МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний авіаційний університет

Азарсков В.М., Гаєв Є.О.

Сучасне програмування

Навчальний посібник

Частина1

"Програмування та математика – із другом MATLABом"

Київ 2014

МІНИСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний авіаційний університет

Азарсков В.М., Гаєв Є.О.

Сучасне програмування

Навчальний посібник з дисциплін
„Програмування”, „Алгоритмічні мови та програмування”

Частина 1: модулі 1, 2

"Програмування та математика – із другом MATLABом"

Київ 2014

УДК 004.94 (075.8)
ББК З 973.2-018.1я7
А 351

Рецензенти: д.т.н. проф..Бідюк П.І. – Інститут прикл. системного аналізу НАНУ і МОНУ;
к.т.н. доцент Нестеренко Б.М. – кафедра КСУ НАУ.

Затверджено на засіданні кафедри систем управління літальних апаратів Національного Авіаційного Університету *7 липня* 2014 року.

Азарсков В.М., Гаєв Є.О.

А 351 ***Сучасне програмування.*** Навчальний посібник з дисциплін „Програмування”, „Алгоритмічні мови та програмування”. Модулі 1,2 "Програмування та математика – із другом MATLABом" – К.: ТОВ «НВП «Інтерсервіс», 2014. – 254 с.

ISBN 978-617-696-284-7

Вступний курс з програмування (два модулі з шести) для студентів першого року навчання використовує математичне і програмне середовище MATLAB. Оволодіваючи цим важливим пакетом, студенти знайомляться із структурним програмуванням, з багатьма типами даних, комп’ютерною анімацією, рекурсією, GUI-програмами, елементами об’єктно-орієнтованого програмування тощо. Теоретичні питання науки програмування та алгоритмізації реалізуються у простих і привабливих MATLAB-програмах. Потрібний теоретичний апарат спирається лише на матеріал першого курсу з математики. Студент одночасно отримує інструмент для останньої, а також для багатьох наступних університетських дисциплін. Окремий розділ присвячено можливим власним „експериментам” у MATLAB. Контрольні питання і лабораторні завдання закріплюють практичні навички студентів.

Для студентів молодших курсів, які навчаються за спеціальністю „Системна інженерія” та іншими спеціальностями інформаційних технологій.

УДК 004.94 (075.8)
ББК З 973.2-018.1я7

ISBN 978-617-696-284-7

З М І С Т коротко

З М І С Т детальний

* Позначкою помічаємо розділи, що можна опустити при першому читанні

План наступної частини **(модулі 3 і 4, 5 і 6)**

5. Реалізація складних алгоритмів
6. До розумних комп'ютерів, графічні інтерфейси
7. Об'єктно-орієнтоване програмування в MATLAB
8. Професія – програміст
9. Мова програмування Java
10. Java у MATLAB

Для кого ця книжка?

Ця книжка для тих,

- ✓ хто хоче оволодіти основами програмування,
- ✓ кому цікаво „вести діалог” з власною розумною програмою з математики, фізики, інших дисциплін, зробленою для курсових або дипломних робіт, або

„огорнути” власну програми у графічну оболонку (*графічний інтерфейс користувача,* GUI).

Ця книжка для тих,

- ✓ хто хоче краще засвоїти математику, науку над усіма іншими науками, однак – найбільш важку,
- ✓ хто хоче отримати надійного провідника і друга у цій науці. Звати його – MATLAB.

Ця книжка для тих,

- ✓ хто хоче програмувати легше та швидше, аніж із звичайними мовами програмування.

Ця книжка найбільш підходить студентам першого року навчання

- ✓ бо проста та зрозуміла,
- ✓ бо полегшить Вашу роботу на першому курсі і наступних,
- ✓ бо навчить Вас MATLABу, що прийнятий Міністерством освіти України основним математичним пакетом вітчизняних університетів.

Так само, ця книжка – для студентів старших років навчання

- ✓ бо зробить MATLAB помічником у курсових роботах, у дипломному проекті, у наступній практичній роботі;

Ця книжка також для інженерів та науковців

- ✓ що бажають з легкістю вивчити та використовувати MATLAB – сучасну, майже всеохоплюючу математичну систему.

Отже, знайомтеся: MATLAB – Ваш друг і помічник у математиці і програмуванні!

Передмова для викладачів

Студенти приходять до першого курсу підготовлені, у більшості, недостатньо. Про це сьогодні лише байдужий не каже...

Натомість оснащені вони сьогодні непогано: майже кожен другий приносить на заняття ноутбук. І то є ключова риса сучасних студентів, „дітей Google" та Інтернету – всі вони фанати комп'ютера!

Одначе, як використовують вони свої коштовні ноутбуки? Для „прогулянок" Інтернетом, для комп'ютерних ігор, для скачування фільмів та музики... Навчити їх використовувати неосяжні потужності цифрової ери „за призначенням" – це є вихідне завдання їхнього першого ВУЗівського курсу програмування та алгоритмічних мов. Кількарічний досвід авторів та їх пошуки [1-7] викладено у цій книжці.

Чи повністю задовольняють такій вимозі випробувані часом задачі курсу програмування „*Hello World*", розв'язок квадратного рівняння у залежності від його дискримінанту D, обчислення функції за її рядом Тейлора тощо? Сьогоднішній молоді, необтяженій досвідом, це здається нудним...

Тому й спробували автори чотири модулі з шести курсу „Програмування" та „Програмування та алгоритмічні мови"[1] присвятити математично-програмному середовищу MATLAB. Саме воно дозволяє пояснити всі концепції (ну, погодимося з критиками – більшість) комп'ютерних наук, суттєво скоротивши шлях від теорії до красивої, захоплюючої програми з „розумним" діалогом або навіть з GUI. Програма *Clock*, що зображена на рис. 0.1, потребувала від автора-студента лише пару годин. А скільки потрібно було б у Delphi або C^{++}? Результати авторського „викладацького експерименту", здається, позитивні. Вам судити!

[1] Два інших модуля, як зазвичай в українських університетах, присвячено або Pascal, або C. Їх ведуть інші викладачі.

У числі авторських задач пропонуються:

- Програма *Welcome* (стор. 170,232) для оволодіння умовним оператором IF…END, яка „каже” студентові „Добрий ранок” і виконує музику, якщо програму запущено до 12:00; „Добрий день” між 12:01 та 18:00, і таке подібне;

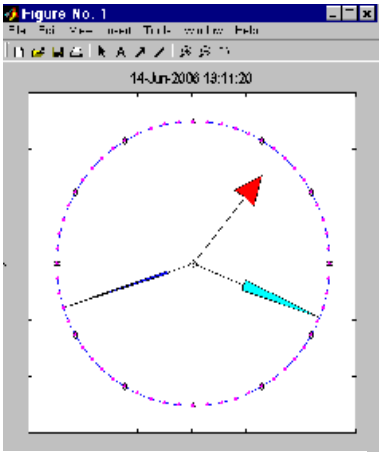


Рис.0.1. Програма ***Clock***, створена студентом.

- Програма *Helicopter* для ***анімації*** обертання „гвинта” на екрані комп’ютера задля оволодіння студентом оператором циклу FOR…END, стор. 234;
- „Експериментальне” встановлення ефективності (потрібного часу) власноручної програми шляхом виміру часу обробки масивів в залежності від їх розміру n, побудови графіків $Час = f(n)$ у порівнянні з внутрішньою програмою MATLAB і встановлення ***порядку зростання*** $O(n^{\alpha})$ за допомогою інструменту *Basic Fitting* (розділ 2.3);

У посібнику розглянуто також

- ✓ Інші стандартні оператори структурного програмування;
- ✓ Деякі ключові задачі обробки масивів (визначення його найменшого\найбільшого елементу, упорядкування елементів „алгоритмом бульбашки” у порядку збільшення або зменшення) [8,9];
- ✓ Поняття ітерації та рекурсивної програми (розділ 4.3) та пов’язані з цим приклади простого фракталу та функції Дебєчі (у другій частині підручника);
- ✓ Важливі типи даних, що використовуються у комп’ютерних науках (числові, текстові, масиви,

символьні дані згідно до Symbolic Math Toolbox, комірки, структури, вказівники (handles), об'єкти та команди визначення і встановлення їхніх властивостей *get* і *set*);

- ✓ поняття про наслідування ознак та інші ідеї об'єктно орієнтованого програмування (у другій частині);
- ✓ Статичні та динамічні масиви і відповідне їх розміщення у пам'яті комп'ютера;
- ✓ „Події" у програмуванні";
- ✓ Соціальний контекст інформатики: історія комп'ютера і інформатики, інтелектуальна власність на ПЗ, основи взаємодії людини і машини, специфікації й вимоги до ПЗ тощо.

Так, все це можливо в MATLAB, особливо у сучасних версіях v.2010 (тобто v.8 вітчизняних „піратів") – v.2014. Можна сказати, що MATLAB може виступати таким же навчальним середовищем, яким був свого часу Basic. Зараз в MATLAB можливо навіть *exe*-файли компілювати [52,53]. Така еволюція цього середовища відповідає, ймовірно, загальній тенденції до синтезу й уніфікації, прикладом чого є мови Java, C#, програмна технологія .NET Framework тощо...

Перелік питань, що ми зачіпаємо, відповідає Програмі дисципліни, що затверджена Міністерством Освіти і Науки України [11]. У той же час, враховуємо більш загальний документ *Computing Curricula 2012* [44], що створений групою світових експертів комп'ютерної науки. Зрозуміло, наш підхід не заперечує вивчення інших мов програмування (Мова Java й вивчається у двох останніх модулях курсу!); автори прагнуть лише показати студентові горизонти цієї науки якнайшвидше. Використання MATLAB задля цього відповідає й закордонним тенденціям [16,20,42,43].

„*Менше голої теорії, більше привабливих задач*" – лозунг даного посібника. Джерелом таких задач є виключно матеріал, що студентами принесено зі школи або вивчається на першому році навчання вищої математики. З іншого боку, все, що студент узнає з цих модулів, може бути відразу з користю застосовано в оволодінні тією ж математикою. MATLAB стане другом студента в оволодінні ним відразу

програмуванням, математикою та іншими дисциплінами. Хіба не на це спрямовано згадане вище Рішення Міністерства Освіти [11]?

Однак автори категорично не погоджуються з методикою багатьох книжок з MATLAB. Вони пропонують: „*Для такої задачі натисніть таку кнопку, виконайте таку-то команду*” – це не дозволяє студентові зрозуміти математичний зміст задачі, оволодіти саме математичним методом, і тому зустрічає виправданий опір[2] „класичних викладачів” математики. Замість цього „методу мавпи” авторський метод полягає в

- використанні MATLAB спершу, на стадії початкового оволодіння у Розділі 2, як розвиненого калькулятора, і
- дослідницького характеру більшості запропонованих задач, чисельному експериментуванні з MATLAB.

Пояснимо нашу методику.

1.1. Вже в Лабораторній роботі 1 студентам пропонується "дослідити графічно", що певні частинні суми функцій наближаються до якоїсь іншої. Не треба студентам знати про ряди Тейлора або Фур’є; однак вони легко будують командою *ezplot*[3] кілька графіків частинних сум в одному вікні з граничною функцією і бачать таке наближення на власні очі! При цьому, для початку, нічого ще не треба знати з програмування: просто повторювати попередню команду, копіювати і вставляти „спільний член” з попереднього тексту та вручну редагувати його.

