*x-b


ans=
```

```
    0
    0
    0
    0
```

```
% Розв’яжемо систему Ax=b методом Гаусса
% Для цього сформуємо розширену систему
A=[1 2 3 4; -1 2 -3 4; 0 1 -1 1; 1 1 1 1];
b=[30;10;3;10];
C=[A b];
% Приведемо її до ступінчатого вигляду, виконавши прямий
% та зворотний ходи методу Гаусса
D=rref(C)
D =
   1 0 0 0 1
   0 1 0 0 2
   0 0 1 0 3
   0 0 0 1 4
% Останній стовпець матриці є розв’язоком
x=D(:,5)
x =
   1
   2
   3
   4
% Перевірка розв’язку
A*x-b
ans =
   0
   0
   0
   0
```

ДОДАТКИ

Додаток А

Під час написання посібника було декілька помилок у командах FreeMat для графіки. Перша помилка – це положення тексту при використанні команди **xlabel**. Друга помилка полягає у використанні команди **get(gca,'position')**. Ця команда відмінно працює, коли вводиться вручну у вікні командного рядка. Однак, якщо ви намагаєтесь використовувати її в будь-якій програмі або функції, вона повертає неправильні дані.

Поки помилки виправлять, ось функція, яку можна зберегти у вигляді файлу **labelSet.m** десь на вашому шляху. Ви можете використовувати її кожен раз, коли ви створюєте графік, який має назву та підпис осей.

```
function labelSet
xaxisMarks=24;
labels.
yaxisMarks=40;
labels.
marginX=20;
marginY=20;
curHand=gca;
curSize=get(gcf,'figsize');
curTitle=get(curHand,'title');
if(isempty(curTitle));
titleFontsize=0;
else
titleFontsize=1.5*get(curTitle,'fontsize');
set(curTitle,'verticalalignment','middle');
set(curTitle,'horizontalalignment','center');
if(titleFontsize<20)
titleFontsize=20;
end
end
curXlabel=get(curHand,'xlabel');
if(isempty(curXlabel));
xFontsize=0;
else
xFontsize=2.5*get(curXlabel,'fontsize');
```

```
set(curXlabel,'horizontalalignment','center');
set(curXlabel,'verticalalignment','bottom');
if(xFontsize<20);
xFontsize=20;
end
end
curYlabel=get(curHand,'ylabel');
if(isempty(curYlabel));
yFontsize=0;
else
yFontsize=3*get(curYlabel,'fontsize')
set(curYlabel,'horizontalalignment','center');
set(curYlabel,'verticalalignment','top');
if(yFontsize<20);
yFontsize=20;
end
end
leftSide=(yFontsize+yaxisMarks)/curSize(1)
rightSide=marginX/curSize(1);
plotWidth=1-leftSide-rightSide;
bottomSide=(xaxisMarks+xFontsize)/curSize(2);
topSide=(marginY+titleFontsize)/curSize(2);
plotHeight=1-bottomSide-topSide;
set(curHand,'position',[leftSide bottomSide plotWidth plotHeight]);
if(titleFontsize>0);
titleTop=1-topSide/2;
set(curTitle,'position',[0.5 titleTop]);
end
```

Додаток Б

У FreeMat існує проблема з функцією **hist**. Вона не працює правильно. Поки це не виправлено, ви можете використовувати нижче наведену функцію, зберігши її як **hist.m** в одній із папок вашого шляху.

```
function [rv r]=hist(input_array,bins,normal,partial)
if(nargin>4);
printf('ERROR: The hist function requires no more than 4
arguments.\n');
return
end
if(nargin<4)
partial='full';
end
if(nargin==1);
bins=10;
normal=0;
max_amp=max(input_array);
min_amp=min(input_array);
bin_val=linspace(min_amp,max_amp,bins+1);
w=bin_val(2)-bin_val(1);
r=min_amp+w/2:w:max_amp-w/2;
elseif(nargin==2);
normal=0;
s=size(bins);
if(s(1)>s(2))
bins=bins';
end
if(sum(diff(diff(s)))>0)
printf('The bin array must be equally spaced.\n');
return
end
if(isscalar(bins))
max_amp=max(input_array);
min_amp=min(input_array);
bin_val=linspace(min_amp,max_amp,bins+1);
w=(max_amp-min_amp)/bins;
r=(min_amp+w/2):w:(max_amp-w/2);
else
r=bins;
w=bins(2)-bins(1);
max_amp=max(bins)+w/2;
```