1.2. Вже з першими навичками програмування студент повертається до цієї ж задачі, Лабораторна робота 5. Тут йому пропонується вже запрограмувати той алгоритм, що раніше виконувався вручну. Крім того, він вже може створити і використати програму розривної функції, потрібної для прикладів з рядами Фур’є. Крім того, він вже знає команду *tailor*, що „аналітично” видає запрошену кількість доданків вказаного ряду.

1.3. В наступному розділі знов повертаємось до цих задач задля створення програм з елементами штучного інтелекту – діалогових або з графічним інтерфейсом (у

[2] А вже ті, хто математикою володіє, на етапі курсової або дипломної роботи з будь-якої дисципліни – будь ласка, „тисніть кнопки”, вже розуміючи, що за ними стоїть...

[3] Назва-то команди походить від *easy-to-plot*, *легко будувати*...

другій частині підручника). На кожному етапі студентові пропонується „погратися” шляхом вибору іншої функції та зробити (математичні!) висновки. Вчити їх строгому доведенню фактів про ряди Тейлора і Фур’є буде викладач-математик на другому році навчання, а тепер – лише наочна ілюстрація до цих фактів!

Аналогічно, автори проти „бездумного” використання першочками MATLAB-команди *limit*. Замість цього пропонується

2.1. дослідити, наприклад, функцію $\frac{\sin x}{x}$ поблизу 0 (Лабораторна 1), або

2.2. послідовність $\left(1+\frac{1}{n}\right)^n$ за умов необмеженого зростання цілого n, та знайти графічно граничне значення знаменитого числа *e* (Розділ 4).

Згідно з сучасними закордонними тенденціями викладання математики з застосуванням MATLAB та з проведенням чисельних MATLAB-досліджень [16,20], автори включили Розділ 4 дослідницького характеру. З урахуванням обмежених ще знань першокурсників, тут пропонуються прості „ігри” (а „по дорослому” – дослідження) з механіки та теорії ймовірностей, а також „суто програмістські” проблеми з рекурсивних програм та вимірювання „ефективності” власних програм.

Таким чином, математичні задачі і математичні можливості MATLAB слугують для навчання програмуванню, і одночасно – програмування допомагає студентові (не замінюючи фахового викладача) оволодіти математикою та полюбити цю складну науку. Автори сподіваються, що цією Передмовою для викладачів вони задовільно відповіли на ті пересторогі з боку колег, як з інформатики, так і математиків, що виникали на різноманітних попередніх обговореннях.

В С Т У П

Сьогоднішня доба в інформатиці така, що швидкісні і потужні ЕОМ (переважно архітектури IBM) стають дедалі доступнішими, багато хто з студентів вже зараз має їх у персональному користуванні (що не можна, на жаль, сказати про їхніх вчителів-:). А з точки зору програмного забезпечення – більшість з майбутніх фахівців вже не будуть працювати з якоюсь окремою мовою програмування *Fortran*, *Pascal* або *C*$^{++}$. На зміну останнім приходять інтегровані програмні продукти (“пакети”) з так званим “дружнім інтерфейсом”. Дизайнерська ідеологія розраховувати машину (праску, холодильник, пральну машину тощо), як кажуть, “*на чайника*” торкнулася вже й галузі “розумних” машин: згадані програмні продукти дозволяють розв’язувати математичні задачі навіть людям, які в цих задачах не розуміються… Так би мовити “Не треба думати, програма підкаже!”

Відомо кілька таких інтегрованих пакетів – це MathCAD, Maple, MATLAB та інші. MATLAB є одним з найвідоміших і найпоширеніших з них. В багатьох американських університетах, наприклад, вивчення MATLAB вже більше десяти років є обов‘язковим для студентів першого – другого курсів. Університети України, інших країн СНД також почали включати інтегровані математичні пакети до навчальних програм, про що свідчить перелік щойно надрукованих посібників [1-10,12-14,16-20]. Міністерство освіти і науки України оголосило MATLAB основним програмним засобом для ВУЗів країни [11], та поки що (середина 2014 р.) мало зробило реального! Кількість літератури з MATLAB також з кожним роком зростає.

Даний підручник відрізняється від названих книжок орієнтацією саме на програми вищої школи з інформаційних технологій та математичного моделювання систем і процесів. Він базується на наших книжках [1–3] і подальшому досвіді викладання курсу *"Програмування на алгоритмічних мовах"* під кутом зору MATLAB та курсу *"Вища математика"* з MATLAB.

Комп'ютери обіцяють зробити революцію у технологіях навчання. “Нудні” математичні теми можна зробити цікавими, якщо пов′язати їх з комп′ютерними захопленнями сучасної молоді. Кращі і найтворчі викладачі України шукають шляхи „впровадити” комп’ютер у свою дисципліну. Поява нових книжок [12,13,19,21,44,47] та інших це підтверджує. Універсальний математичний пакет MATLAB такій вимозі відповідає у найбільшій мірі.

Оволодіти MATLAB без допомоги викладача доволі складно. Даний підручник починає з початкового рівня (розділ 1) і доводить студента до достатньо складних задач курсу математики. Для швидкого навчання використано не формально-академічний підхід (алфавіт, операції, синтаксис і т.д.), а численні приклади і метод "*роби як я*", що спирається на інтуїцію студента.

Ознайомившись з даним підручником ви зможете самі програмувати доволі цікаві задачі, такі як годинник зі стрілками, що рухаються (рис. 0.1), синхронізований до годинника на Вашому комп’ютері; трикутник або багатокутник, що обертаються; програми тестування знань з аналізу функцій та аналітичної геометрії; побудову фракталу (у наступній частині). Окрема програма *Welcome* привітає Вас в залежності від часу доби. Автори згодні із львівським викладачем І.О. Хвищуном [46], що проголосив справжнє захоплення програмуванням у задачах фізики. Сподіваємось, що таке захоплення і насолоду відчуєте і Ви!

Дехто ще досі помилково вважає (мабуть, на підставі знайомства ще з версіями до 5.*), що MATLAB – це лише математика. Ні, сьогодні це вже не так. MATLAB сьогодні – це ВСЕ. Це все, чого торкається сучасна наука і інформатика. Припустимо, музика і звук, образотворче мистецтво і зображення. Хоча тут немає місця торкатися таких тем, та легко показати, що і це присутнє у MATLAB. Нутко, виконайте у командному рядочку таку команду

>> load handel; sound(y)

Подобається Вам ораторія Генделя? Спробуйте також трохи побавитися частотою звука,

>> load handel; sound(y, 6000)

або

>> load handel; sound(y, 9000)

Тепер побавтеся художніми можливостями MATLAB[*)]:

>> I = imread('peppers.png'); imshow(I);

або

>> imageext

Подобається? Можна очікувати, що усе це – не лише для розваг.

Уявлення про перелік тем даного підручника дає ***Зміст***. Матеріалом для роботи з MATLAB обрані деякі типові задачі математичного моделювання, які часто зустрічаються в математичному моделюванні систем і процесів і тому включені до навчальних програм з багатьох дисциплін, що вивчають у вищій школі. Для оволодіння таким програмним матеріалом вважаємо за доцільне їх "ручне" розв'язування; цьому присвячено розділ 2. Це дозволяє засвоїти основи того чи іншого математичного методу, зосередитись на умовах правильної, коректної постановки задачі, а також на алгоритмічній стороні обчислень, не відволікаючись на програмування. На даному етапі студенти використовують середовище MATLAB як швидкий калькулятор, як свого часу в курсах вищої математики використовували "кишенькові" програмовані мікро-ЕОМ *Б3-34*, *МК-52*, *МК-54* [40]. У викладенні основ того чи іншого математичного методу ми спираємось на його наочне представлення, нагадуємо лише головні ідеї і властивості, які завжди потрібно пам'ятати студенту, інженеру і досліднику з посиланням на посібники, де можна знайти подробиці і обґрунтування.

У розділі 3 студенти повертаються до тих самих задач, але вже з метою їхнього програмування найпростішими засобами MATLAB. Слід розуміти, що до версії v.5 MATLAB розвивався як інструмент для, в основному, математики. Однак зараз він "виріс" до версій v.6 і v.7 і вище, включивши

у себе не лише класичні і новітні (такі як *вейвлети*) математично орієнтовані науки ("тулбокси" обробки зображень, звуку, тощо), і перетворився на середовище розробки прикладних комп'ютерних програм. Алгоритмічна мова MATLAB є навіть "трохи вищого рівня", аніж алгоритмічні мови Fortran, Pascal, C, Java. Дійсно, наш студент легко зробив програму годинник (рис. 0.1), яка на інших мовах потребувала б до кількох днів роботи! Для всіх прикладних програм MATLAB дозволяє легко запрограмувати графічну оболонку (GUI), аби вона виглядала красиво, сучасно, у відповідності до стандартів Windows.

Даний посібник включає як теоретичний матеріал, так і лабораторні роботи, що у попередній редакції видані в [3]. Використання останніх дозволяє комбінувати різні методики навчання. Можна йти традиційним шляхом, від теорії (даний посібник або [1]) до вправ у лабораторних роботах. Але у багатьох випадках вивчення теми можна починати саме з лабораторної роботи, мотивуючи студентів самостійно знайти відповіді на поставлені питання. У такий спосіб слід спиратись на евристичний (дружній) інтерфейс MATLAB і на його розвинену довідкову систему (останнє ще дає додаткову практику в англійській мові…). Після цього теоретична частина – це перевірка Вашої роботи, відповіді на запитання, які залишилися.

Цей підручник дозволяє оволодіти MATLAB у такому степені, що користувач зможе самостійно робити складні, розгалужені обчислення для власної навчальної або навіть професійної роботи. У той же час слід зауважити, що підручник може вмістити лише **основи** роботи в середовищі MATLAB, і далеко не в повній мірі вичерпує можливості останнього. Тут ми обмежилися так би мовити *одновимірною* математикою. Подальшому викладенню MATLAB, його застосуванню до двовимірних математичних задач, плануємо присвятити наступні роботи.

Підручник створено на підставі значного досвіду викладання дисциплін „Вища математика” і „Алгоритмічні мови та програмування” (останній курс – для студентів першого року навчання) у Національному авіаційному

університеті. Автори радіють щирому захопленню студентів від саме такого викладання, що поєднує математику з програмуванням та навпаки. Результат – цей курс, що можна назвати авторським.

Висловлюємо щиру вдячність фірмі MathWorks за гранти у вигляді постійно поновлюваних версій MATLAB, за промоцію попередніх книжок на їхньому сайті http://www.mathworks.com/support/books/ та організацію Web-семінарів і консультацій через MATLAB Central Newsreader. Щира подяка колишній дипломниці Н. Шубіної – наполегливій „перекладачці" на *html*–мову, що створила мультимедійні версії книг [1,2], та багатьом студентам, які з захопленням розробляють курсові та дипломні роботи на підставі даного курсу.

Автори

Модуль 1: підготовчий

1. АЗБУКА МАТЕМАТИЧНОГО СЕРЕДОВИЩА MATLAB

Назва комп'ютерної програми MATLAB походить зовсім не від слова *математика*, а від *MATrix LABoratory*, "Матрична лабораторія", хоча це дійсно є розвинена і універсальна система для наукових і інженерних розрахунків. Тобто, в основі її лежить *спосіб представлення* ***даних***.

MATLAB популярний у багатьох університетах і науково-виробничих фірмах світу. Система "розуміє" програми на мовах Fortran, C, C^{++} та Java, пов'язана з відомим текстовим редактором Microsoft Word і з табличним процесором Excel, що робить її зручною для виконання складних розрахунків (зокрема – розрахунків навчального характеру) та оформлення звітів про них, курсових та дипломних робіт. За досить тривалий час її розповсюдження (з 1984 р.) пакет MATLAB здобув собі стійку популярність. Ось чому обрано саме цю програму з кількох відомих (і певною мірою еквівалентних), таких як MathCAD[4], Maple, Mathematica та інших...

Існують декілька модифікацій (версій) системи MATLAB. Вже майже вийшли з обігу версії MATLAB-5.x, для яких видані посібники [6-13,17,18]. В даному посібнику обрано більш сучасні на сьогоднішній день версії MATLAB v.6.5 [5,14], наступні, а також MATLAB-2014a (нумерація версій в останній час змінилася, стала слідувати номеру року). Цей продукт постійно розвивається і збагачується все більшими можливостями.

У даному розділі 1 пропонуємо оволодіти основами MATLAB крок за кроком шляхом певних вправ, від зовсім простих до складних за методом "*роби як я*". Наприкінці розділу студент придбає досить широкий круг навичок роботи в MATLAB, після чого можна перейти до

[4] А от назва цієї програми включає в себе слово mat**h**ematics – на відміну від MATLAB!

розв′язування навчальних задач курсу вищої і прикладної математики (розділ 2), або до основ програмування (розділ 3) та подальшого оволодіння цією наукою (і одночасно – мистецтвом), наступні розділи. Проте, вже після розділу 2 студент є достатньо „озброєний", аби розпочати (уявіть собі!) **власні дослідження**. Кілька тем з прикладами наведено в розділі 4. Запропоновано як прості дослідження (наочне уявлення, що таке *граница*), так і доволі складні з рядів Тейлора і Фур'є, рекурсивних функцій і програм, поняття фракталу, з механіки й теорії ймовірностей. Ніяких попередніх знань з математики при цьому не потрібно!

Тенденція залучення студентів до творчих й дослідницьких задач притаманна сучасним західним підручникам [20] та кращим вітчизняним [19,46,47]. До такого викладання науки про програмування автори прийшли на підставі багатьох років викладання цієї дисципліни та кількох попередніх книжок [1-3], з урахуванням вимог, що виставляє міжнародне комп'ютерне співтовариство [44].

1.1. ІНТЕРФЕЙС ПРОГРАМИ MATLAB

Як "театр починається з вішалки", так сучасна комп'ютерна програма починається і надає користувачеві всі свої можливості через **інтерфейс** – графічну підпрограму для спілкування користувача з програмою основною.

Студентам з певним комп'ютерним досвідом і з розвиненим почуттям "*Я сам*!" радимо розпочати зі самостійного виконання лабораторної роботи 1 даного посібника. Дружній інтерфейс програми MATLAB і її *Help* підкажуть відповіді на запитання, запропоновані наведеним планом. Саме так студенти часто роблять, розбираючись з новою програмою. Перевірити вашу відповідь та знайти розв'язання нез'ясованих питань зможете у наступному підрозділі. Виконання лабораторної роботи 1 може бути також задане викладачем.

1.2. КОРОТКІ ВІДОМОСТІ ПРО КОМП'ЮТЕРНИЙ ПАКЕТ MATLAB

Після запуску MATLAB, користувач бачить на екрані комп'ютера вікно, що виглядає приблизно так, як зображено на рис. 1.1. Саме воно разом із "кнопками меню" та "іконками" і утворює ***інтерфейс програми***. Його вигляд залежить від того, які опції обрано з меню програми

Viev \ Desktop Layout ‣

Варіанти того, як буде виглядати інтерфейс системи, є: *Default* (без втручання користувача), *Command Window only* (лише вікно команд), *Simple* (найпростіший), *Short History* (вікно з історією команд користувача, скорочений варіант), *Tall History* (історія команд, повний варіант) та *Five Pannels* (усі можливі п'ять панелей з інформацією про ресурси MATLAB). Вигляд програми залежить також від версії MATLAB. На рис. 1.2 представлено інтерфейс програми MATLAB-2012b; хоч виглядає він трохи інакше, та легко може бути наближений до першого вигляду.

Для наших цілей достатньо мати справу лише з *Command Window* – вікном команд, показаним на обох рисунках. На рис. 1.1 показано ще вікно "Історія команд" роботи користувача (*History Window*). Така конфігурація вікон відповідає опції *Simple* (найпростіша).

У вікні праворуч, яке називається командним, бачимо так зване "*запрошення*" – пару символів ">>", яка показує, куди треба вводити команди. На рис. 1.1 наведено кілька прикладів команд MATLAB-середовища. Приклад 1 і 2 – це обчислення складного математичного виразу. Інформацію про це написано латинськими буквами після знака "***%***". Якби знака "%" не було, після натискання клавіши *Ввод* (*Enter*) система "облаяла" б користувача:

"*Error: Missing operator, comma, or semicolon*" ("*Помилка: відсутній оператор, кома або кома з крапкою*").

Таким чином, знак "%" ставлять для того, аби система не помічала наступного тексту. Такий прийом використовують для написання ***коментарів*** до складної задачі.

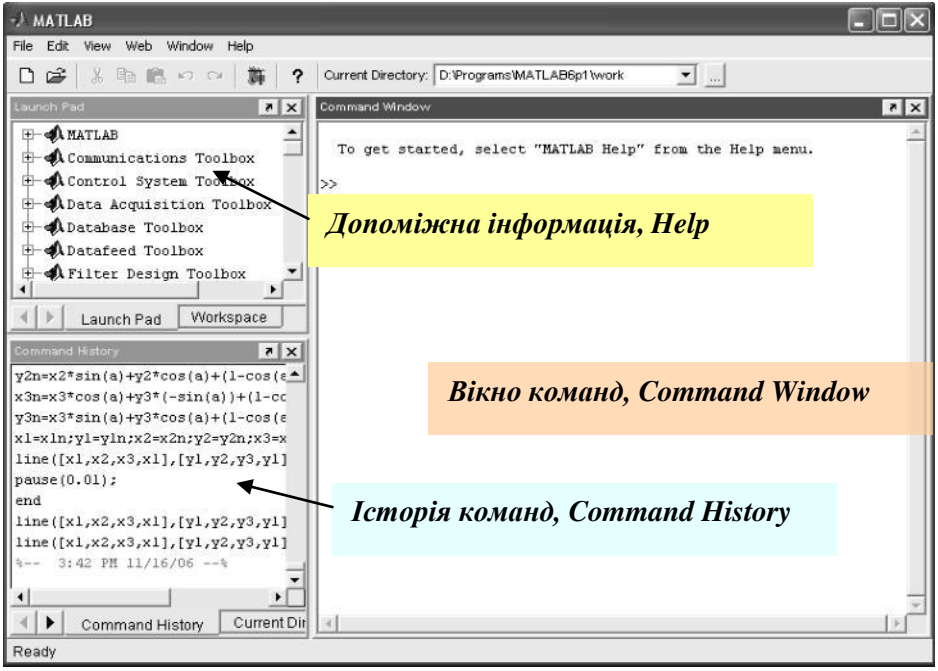


Рис. 1.1. Можливий вигляд програми MATLAB відразу після запуску: Праворуч – Вікно команд (розмір можна змінювати) – найважливіше для нас; Ліворуч –Вікна "Lanch Pad " (нам не потрібно, можна закрити) і „Command History” (також важливе для нас!).

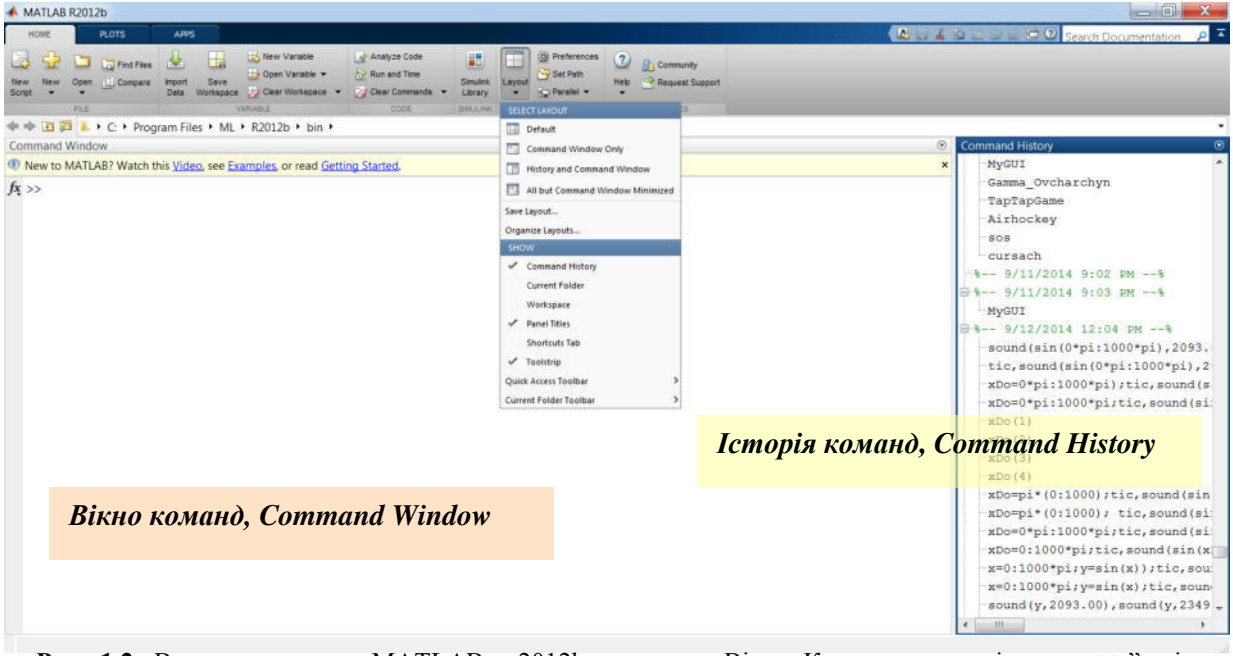


Рис. 1.2. Вигляд програми MATLAB v.2012b: так само Вікно Команд з „привітанням >>”; вікно „Command History” праворуч та розкрите „меню”, що дозволяє обрати інші вікна середовища MATLAB.

(Радимо писати коментарі латинськими, або великими українськими літерами – пояснення див. в розділі 3.4).

Приклади 1 – 5 (див. рис. 1.3) знайомлять із способами простих обчислень в системі MATLAB, а також із *алфавітом* і *синтаксисом* системи. Останні наближені до звичної нам форми математичних записів. Розглянемо кілька прикладів.

Приклад 1.1. Обчислити вираз

$$\frac{\sqrt{2+\sqrt{3}}}{\sqrt{2-\sqrt{3}}}+\frac{\sqrt{2-\sqrt{3}}}{\sqrt{2+\sqrt{3}}}.$$

У командному вікні після “>>” набираємо на спеціальній мові, яка наближена до звичайної, але зрозуміла програмі:

```
>>[5] sqrt(2+sqrt(3))/sqrt(2-sqrt(3))+sqrt(2-
      sqrt(3))/sqrt(2+sqrt(3))
```

Натискуємо *Ввод*, і система дає відповідь: *ans =4* (або *4.0000*, залежно від замовленої точності).*)

Можна зробити і інакше, вводячи кілька команд у рядочку:

```
>>  x=2; y=3; sqrt(x+sqrt(y))/sqrt(x-sqrt(y))+sqrt(x-
      sqrt(y))/sqrt(x+sqrt(y))
```

Або так:

```
>>  x=2;y=3;  Num[6]=sqrt(x+sqrt(y));  DeNum=sqrt(x-sqrt(y));
Sum= Num / DeNum
```

В обох випадках програма надає *x* значення *2* і *y* значення *3* і підставляє їх у наступні вирази. Натискуємо *Ввод* і система дає таку саме відповідь: *ans*[7] *= 4*, або *Sum = 4*.