```
min_amp=min(bins)-w/2;
bins=length(bins);
bin_val=linspace(min_amp,max_amp,bins+1);
end
elseif(nargin>=3);
s=size(bins);
if(s(1)>s(2))
bins=bins';
end
if(sum(diff(diff(s)))>0)
printf('The bin array must be equally spaced.\n');
return
end
if(isscalar(bins))
max_amp=max(input_array);
min_amp=min(input_array);
bin_val=linspace(min_amp,max_amp,bins+1);
w=(max_amp-min_amp)/bins;
r=(min_amp+w/2):w:(max_amp-w/2);
else
r=bins;
w=bins(2)-bins(1);
max_amp=max(bins)+w/2;
min_amp=min(bins)-w/2;
bins=length(bins);
bin_val=linspace(min_amp,max_amp,bins+1);
end
end
for(k=1:bins)
t=(input_array>bin_val(k)).*(input_array<(bin_val(k+1)));
cdf_array(k)=sum(t);
end
if(strcmp('full',partial))
t=input_array<bin_val(1);
cdf_array(1)=cdf_array(1)+sum(t);
t=input_array>bin_val(bins+1);
cdf_array(bins)=cdf_array(bins)+sum(t);
end
if(normal>0);
rv=cdf_array*normal/length(input_array);
else
rv=cdf_array;
end
```

СПИСОК ЛІТЕРАТУРИ

1. Дьяконов В. П. Справочник по расчётам на микрокалькуляторах. Издание 3-е, дополненное и переработанное / В. П. Дьяконов. – М. : Наука, Физматлит, 1989. – 464 с.
2. Дьяконов В. П. Справочник по алгоритмам и программам на языке Бейсик для персональных ЭВМ / В. П. Дьяконов. – М. : Наука, Физматлит, 1987. – 240 с.
3. Дьяконов В. П. Язык программирования ЛОГО / В. П. Дьяконов. – М. : Радио и связь, 1991. – 145 с.
4. Дьяконов В. П. Форт системы программирования персональных ЭВМ / В. П. Дьяконов. – М. : Наука, Физматлит, 1992. – 352 с.
5. Дьяконов В. П. Mercury – отличная система для всех / В. П. Дьяконов. – М. : Монитор-Аспект, 1995. – 86 с.
6. Дьяконов В. П. Справочник по применению системы Eureka / В. П. Дьяконов. – М. : Наука, Физматлит, 1993. – 96 с.
7. Дьяконов В. П. Система MathCAD. Справочник / В. П. Дьяконов. – М. : Радио и связь, 1993. – 128 с.
8. Дьяконов В. П. Справочник по применению системы PC MatLAB / В. П. Дьяконов. – М. : Наука, Физматлит, 1993. – 112 с.
9. Дьяконов В. П. Энциклопедия Mathcad 2001i/11 / В. П. Дьяконов. – М. : СОЛОН-Пресс, 2004. – 832 с.
10. Дьяконов В. П. Mathematica 4 с пакетами расширений / В. П. Дьяконов. – М. : Нолидж, 2000. – 608 с.
11. Дьяконов В. П. Mathematica 4.1/4.2/5 в математических и научно-технических расчётах / В. П. Дьяконов. – М. : СОЛОН-Пресс, 2004. – 696 с.
12. Дьяконов В. П. Maple 8 в математике, физике и образовании / В. П. Дьяконов. – М. : СОЛОН-Пресс, 2003. – 656 с.
13. Дьяконов В. П. Maple 9.5/10 в математике, физике и образовании / В. П. Дьяконов. – М. : СОЛОН-Пресс, 2006. – 720 с.
14. Дьяконов В. П. MATLAB 5 – система символьной математики / В. П. Дьяконов. – М. : «Нолидж», 1999. – 640 с.

15. Дьяконов В. П. MATLAB 6/6.1/6.5 + Simulink 4/5 в математике и моделировании. Основы применения. Полное руководство пользователя / В. П. Дьяконов. – М. : «СОЛОН-Пресс», 2003. – 520 с.
16. Дьяконов В. П. MATLAB 6.5/7.0 + Simulink 5/6 в математике и моделировании. Библиотека профессионала / В. П. Дьяконов. – М. : «СОЛОН-Пресс», 2005. – 576 с.
17. Иглин С. П. Математические расчёты на базе MATLAB / С. П. Иглин – СПб. : «БХВ-Питербург», 2005. – 640 с.
18. Дьяконов В. П. MATLAB 7.*/R2006/2007. Самоучитель / В. П. Дьяконов – М. : «ДМК-Пресс», 2008. – 768 с.
19. Алексеев Е. Р. MATLAB 7. Самоучитель / Е. Р. Алексеев, О. В Чеснокова. – М. : «СОЛОН-Пресс», 2005. – 464 с.
20. Курбатова Е. А. MATLAB 7. Самоучитель / Е. А. Курбатова. – М.: «Диалектика», 2005. – 256 с.
21. Іглін С. П. Варіаційне числення на базі Octave / С. П. Іглін. – Х. : НТУ "ХПІ", 2011. – 340 с.
22. Samit Basu FreeMat v4.0 Documentation. – 556 p. [Электронный ресурс]. Режим доступа: http://freemat.sourceforge.net/download.html
23. Gary Schafer, Timothy Cyders The Freemat Primer. First Edition. 2011. – 218 p. [Электронный ресурс]. Режим доступа: http://sourceforge.net/projects/freemat/files/Freemat%20Primer/V4e1/FreematPrimerV4e1.pdf/download?use_mirror=garr

ЗМІСТ

Навчальне видання

ОЛЬШАНСЬКИЙ Станіслав Васильович
АСЮТІН Олексій Дмитрович

Програмне забезпечення обчислювальних систем на базі FreeMat
Навчально-методичний посібник
для студентів спеціальностей “Прикладна математика” та “Інформатика” з курсів “Програмне забезпечення обчислювальних систем” та “Математичне моделювання”

Українською мовою

Роботу до друку рекомендував С. К. Шелковий

Редактор Н.В. Верстюк

План 2012 р., поз. 87/

Підп. до друку . . . Формат 60х84 1/16. Папір друк. №2.
Riso-друк. Гарнітура Таймс. Ум. друк. арк. .
Наклад 50 прим. Зам. № . Ціна договірна.

Видавничий центр НТУ “ХПІ”. Свідоцтво ДК № 3657 від 24.12.2009 р.
61002, Харків, вул. Фрунзе, 21

Друкарня НТУ “ХПІ”
61002, Харків, вул. Фрунзе, 21