---

[5] Ця позначка у тексті **>>** (два знака „більше”) тут і далі буде позначати, що написаний рядочок слід виконувати у Командному вікні.

[6] Імена для змінних *Num* і *DeNum* введено, виходячи з англійських *Nominator* і *Denomenator* (*чисельник* і *знаменник*).

[7] Змінну з іменем *ans* (що означає *answer*, *відповідь*) MATLAB створює автоматично, коли про найменування забув програміст.

Приклад 1.2. Обчислити

$$(3^{-1}(\frac{1}{2})^{-2} - 27^{-\frac{1}{3}})(3 \cdot \frac{2}{5} - 0.9).$$

У командному вікні набираємо:

>> *y=(3^ (-1)*(1/2)^ (-2) -27^ (-1/3))*(3*2/5 - .9).*

Після *Ввод* маємо відповідь: *y =0.3000.*

Увага: між двома множниками (виразами в дужках) ставити знак множення * обов′язково!

Приклад 1.3. Обчислити значення функції:

$$\sin(\frac{25\pi}{6}) - \cos(-\frac{\pi}{3}).$$

На мові MATLAB така формула виглядає як

>> *z=sin(25*pi/6) -- cos(-pi/3).*

Отримуємо результат *z = 4.4409e-016*. Практично, це *0*, але приклад 1.3 свідчить, з якою точністю MATLAB обчислював функції *sin* і *cos*.

У прикладах 1.2 і 1.3 отримано величини заданих змінних *y* і *z*, а у прикладі 1.1 система дала назву самостійно – *ans* (від *answer*, відповідь). У подальшому система вже знає, що величини *y*, *z* і *ans* мають певні значення, так що на введення *y^2+z^2* або *sqrt(ans)* система видасть значення відповідно *0.09* та *2*. Тобто, запис із знаком рівності " = " дозволяє надати змінній потрібне числове значення.

MATLAB виконав ці обчислення, оскільки відповідні завдання (*оператори*) записані без синтаксичних помилок. Наприклад, кількість відкриваючих дужок обов′язково дорівнює кількості закриваючих. Спробуйте зробити помилки навмисно – і система відмовиться обчислювати, відразу вкаже на помилку: *Error*.

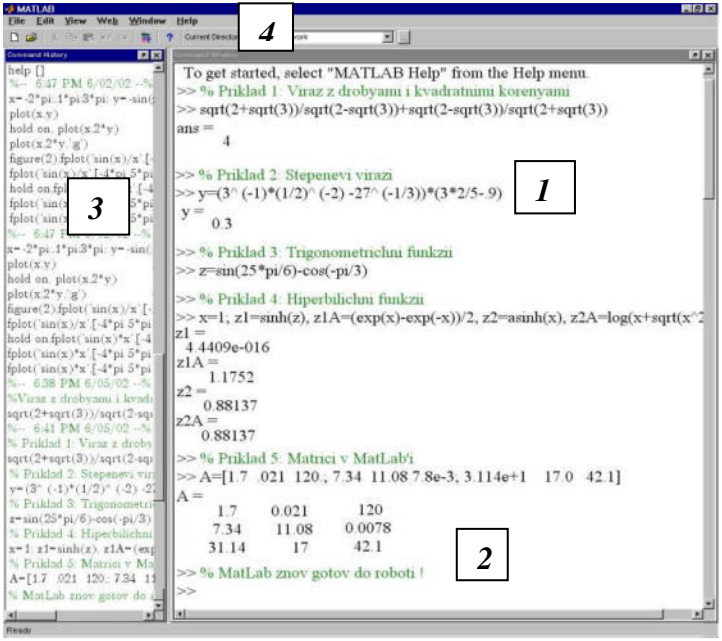


Рис. 1.3. Вигляд програми MATLAB у конфігурації Simple з прикладами певних обчислень: 1 – Вікно команд (розмір можна змінювати); 2 – "Запрошення" до вводу команд; 3 – Вікно "Історія" (розмір можна змінювати); 4 – Меню з багатьма можливостями

Цей перший досвід дозволяє заключити: в MATLAB імена змінних записуються буквами латинського алфавіту і можуть бути довільними. Лише певні імена, зокрема

ans, *pi*, *i*, *j*, *inf*, *NaN*,

зарезервовано для внутрішніх потреб MATLAB: *pi*=π≈3,141592 (число π), $i = j = \sqrt{-1}$ (уявна одиниця), *inf*=∞ (infinity, нескінченність) і невизначеності $NaN = \frac{0}{0}$ або $NaN = \frac{\infty}{\infty}$ ("*nane*", not a number). Користувач не має права надавати такі імена власним змінним.

Приклад 1.3 показує, що імена *sin*( ), *cos*( ), *exp*( ), *log*( ), *log10*( ) стають іменами функцій (відповідно – синуса, косинуса, експоненти $e^{()}$, логарифма натурального *ln* і логарифма десяткового *lg*), коли вони у дужках заключають імена змінних або математичні вирази. Імена інших функцій, відомих MATLAB, дещо відрізняються від звичних у вітчизняній літературі.

Таким чином, відомі математичні співвідношення

$$\operatorname{sh} x = \tfrac{1}{2}(e^{x} - e^{-x}) \qquad \text{та} \qquad \operatorname{arcsh} x = \ln(x + \sqrt{x^2 + 1})$$

на мові MATLAB будуть виглядати як

sinh(*x*)=(*exp*(*x*)-*exp*(-*x*))/2 та *asinh*(*x*)=*log*(*x*+*sqrt*(*x*^2+*1*)).

Приклад 1.4 на рис. 1.3 наводить обчислення цих функцій для *x*=1.

Увага: в MATLAB введено кілька власних математичних операцій – ділення "справа наліво" **** і поелементні множення **.*** і ділення **./**. Для чисел ділення *a/b* означає, як звичайно, $\frac{a}{b}$; а "*ділення справа наліво*" a\b означає $\frac{b}{a}$. Проте для матриць різниця між двома операціями більш суттєва. Поелементні операції (операції з крапкою) **.*** і **./,** а також **.^** застосовуються лише до векторів і матриць однакового виміру – див. про них в розділах 1.3 і 1.5.

Наступне правило MATLAB стосується запису чисел. Числа у звичній для нас формі

2,17 0,00217 $0{,}217\ 10^1$

(десяткова частина відокремлюється від цілої комою) для MATLAB треба писати із крапкою:

2.17 0.00217 .217e+1,

або ще так: .217e+1 або .217e-2. Скільки значущих цифр буде видавати MATLAB у своїх відповідях – залежить від обраної користувачем форми (*short* або *long*, без суфіксів, або з суфіксами E, D), вказаної в пункті меню

File\Preferences… \Array Editor.

Можна також виконати команди

>> format short

або

>> format long

Команда MATLAB у командному рядку може складатись з кількох операторів. Тоді їх треба відокремлювати один від одного за допомогою коми "**,**" або крапки з комою "**;**". У першому випадку після *Ввод* отримаємо значення всіх змінних, що обчислювалися. У другому випадку обчислення будуть проведені, але надруковані не будуть. Це слід використовувати, аби виводити важливі дані (кінцеві, вузлові або результати обчислень задля відлагодження) і не виводити другорядні.

Отже, можна починати практичну роботу в середовищі MATLAB!

1.3. ПРИЙОМИ РОБОТИ З MATLAB. ДОПОМОГА ВІД MATLAB

Щойно ми працювали у командному вікні MATLAB, вводячи певні команди (накази) після „запрошення" ">>". Ознайомимось тепер з вікном MATLAB "Історія команд" (ліворуч на рис. 1.1, праворуч на рис. 1.2.). У цьому вікні фіксуються час і дата даної і попередніх сесій роботи і всі команди, які робив користувач у даній сесії роботи (після

запуску MATLAB), і навіть у попередніх сесіях. (Щоб очистити "Історію", треба правою кнопкою миші клацнути в довільному місці вікна і в меню, яке з′явиться, обрати "*Delete Entire History*").

На рисунках у вікні "*Command History*" видно лише частину команд, що виконувалися. Щоб побачити більше, треба курсор миші навести на смугу з правого боку вікна і "потягнути" смугу вниз. Так само можна, навпаки, розширити командне вікно, коли курсор перетвориться у "↔".

Деякі спеціальні прийоми можуть бути корисними під час роботи з MATLAB. Часто треба повторити команду, яка виконувалася раніше. Для цього існує кілька способів:

1. Якщо треба повторити щойно виконану команду – не треба знову набирати її з клавіатури. Натисніть клавішу "стрілка уверх" ↑ — і остання команда з′явиться у командному рядочку знову. Натисніть "стрілку уверх" ще раз – з′явиться передостання команда. Такий прийом зручний для двох - п′яти останніх команд.
2. Якщо команда виконувалася давно в дану сесію, або навіть в попередню – знайдіть її у вікні „Історія" (можливо, для цього потрібно піднятися по вікну „Історія" вище). Виділіть потрібну команду або потрібну її частину (лівою кнопкою миші "протягніть" від початку до кінця потрібного тексту). Тепер треба скопіювати виділене у допоміжну пам′ять за допомогою комбінації клавіш <Ctrl–C>. Нарешті, повернутися до чергового Запрошення >> і вставити потрібний текст комбінацією <Ctrl–V>.
3. Можна і так зробити: якщо потрібна команда знаходиться у вікні "*Command History*" ("*Історія команд*"), клацніть по ній лівою кнопкою миші і в такий спосіб виділіть її. Тепер натисніть ліву кнопку і "перетягніть" команду у командне вікно.

   (У старших версій MATLAB вікно "*Command History*" стало ще зручніше: у переліку виконаних команд з′явилися значки  і  . Натискуючи на перший

„розвертаємо” команди за той чи інший сеанс роботи (вказана його дата); натискуючи на  , „згортаємо” перелік команд. В результаті, вікно історії команд виглядає більш компактно, у ньому стало легше орієнтуватися).

В короткому посібнику розповісти про MATLAB все – просто не можливо. Користуйтеся книжками, список яких наведено наприкінці. Окрім того, суттєву допомогу можна отримати від самого MATLAB прямо біля комп′ютера – так званий *Help*. Ось деякі способи.

Обрати меню *Help* (праворуч у верхньому рядочку меню, рис. 1.1 або 1.2) і потім вибір *Using the Desktop* або *Using the Command Window* з меню, яке з′являється, призводить до появи кольорового вікна Help з поясненнями англійською мовою щодо, відповідно, загального вигляду програми MATLAB та, окремо, вікна команд. (У кожній версії MATLAB це зроблено трішки інакше).

Натискання на клавішу із знаком запитання **?** (нижній рядочок меню, праворуч), або ж, через меню, *Help* \MATLAB Help призводить до появи того ж вікна *Help*, де можна розшукувати потрібну інформацію. Вікно складається з двох половин – панелі *Help*-навігатора (ліворуч) і панелі інформації. Навігатор подає зміст довідкової системи. Як тільки курсором виділено потрібний розділ – інформація з нього відразу з′являється на правій панелі. Не дивуйтеся: тут все – англійською мовою, навіть в „русифікованій” версії MATLAB. Що ж робити – англійська сьогодні є міжнародним стандартом ☺...

Для початкового ознайомлення з MATLAB доречно запустити демонстраційний “мультфільм”: в меню обрати **Help (Помощь** – так в русифікованій програмі MATLAB) і підменю **Demos (Демонстрации)**. Далі обрати одну з тем демонстраційного показу: Desktop Environment (інтерфейс програми MATLAB), Matrices (матриці), Numerics (деякі чисельні методи – підбір емпіричної функції, інтерполяція, диференціальні рівняння та інші), Graphics (побудова графіків), Language (мова програмування в MATLAB),

Gallery (деякі “топологічні образи”) або More Examples (графічні ілюстрації до деяких математичних задач). Письмові коментарі до "мультиків" – теж англійською мовою.

Нарешті, ще один спосіб отримати інформацію – нам він видається найбільш зручним. Якщо, припустимо, ви знаєте, яку саме команду або функцію MATLAB бажаєте використати, але забули її формат – тоді просто з командного рядка даєте наказ:

>> help Ім'яКоманди,

наприклад: *>> help plot* або *>> help log*. Отримаєте текст з підказкою відповідно про побудову графіків командою ***plot*** та логарифмічну функцію ***ln***. Доречно також запам’ятати:

>> help elfun

надає інформацію про всі елементарні функції, відомі вам зі школи (“*elfun*” скорочення від “elementary functions”); деякі з них пишуться тут на американський манер (*tan* замість *tg*, *log* замість *ln*, *log10* замість *lg* тощо).

Допоміжні матеріали, які для вас мають якесь особливе значення (плануєте використовувати найчастіше або детально вивчити пізніше), можуть бути відібрані у вашу особисту колекцію. Натисніть на кнопку *Add to Favorits* (Додати у Вибране) праворуч зверху – і надалі відповідна сторінка *Help* постійно зберігатиметься в розділі *Favorits* (Вибране) *Help*-навігатора.

1.4. МАТРИЧНА ЛАБОРАТОРІЯ MATLAB

Матрична арифметика – основа MATLAB (пригадайте походження назви цієї програми). Тому наступний матеріал вартий особливої уваги!

Для запису ***числових матриць*** в MATLAB всі елементи матриці заключають у квадратні дужки, елементи одного рядка відокремлюють один від одного *пропуском* або *комою* (значення не має), а рядок від рядка – *крапкою з комою* “**;**”. Тоді, матрицю

$$A = \begin{pmatrix} 1.7 & 0.021 & 120 \\ 7.34 & 11.08 & 0.0078 \\ 31.14 & 17 & 42.1 \end{pmatrix}$$

можна ввести у пам'ять MATLAB такими діями у командному рядку:

>> A=[*1.7 .021 120.* ; *7.34 11.08 7.8e-3* ; *3.114e+1 17.0 42.1*] (1.1)

(див. приклад 5 на рис. 1.3). Тоді натискання на *Ввод* призведе до того, що матрицю буде надруковано у командному вікні у звичній для нас формі у встановленому форматі для чисел (чотири знаки після десяткової точки у даному випадку).

Тепер зрозуміло, як передати системі вектор-рядок або вектор-стовпчик, припустимо, такі:

$$row1 = (1.7,\ 0.021,\ 120), \qquad col1 = \begin{pmatrix} 1.7 \\ 7.34 \\ 31.14 \end{pmatrix}.$$

У командному вікні MATLAB пишемо відповідно:

>> row1=[*1.7 .021 120.*], *col1*=[*1.7; 7.34; 31.14*] (1.2)

Натискання клавіші *Ввод* дозволяє перевірити, чи вірно зрозумів вас MATLAB. Наприкінці кожного запису ставимо крапку з комою „**;**", якщо хочемо „подавити" видачу на екран.

Система має також особливі команди для ввода деяких спеціальних матриць. А саме:

zeros(m,n) – матриця вимірності *m*×*n* з самих нулів (нуль-матриця),

ones(*m,n*) – матриця вимірності *m*×*n* з самих одиниць,

eye(*n*) – матриця вимірності *n* з одиницями по головній діагоналі, усі інші елементи – нулі (одинична матриця).

Пояснимо це конкретними прикладами:

$$\text{zeros}(2,3) = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \qquad \text{ones}(3,2) = \begin{pmatrix} 1 & 1 \\ 1 & 1 \\ 1 & 1 \end{pmatrix},$$

$$\mathrm{eye}(4)=\begin{pmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{pmatrix}.$$

Якщо вимірність матриці дуже велика – одного командного рядка може не вистачити. Щоб продовжити введення даних (або набір довгої команди) – наберіть знак продовження **...** (три крапки) і натисніть *Ввод*. Ці три крапки будуть сприйняті як особливий, службовий знак і помічені синім кольором. На новому рядку запрошення “>>” вже не буде. Продовжуємо на ньому введення інформації, і якщо треба – знов переходимо на наступний рядок за допомогою знака продовження **...** (трьох крапок). Коли ж, нарешті, введення закінчено, остаточно натискуємо клавішу *Ввод*. На новому рядку з'явиться знак запрошення “>>”, що свідчить про те, що MATLAB готовий до сприйняття наступної команди.

Проте для великих за розмірами матриць, які незручно набирати у вигляді (1.1), або незручно навіть їхні рядки або стовпчики у вигляді (1.2), існують і інші засоби. MATLAB дозволяє вводити їх частинами. Припустимо, для рядків і стовпчиків (1.2) ввели спочатку частини рядків та стовпчиків

>> row1a=[*1.7 .021*], *row1b*=[*120.*], *col1a*=[*1.7*], *col1b*=[*7.34; 31.14*];

Тоді, виконанням команд

>> row1=[*row1a*, *row1b*], *і* *col1*=[*col1a*; *col1b*]

передаємо у пам’ять комп’ютера повні вектори *row1* і *col1* у відповідності саме із виразом (1.2).

Аналогічно, якщо за такими правилами ввести другий *row2* і третій *row3* рядки матриці (1.1) (або другий *col2* і третій *col3* стовпчики з виразу (1.1)), то сформувати матрицю *A* в цілому можна за допомогою команди

A=[*row1*; *row2*; *row3*] *або* *A*=[*col1*, *col2*, *col3*]

Звернемо увагу ще раз: знак “ ; “ (крапка з комою) відокремлює кожний наступний рядок, знак “,” (або ніякого знака, пропуск) розділяє стовпчики!

Визначений вище *ідентифікатор A* буде для програми не якимось числом, а саме набором чисел, таблицею (масивом). На відміну від цього, запис $A(i, j)$ посилається вже на конкретне число з набору, яке знаходиться на перетині рядка i і стовпчика j, за умов $i \le n$ і $j \le m$. Наприклад, $A(1,2)=0.025$, $A(2,3)=0.0078$. Важливо розуміти і більш складні посилання, наприклад – на частину певного рядка або стовпчика. Якщо "замовляємо" MATLAB-системі *A*(*1:2,2:3*) – тобто, "*надати таблицю* ***з рядками від 1 до 2*** *і із* ***стовпчиками від 2 до 3***", то це буде матриця

$$\begin{pmatrix} 0.021 & 120 \\ 11.08 & 0.0078 \end{pmatrix}.$$

Ось ще приклади зручної роботи у „МАТричній ЛАБораторії" – посилання на частини введеної матриці:

>>r1=A(**:** ,*1*) – перший стовпчик матриці *A*, тобто "рядки – усі, а із стовпчиків обирати лише з індексом *1*".

>>A1=A(*2*, *2* **:** *end*) означає "елементи другого рядка, у яких другий індекс змінюється від 2 до кінця".

>>A(**:** , **:**), або *A*(*1* **:** *end*, *1* **:** *end*) дає саму матрицю *A*. Таким чином,

$$\mathrm{r1} = \begin{pmatrix} 1.7 \\ 7.34 \\ 31.14 \end{pmatrix}, \qquad A1 = (11.08 \quad 0.0078).$$

MATLAB вміє робити складні перетворення матриць – їхнє додавання, віднімання, множення, і робить це у відповідності з загальними правилами матричної алгебри [15,19,35,41,45].

Додавати і віднімати вектори і матриці ($A+B$ і $A-B$) можна, якщо їхні виміри співпадають. Інакше MATLAB видасть повідомлення про помилку:

Error using ==> +

Matrix dimensions must agree.

При додаванні (відніманні) матриць названа операція здійснюється з кожною відповідною парою елементів

матриць A і B, і з цих сум (різниць) утворюється нова матриця $S=A+B$ або $R=A-B$ (по-елементні дії).

Результат **множення векторів або матриць** $A*B$ вимірності $i\times n$ і $n\times j$ є матриця вимірності $i\times j$ (вимірність результату, тобто, утворюється з лівого показника першого множника і правого показника другого, а їх суміжні показники мають співпадати – звичайне правило теорії матриць). Кожний з елементів матриці-добутку обчислюється за складним правилом, що вивчають у лінійній алгебрі на першому курсі університету. MATLAB його знає, а студент може пригадати з підручників [35].

Наслідком цього є **піднесення матриці до степеня**. Дійсно[8], $A^2 \overset{Df}{=} A*A$. Тобто така операція для матриці $i\times n$ і $i\times n$ можлива, коли $i=n$, тобто **лише для квадратних матриць**. Аналогічно, третій степінь матриці $A^3 \overset{Df}{=} A^2*A$ визначений, знов-таки, лише для квадратних матриць. Так само четвертий степінь $A^4 \overset{Df}{=} A^3*A$, і так далі...

Операція ділення *A/B* ні для векторів, ні для матриць у підручниках з математики не визначена. Проте, MATLAB-система все-таки вводить таку операцію ділення, розуміючи під нею множення $A*B^{-1}$. Така операція визначена, коли B – квадратна невироджена матриця $n\times n$, а A — матриця з довільним числом рядків і з n стовпчиками. Аналогічно визначається в MATLAB *операція ділення* "*справа наліво*": $A\backslash B = A^{-1}*B$. Припускається, що A – квадратна матриця $n\times n$, а B має вимірність $n\times k$, наприклад – вектор-стовпчик з n рядків. В такому випадку $A\backslash B$ є розв′язком системи лінійних рівнянь $Ax=b$, коли x і b є вектор-стовбці.

Увага: B^{-1} – усього лише *позначення* для оберненої матриці, яку слід обчислювати як *inv*(*B*)!

[8] ***Df*** над знаком рівняння означає *after Definition*, „за визначенням".

MATLAB вводить і інші “нелітературні” операції “*поелементне множення*” і “*поелементне ділення*” (операції з крапкою), коли матриці мають однакові виміри і операція застосовується до їхніх відповідних елементів, наприклад:

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1k} \\ a_{21} & a_{22} & \dots & a_{2k} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nk} \end{pmatrix} .* \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1k} \\ b_{21} & b_{22} & \dots & b_{2k} \\ \dots & \dots & \dots & \dots \\ b_{n1} & b_{n2} & \dots & b_{nk} \end{pmatrix} = \begin{pmatrix} a_{11}*b_{11} & a_{12}*b_{12} & \dots & a_{1k}*b_{1k} \\ a_{21}*b_{21} & a_{22}*b_{22} & \dots & a_{2k}*b_{2k} \\ \dots & \dots & \dots & \dots \\ a_{n1}*b_{n1} & a_{n2}*b_{n2} & \dots & a_{nk}*b_{nk} \end{pmatrix}.$$

Аналогічно, “*поелементне ділення*” двох матриць $n \times k$ – це нова матриця, яка позначається *A* **.** */B* і складається з відповідних пар a_{ij} / b_{ij}. Бачимо, що MATLAB сконструйовано саме для роботи з матрицями!

Для такого „*множення з крапкою*”, аналогічно попередньому множенню, можна визначити **піднесення до степеня**.

Створити в MATLAB *упорядкований* вектор-рядочок дуже просто. Для цього призначений оператор **:** (дві крапки, semicolon). Конструкція

>> v=*3.5* **:** *.15* **:** *5.0* **;** (1.3)

створює вектор v з першим елементом *3.5*, другим елементом *3.65*, третім елементом *3.80*, і так далі з кроком *.15*, поки черговий елемент не наблизиться до *5*. Коли у виразі (1.3) присутній лише один семіколон „**:**”, то крок обирається *1*, тобто вектор v*1=3.5* **:** *5.0* складається лише з елементів *3.5* і *4.5*.

<u>Увага:</u> в конструкціях (1.3) слід не забувати „подавити” вивід на екран, тобто ставити крапку з комою наприкінці ; інакше весь екран може “засипати” великою кількістю цифр!

Коли v ─ вектор, то й змінні *x=sin*(v), *y=tan*(v), *z=exp*(v) стають векторами з елементами, що дорівнюють вказаним функціям від відповідних елементів вектора v. Наприклад, *z1=exp*(v1) є вектором з двох елементів *z1*=[$e^{3.5}$, $e^{4.5}$], а

z=*sin*(v) є вектором з елементів *z1*=[$\sin 3{,}5$, $\sin 3{,}65$, $\sin 3{,}80$, . . ., $\sin 5{,}0$].

Проте, спроба в такий спосіб виконати команду

>> *z*= *sin*(v) **/** v або >> *z*=v******exp* (v)

призведе до повідомлення про помилку на зразок

??? *Error using ==> **
Inner matrix dimensions must agree.

Чому не вдається? Адже ж Ви замовили операції над векторами *sin*(v) і v, а вони не для кожної пари операндів коректно визначені! Використайте ті ж самі операції, але "з крапкою" **.** / , **.*** – і отримаєте жаданий результат! Розуміння різниці між вказаними операціями буде до нагоди у розділі 1.5.

Комплексне число[9] *z* = *x*+***i****y* – це той самий вектор з двох елементів *z* = {*x* , *y*}. Тому не дивно, що MATLAB вміє робити обчислення з комплексними числами. При цьому останні можна записувати як *z* = *x*+***i****y*, або *z* = *x*+***i*****y* (зі знаком множення, так і без нього!), так і як *z* = *x*+***j*** *y* (тобто уявну одиницю $\sqrt{-1}$ можна позначати як ***i***, так і ***j***). Легко перевірити комплексну арифметику:

```
>> (2+3j)+(3+2i)
    ans =   5.0000 + 5.0000 i

>> (2+3j)*(3+2j)
    ans =    0 +13.0000 i
>> sqrt(j)
    ans =   0.7071 + 0.7071 i
```

(**Ще раз:** у випадку комплексних чисел знак множення на уявну одиницю можна не ставити!)

1.5. ФУНКЦІЇ ВІД МАСИВІВ

MATLAB орієнтований на систематичне використання масивів. „Масив" – так програмісти називають матриці. Їх

[9] Оскільки не у кожній школі зараз вивчають тему „Комплексні числа", радимо проконсультуватися у Вікіпедії.

використання дозволяє розробляти більш загальні і короткі алгоритми, а також значно збільшити швидкість обчислення складних задач. Заради цього програми визначення певних характеристик масивів, такі, як знаходження розмірів масиву або його найбільшого (найменшого) елементів тощо, включені в MATLAB як його "внутрішні" функції (команди). Їх слід використовувати у наших обчисленнях.

1. Якщо треба знайти або використати **розмір вектора** (тобто кількість елементів одновимірного масиву), то можна застосувати функцію *length*. Як і у випадку звичайної функції, за ім'ям функції у дужках вказують аргумент, у даному випадку – ім'я вектора, довжину якого шукаємо. Вище були визначені вектори *r1* і *A1*. Для них

$$length(r1)=3, \qquad length(A1)=2.$$

Таку функцію можна застосувати і до матриці. Отримаємо найбільшу з двох вимірностей матриці, наприклад

$$length(zeros(2,3))=3.$$

(Приклад використання див. в підрозділі 3.2).

Проте, щодо матриць часто треба знати її обидві вимірності, тобто вектор з двох чисел. Тому треба застосувати іншу функцію, *size*, у форматі

$$[M\ ,\ N] = size(A).$$

Виконання такої команди видає на екран величину змінних *M* і *N*, з яких перша, *M*, дає кількість рядків матриці *A*, а *N* – кількість стовпців. Деякі інші корисні формати застосування такої функції див. в довіднику *Help*, або *>>help size*.

2. MATLAB вміє обчислювати визначники матриць і знаходити обернені матриці; для цього існують внутрішні MATLAB′івські функції

$$det(A) \qquad \text{і} \qquad inv(A).$$

Зауважимо ще раз, що MATLAB обов'язково звертає увагу на відповідність вимірностей матриць, дії з якими йому запропоновані. Якщо він "залається"

Error using ==> +
Matrix dimensions must agree,

то це означає, що користувач запропонував йому некоректну математичну операцію, і MATLAB нагадує: "*Виміри матриць повинні погоджуватись*". Якщо ж MATLAB дає попередження

Warning: Matrix is singular to working precision,

то це означає неможливість отримати обернену матрицю, оскільки її визначник дорівнює " машинному нулю", матриця вироджена.

3. Функції *max*(*A*) і *min*(*A*) у випадку, коли *A* є вектор, дають відповідно найбільший і найменший з його елементів. Так, *max*(*r1*)=*31.14*, *min*(*r1*)=*1.7*. Однак, ми не отримали положення знайденого елемента у векторі. Аби знайти ще й положення, існує й інший формат звертання до цих функцій: [*E*, *n*]. В такому випадку MATLAB визначить два числа – елемент вектора *E*, найбільший чи найменший, і також номер елемента у векторі *n*. Приклад: на команду

>> [*E, n*]=*max*(*r1*)

MATLAB відповість *E=31.14*, *n=3*. Результатом команди

>> [*E*, *n*]=*min*(*r1*)

буде *E=1.7*, *n=1*.

4. Доволі часто використовується функція *sum*(*A*). Коли *A* є вектором, то названа функція видає суму всіх його елементів. Якщо ж *A* є матрицею з *n* стовпчиків, то відповіддю буде вектор-рядок з *n* елементів, кожний дає суму елементів відповідного стовпчика *A*. Приклади:
>> *S*=*sum*([*1 2 3*]) =*6*; *S*=*sum*([*1 2 3* ; *4 5 6*]) =[*5 7 9*].

5. Функція *норми вектора* $norm(\mathrm{v}, p)$ обчислюється для цілих $p=1$, $p=2$, $p=3$ за формулою

$$norm(\mathrm{v}, p) = \left(\sum_{i=1}^{n} | \mathrm{v}_i |^{p} \right)^{1/p}.$$

Очевидно, що $norm(\mathrm{v},1) = \sum |\,\mathrm{v}_i\,|$ – просто сума модулів компонент вектора, $norm(\mathrm{v},2) = \sum \mathrm{v}_i^2$ – сума квадратів. Такі функції стають у нагоді для розв'язування лінійних алгебраїчних рівнянь, для побудови ітераційних алгоритмів, для оцінки „близькості" сигналів (векторів).

6. Ще одна внутрішня функція MATLAB може буди корисною. Фукція *diag(A)*, якщо вона застосовується до квадратної матриці *A* вимірності *n*x*n*, видає вектор-стовпчик вимірності *n*, який складається лише з елементів головної діагоналі *A*. Якщо ж функцію *diag*(v), застосувати до вектора v з *n* елементів, то отримаємо матрицю *n*x*n*, в якій всі елементи дорівнюють нулю, окрім головної діагоналі, по якій розташовані елементи вектора v. Таким чином, матриця

```
>> D=diag(diag(A))
```

є квадратною матрицею тієї ж вимірності *n*x*n* , що і матриця *A*, в якій елементи на головній діагоналі співпадають з відповідними елементами *A*, а усі останні дорівнюють нулю. Радимо перевірити роботу такої функції самостійно. (Вказівка: коли функція набрана у Командному рядочку, виділить її мишкою та у контекстному меню, що з'явиться, оберіть „*Help on Selection* F1").

Повний перелік функцій над матрицями дивіться у довіднику Help, або командою *help elmat* (Elementary matrices).

У будь-якій іншій алгоритмічній мові при використанні змінних та особливо масивів є обов'язковим „повідомляти компілятор" про тип змінних та, щодо масивів, про їх вимірність. У мові Java, наприклад,

```
int n, m=5;
int[] N={2, -4, 6};
double x, pi=3.1415926;
double[] X={2.1, -1.2};
char[] Sym={'Hello, World!'};
```

такий запис наказує компілятору зарезервувати у пам'яті комп'ютера для *N* втричі більше місця, аніж для *n*, бо це є масив; для *X* вдвічі більше аніж для *x*; а блок пам'яті для *Sym* має розміщувати рядочок з 13 символів „Hello, World!".

А от у MATLAB цього робити не треба! Він завжди готовий працювати саме з масивами (матрицями). Якщо останній о ході обчислень збільшує свій розмір – місце у пам'яті буде збільшено „динамічно".

1.6. ПОБУДОВА ДВОВИМІРНИХ ГРАФІКІВ

Графічне зображення результатів розрахунків – найбільш інформативне і переконливе. Тому дуже важливо засвоїти хоча б частину тих вражаючих можливостей, які надає MATLAB. Перш ніж вивчати наведену нижче теорію, пропонуємо спочатку спробувати будувати графіки самостійно шляхом виконання лабораторної роботи 2. Після цього – подивіться даний розділ.

Середовище MATLAB має декілька спеціальних команд побудови графіків. Їх вівчення почнемо з найпростіших. Команди *ezplot* і *fplot* майже не потребують якихось знань; недаремно їх називають easy-to-use commands, тобто легкими для застосування.

1. Припустимо, треба побудувати графік функції $y = \dfrac{\sin x}{x}$, яка відіграє дуже важливу роль у математиці, і навіть „впроваджена" у Вашому мобільному телефоні. Зробити це дуже просто. Наберіть у командному рядочку MATLAB

```
>> ezplot('sin(x)/x')                    (1.4)
```

і натисніть „Ввод". Недаремно цю команду називають *easy-to-use*!

Ну, а якщо графік треба побудувати на ширшій області визначення, припустимо, для $-5\pi \le x \le 10\pi$, то додатково вказуємо цей діапазон у прямокутних дужках:

>> *ezplot*(' *sin*(*x*)/*x* ', [*-5*pi*, *10*pi*, *-.3*, *1.1*])

Тут остання пара [***-.3, 1.1***] чисел позначає діапазон осі *Oy*, в якому хочемо бачити графік по вертикалі. Його вказувати не обов'язково, та за його допомогою можемо збільшувати або зменшувати зображення відносно вікна (ще одного!) Figure. Можна не вказувати і перший діапазон, по горизонтальній осі *Ox*, як в (1.4). Але тоді він, за умовчанням, дорівнює $-2\pi \le x \le 2\pi$.

Увага: в (1.4) і нижче виділені апострофи ', в які ОБОВЯЗКОВО треба заключити рівняння функції! Чому це так – дивіться розділ 1.7.

Побудуйте якийсь інший графік, наприклад[10]

>> *ezplot*(' *abs*(*sin*(*x*)/*x*) ', [-5**pi*, 10**pi*, -.3, 1.1])

Зверніть увагу, що він „затирає" попередній! Але ж, як порівняти два та більше графіків в одному вікні? Або, навпаки, як кілька графіків побудувати у різних вікнах?

Вказівка: для більш детального вивчення команди *ezplot* зробіть кілька власних графіків, подивіться Help та зробіть лабораторну роботу 1, де детально вивчіть „іконки" і меню графічного вікна!

2. Команда MATLAB ***fplot*** також легка до застосування. Вона дозволяє побудувати кілька графіків одночасно, але відрізняється *форматом*, правилами її запису:

>> *fplot*(*'* [*sin*(*x*)*/x* , *sin*(*x*)**x*] *'*, [*0* *2*pi*])

Яка розумна: кожну криву малює окремим кольором! Згідно із *форматом команди*, апострофи[11] *'* *'* заключають увесь перелік функцій, що стоять у квадратних дужках **[** *sin*(*x*)*/x* , *sin*(*x*)**x* **]** – їх може бути скільки завгодно, відокремлених комою або пропуском. Вказівка на діапазон побудови графіків **[***0 2*pi***]** для цієї команди також є ОБОВ'ЯЗКОВОЮ.

---

[10] Тут функція *abs*(*x*) означає модуль аргументу | *x* |.

[11] Призначення апострофів пояснюється в розділі 1.7.1.3.

3. Розглянемо тепер команду ***plot***, більш складну для використання. Перші дві команди „порушили" те, чому вчили у школі: будувати графік „по точках". Наступна команда реалізує саме такий алгоритм.

Графік одновимірної залежності – це є лінія на площині *x*O*y*, яка з′єднує певні точки P_i з координатами $\{x_i, y_i\}$ на площині. Це значить, що ми повинні вказати MATLAB таку систему точок. Після цього програма *plot* з′єднає їх лініями. Покажемо це на прикладах.

3.1. Ця команда, на відміну від попередніх, може бути застосована до функцій, що задані у вигляді таблиці (стор. 98, 102, 111), а не формули. Наприклад, маємо якусь експериментальну таблицю, де у залежності від змінної *X* наведені виміри величин *Y* і *Z*. Вводимо у середовище МАТЛАБ значення аргументу і виміряних функцій *Y* і *Z*:

>> x=[*0,.7, 1.5, 2.3, 3, 3.7, 4.5, 5.1, 6, 7*];

>> y =[*0, 0.451 , 1.496, 1.715, 0.423, -1.960 , -4.399 , -4.7217, -1.676, 4.599*];

>> z =[*-1.00*, *0.770*, *0.815*, *0.492*, *0.082*, *-0.275*, *-0.460*, *-0.4100*, *-0.114*, *0.248*];

Будуємо обидві експериментальні залежності однією командою, що сполучає експериментальні точки синію та зеленою лініями (це наказано в апострофах '***b****lue*' та '***g****reen*'):

>> plot(*x*,*y*, '***b***', *x*, *z*, '***g***') (1.5)

Ті ж точки позначимо значками без ліній

>>hold on, plot(*x* , *y*, '***r****o*', *x*, *z*, '***r****h*') (1.6)

(колір значків – '***r****ed*') і пояснимо командами

>> title('Експериментальні дані')

>> legend(' *Величина Y* ' , *'Величина Z* ') (1.7)

MATLAB автоматично співставить назву лінії з її позначками.

3.2. Нехай потрібно побудувати графік функції $y=sin(x)$ в діапазоні зміни аргументу $x \in [-\pi, 3\pi]$. Припустимо також, що нам достатньо мати 10 точок на цьому відрізку довжиною $l = 4\pi$, тобто крок між точками має бути $\Delta x = 0{,}4\pi$. У командному вікні MATLAB виконаємо таку послідовність команд:

>> x =-pi **:** *.4*pi* **:** *3*pi* % Задали значення аргументу
>> y = sin (*x*) % Отримали значення *sin*(*x*)
>> ***plot***(*x,y*)

Перша команда, як вже знаємо, будує *упорядковану послідовність*, тобто вектор чисел *x,* про що MATLAB (крапка з комою не поставлена!) повідомить на зразок:

x=
Columns 1 through 6
3.1416 -1.885 -0.62832 0.62832 1.885 3.1416
Columns 7 through 11
4.3982 5.6549 6.9115 8.1681 9.4248.

Друга команда обраховує вектор з ім'ям *y* та із значеннями функції *sin*(*x*) від відповідних значень *x*. А третя команда вже будує на площині *x*O*y* утворені в такий спосіб точки $\{x_i, y_i\}$ і з′єднує їх лінією. Результат навряд чи сподобається: природно, що графік доволі грубий, бо крок між точками осі O*x* заданий як $0{,}4\pi \approx 1{,}3$. Графік буде виглядати краще, якщо всі наведені команди повторити, задавши в десять разів (або навіть у сто!) менший крок аргументу:

>> x=-pi **:** *.004*pi* **:** *3*pi* **;** *y=sin* (*x*) **;** *plot*(*x,y*)

Увага: тепер команди розділені між собою знаком **;** , аби не ускладнювати вигляд екрану виводом двічі по 110 даних з масивів *x* і *y*. Не забувайте використовувати „**;**" коли маєте справу з великими масивами чисел!

Нагадаємо, що команду можна використати з двома і більше парами аргументів, як (1.5); побудувати лінію певного кольору одночасно із значками на зразок (1.6), наприклад *>>***plot**(*x,y*, '**r***o-*'); надати пояснень до кривих як (1.7). Запросіть в MATLAB розповісти про це детальніше,

>>help plot.

3.3. Сказаного ще недостатньо для побудови графіка функції, скажімо, $y = \dfrac{\sin(x)}{x}$. Для утворення масиву значень функції y треба застосовувати не звичайне ділення, а "***ділення з точкою***" **./** . Тоді послідовність команд

```
>> x= - pi : .01*pi : 3*pi;  y=sin (x) ./ x ;   plot(x,y)        (1.8)
```

дасть нам потрібний графік, побудований для області $x \in [-\pi, 3\pi]$. (Аналогічно, у випадку функції $y = x \sin x$ слід використати „множення з точкою" **.***/*)

4. А як побудувати два графіка на одному рисунку, припустимо, ще графік функції $y = x^{1/3} \sin(x)$?

4.1. А як відрізнити одну лінію від іншої?

5. Отримані графіки можна зробити більш "красивими" за допомогою команд *legend*, *title*, *grid*, *xlabel*, *ylabel*, *axis* – разберіться у цьому самостійно.

6. Нарешті, спробуйте побудувати графік функції (1.8) у вигляді "комети":

```
>> N=10000; x= -10*pi:pi/N:10*pi; comet(x,sin(x)./x).
```

(А як „керувати" швидкістю „комети"? Ще раз спробуйте, але цього разу з *N=1000*)! Красиво?

7. Інший спосіб зробити графік більш інформативним – використати розвинене меню, яке включене в графічне вікно *Figure*. Дослідіть самостійно!

8. Що робити, аби зберегти вже побудовані графіки, а наступні – побудувати від них окремо?
Виконайте попередньо команду

```
>> Figure.
```

Тепер з′явиться нове вікно, поки ще пусте, для наступних графіків. Можна замовити вікно з певним номером: *Figure*(*3*), і т.п.

9. Ми бачили, що команда *plot* потребує певної підготовчої роботи – подбати про масив значень аргументу, крок між ними тощо. Нагадаємо, що цю роботу можна

доручити “більш розумним” функціям MATLAB *fplot* і *ezplot*. Наступні приклади використовують розширені *формати*:

>> fplot (' [tan(x)/x, sin(x)/x]' , [-pi/2 pi/2 0 4] ,': m')

та

>> figure, fplot (' tan(x)/x' , [-pi/2 pi/2 0 4] ,'-- pc')

В першій позиції, першому аргументі, (**у лапках!**) вказані

- функції, які треба побудувати – в останньому випадку *y=tg(x)/x*;
- другий аргумент – щоб вказати (у прямокутних дужках!) цікаві для вас області зміни аргументу $-\pi/2 \le x \le \pi/2$ (для команди *fplot* це обов’язково, для команди *ezplot* можна не писати!)
- область зміни функції $0 \le y \le 4$ можна опустити;
- нарешті, у третьому аргументі можна вказати (знов **у лапках!**) тип лінії і її колір (у даному випадку *':'* означає пунктирну лінію, *'p'* використання маркеру „зірочка” („*pentagon*”) у точках, *'m' 'c'* відповідають кольорам кривої або маркерів *'**m**agenta'* та *'**c**yan'*).
- Дозволені стилі можна знайти у ***Help***

>> help plot ,

або за таким шляхом:

Contents → MATLAB → Using MATLAB → Graphics → Basic Plotting → Specifying Line Style

або

Specifying the Color and Size of Lines

10. Часто виникає потреба надрукувати побудований графік. Для цього натисніть кнопку  вікна *Figure*. Існує також багато способів вставити графік у ваш звітний Word-документ (дипломну роботу тощо).

10.1. Перший спосіб: у вікні *Figure* виконати

Edit \ Copy Figure,

після чого перейти у *Word*-документ і зробити *Правка\Вставить*.

10.2. Другий спосіб передбачає попереднє збереження рисунку у тому чи іншому графічному форматі:

в меню по шляху *File\Export*… обрати той чи інший графічний формат (*bmp*, *jpg*, інший) та зберегти файл у вашій робочій директорії (папці). Тепер, або в інший зручний час рисунок можна вставити у *Word*-документ за допомогою *Вставка\Рисунок* ►*\Из файла…*

11. MATLAB, як ми знаємо, вміє працювати з комплексними числами, стор. 36. Це значно розширює наші можливості у будуванні цікавих графіків [1].

Побудова графіків дозволяє глибше засвоювати фізико-математичні науки, і просто дає насолоду від роботи за комп'ютером!

1.7. ЧИСЕЛЬНІ І "АНАЛІТИЧНІ" ОБЧИСЛЕННЯ В MATLAB

Нагадаємо, що розділи посібника, відмічені знаком (окуляри), під час першого читання можна пропустити. З іншого боку, на ньому базується значна кількість наступних задач з програмування. Для останнього він нам буде конче потрібний.

Числа, а також вектори і матриці з ними (тобто *числові масиви*) – це **числові дані**, бо складаються з певної кількості чисел, і дії над ними проводяться за арифметичними алгоритмами. Логіка дій і, відповідно, алгоритми дій – зовсім інші, коли ми проводимо аналітичні перетворення, такі, як, припустимо, піднесення до квадрату

$(a+b)^2 = a^2 + 2ab + b^2$. Результат такого перетворення дійсний для довільних чисел a і b.

Також ми знаємо, що

$\sin^2\varphi + \cos^2\varphi = 1$ для будь-яких φ. А чи знає про це MATLAB?

Чи знає MATLAB матеріал 5-го – 11-го класів звичайної школи? Про це тут мова.

В початковій школі ми навчилися *роботі з числами*, а в старших класах вивчили правила роботи з узагальненими *аналітичними* виразами. „Вища" математика університету поширює можливості *аналітичної* роботи з формулами. Останні є вірними для будь-яких чисельних значень символів, коефіцієнтів, що входять до формули. У цьому – принципова різниця між **чисельними** і **аналітичними** об'єктами. Комп'ютер створений для роботи з числами. А от працювати з аналітичними, символьними об'єктами – за цим стоїть складні алгоритми і програми.

Доречно згадати, що алгоритми **аналітичної математики** для роботи на комп'ютері одними з перших у світі були розроблені в Інституті кібернетики Національної Академії Наук України і реалізовані в ЕОМ "МИР-2". Більш конкурентоспроможним виявився, однак, американський пакет ***Maple***, алгоритми якого пізніше увійшли і в MATLAB. Тут познайомимось з деякими корисними командами з *Symbolic Math Toolbox* (допоміжного пакета *Символічна Математика*), що входить до складу MATLAB.

1.7.1. Яка між ними різниця?

Різницю між числовими і аналітичними діями, розглянемо на прикладі диференціювання.

Ще зі школи та з перших лекцій з математичного аналізу ми знаємо, що диференціювати деяку функцію $f(x)$ – це значить, знайти за певними правилами іншу функцію $f'(x)$. Відомі приклади подані у таблиці:

Функція	**Її похідна**
x^n	nx^{n-1}
$\sin x$	$\cos x$
e^x	e^x
$f(x)+ g(x)$	$F'(x)+ g'(x)$
$f(ax)$	$a f'(ax)$
І так далі, за відомою таблицею похідних...	

Дехто навіть пам'ятає **визначення** (Definition) похідної:

$$f'(x) \overset{Df}{=} \lim_{\Delta x \to 0} \frac{f(x+\Delta x) - f(x)}{\Delta x}. \qquad (1.9)$$

Такий шлях – це **аналітичний спосіб** дій! Він дозволяє знайти функцію $f'(x)$ за функцією $f(x)$. Уявіть собі, MATLAB такому способу навчено! Ось приклади *аналітичних обчислень* у командному вікні MATLAB:

```
>> syms x n a                                        (1.10)
>> y1=x^n;  y2=sin(a*x);
>> dy1=diff(y1);  dy2=diff(y2);
>> pretty(dy1);   pretty(dy2)
```

Величини *dy1* та *dy2* зберігають тепер похідні функцій з першого і другого рядочків таблиці. Функція ***diff***( ) вміє диференціювати, результат – „у вигляді рядку". А команда (функція) ***pretty***( ) вміє цей останній зобразити на екрані у звичному нам вигляді. Спробуйте аналогічно самі!

А тепер розглянемо інший, **чисельний**, спосіб обчислень.

Приклад. Нехай з фізичного експерименту, з вимірів, виникла якась таблиця функції *Y* в залежності від аргументу *X*, такі наприклад, як подано далі. У фізика часто виникає питання – як швидко змінюється *Y* у залежності від аргументу *X*? Тобто, мова йде про швидкість за формулою (1.9):

$$y'(x_i) \approx \frac{y(x_{i+1}) - y(x_i)}{x_{i+1} - x_i}, \quad \text{де} \quad i = 1,2,\cdots,N-1 \qquad (1.11)$$

(**Увага:** якщо значень функції *Y* було N, то значень похідної y' буде $N-1$, на 1 менше! Зрозуміло, тут неможливо отримати границю $\Delta x \to 0$, бо відстань між усіма x_i скінчена і тому обчислення за (1.11) лише наближене).

Ось як найзручніше працювати в MATLAB для обчислення такої похідної:

```
>> x=[0, 0.70, 1.50, 2.30, 3.00, 3.70, 4.50, 5.10, 6.00, 7.00];
>>%Введення значень аргументу
>>y=[0 0.451 1.496 1.715 0.423 -1.96 -4.398 -4.721 -1.676 4.599];
>>%Введення значень функції
```

```
>> length(x), length(x)
>>%Перевірка, чи співпадають кількість? Мають співпадати!
>> x2=x(2 : end);
>>%Утворили допоміжний вектор наступних значень аргументу
>> x1=x(1 : end -1);
>>%Утворили допоміжний вектор попередні" значень аргументу
>> y2=y(2 : end); y1=y(1 : end-1);
>>%Вектора з N-1 попередніх і наступних значень функції
>>Dy=y2-y1; Dx=x2-x1;
>>% Обчислили чисельник і знаменник формули (1.11)
>>DyDx=( y2-y1) ./ (x2-x1);
>>% Обчислили похідну за (1.11)
>> plot(x, y, 'g',x1, DyDx, 'r')
>>% Побудова графіків та пояснень до нього
>> title('Порівняння функції й її похідної')
>> legend('Виміряна функція', 'Швидкість її зміни')
```

Графік, що отримано, показує виміряну функцію (зелена крива 1) й її похідну (червона, 2). Можна зробити такий висновок: швидкість зміни функції Y(x) спочатку збільшується, та скоро починає зменшуватись. Все-ж таки, швидкість лишається додатною, бо функція Y(x) продовжує збільшуватись. А от коли Y(x) починає зменшуватись – швидкість стає від'ємною. Та незабаром швидкість знову починає збільшуватись, бо й функція Y(x) перестає спадати й починає збільшуватись.

Ми продемонстрували ***чисельний спосіб*** знаходження похідної, ***чисельне диференціювання***, на відміну від попереднього способу за формулами і правилами, ***аналітичного***. Зрозуміло, що чисельний спосіб є наближеним. Та чи має він „сильно відрізнятись" від аналітичного? Можна очікувати, що таке обчислення похідної (швидкості) буде тим точніше, і тим ближче до аналітичного результату, чим більше вимірів проведено зі зміною аргументу X від X(1) до X(*end*).

Чи це дійсно так – допоможе завдання 3 Лабораторної роботи 4.

Важливу різницю між числовими і символьними обчисленнями, між числовими і символьними об'єктами MATLAB покажемо й на інших прикладах. Це зробить

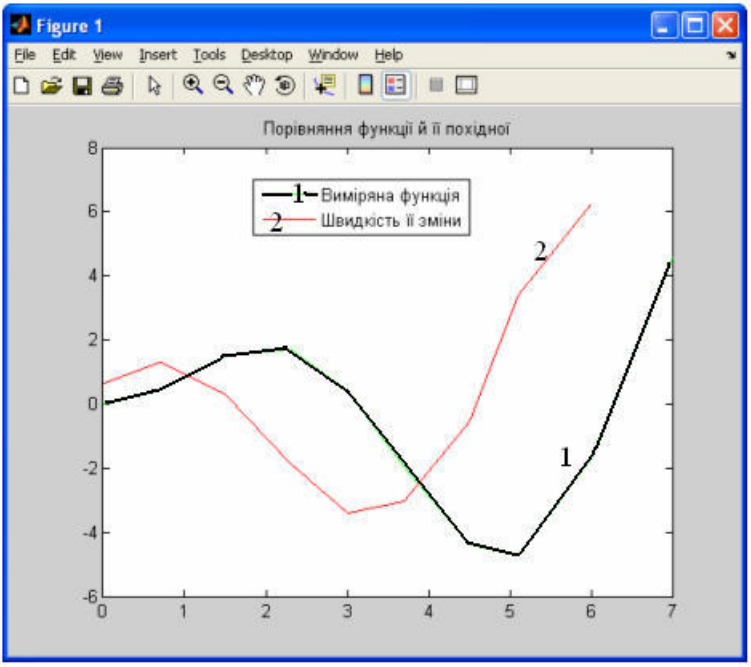


Рис. 1.4. Результат чисельного знаходження похідної (крива 2) за таблично заданої функцією 1 і наближеною формулою (1.11)

MATLAB вашим незамінним помічником у навчанні, виконанні курсових і дипломних робіт, а також в наступній практичній інженерній чи дослідницькій роботі.

1.7.2. ПОНЯТТЯ ПРО ТИПИ ДАНИХ. ТЕКСТОВІ І СИМВОЛЬНІ ДАНІ В MATLAB

На початку комп'ютерної ери були лише числові дані... Але поступово комп'ютер перетворювався на універсальний прилад, у тому числі – для роботи з текстами в офісах, друкуванню газет і журналів, роботі з фото і відео... Виникла потреба розрізняти ***типи даних***, працювати з кожним з них за різними правилами, по різному зберігати їх у пам'яті комп'ютера, обробляти їх різними програмами.

У розділі 1.5.6 пояснено щодо *даних числового типу* – ***int***, ***double*** – їх вказують майже в усіх алгоритмічних мовах

задля розміщення у пам'яті. Це – для поодиноких чисел. Для масивів таких даних в мові Java використовують позначення ***int***[ ], ***double***[ ]. З числовими масивами пов'язані зовсім інші можливі дії, операції, див. розділи 1.4, 1.5. В MATLAB такого робити не будемо, бо він „сам розуміє". У даному розділі та у наступній частині підручника познайомимось і з іншими важливими типами даних.

1.7.2.1. **Текстові дані** – це рядочки з будь-яких можливих символів алфавіту, наприклад

My Name Is…, *моє ім'я -- Євген*,

123+345 – хоча це нібито *числа*, але якщо ми не передбачаємо робити з ними *арифметичні дії*, то це просто текст. Кожна літера цих рядочків зберігається у пам'яті як певний числовий код. Наприклад, за літерою ´А´ має закріплено код 65, за літерою ´а´ – код 97.

Проте, як машині не переплутати код 55 з числом 7? Для цього, у багатьох мовах програмування текстові дані декларують за допомогою оголошення на зразок ключового слова ***char***. MATLAB поступає простіше. Достатньо заключити текстові дані у лапки (апострофи),

>> a11='Yevgen'
>> x='7'

– і машина вже не вимагатиме числового значення для змінної *Yevgen* та *x*. Далі узнаємо й інший спосіб...

Команди

>> Number1='123', Number2='345'

зовсім не надають числових значень змінним *Number1* і *Number2*, тому результат операції

>> Number1+Number2

зовсім не є число *468*! Це просто записи, що використовують цифри. Так само, змінна *F*, де

>> F=' sin(x)/x ' (1.12)

зовсім не є іменем функції, а просто – послідовність літер і символів (,) і /.

Для текстових рядочків визначені певні операції. Пропонуємо зробити такі „експерименти" у командному

рядку та, користуючись наданими коментарями, продумати, що вони означають:

```
>> %Конвертація символу у код:
>> double('A')
>> %Конвертація коду у символ:
>> char(55)
>>%Великі латинські літери мають коди від 65 до 90:
>> char(65 : 90)
>>%Узнати коди „особливих" літер українського алфавіту
>> double(' і ї є ')
>>%кодам від 48 до57  відповідають цифри 0123456789:
>> char(48 : 57)
```

Отримувати доволі складні „речення" можемо за допомогою вже відомих матричних операцій. У припущенні, що величини *a11* та *F* введені у командному рядочку MATLAB, отримувати таке дивне твердження:

```
>> aSentence=[a11,' is ',F] %Складення текстових даних
aSentence =  Yevgen is  sin(x)/x
```

1.7.2.2. **Матриці текстових даних** – це таблиці з символів, які, так само як числові, введено у MATLAB-середовище через прямокутні дужки, елементи у рядку відокремлені комою або пропуском, а для початку нового рядка використовується кома з крапкою „**;**". Приклад:

```
>>M=['My name is  ',' Студенти люблять' ;'  Yevgeny  ', . . .
'  MATLAB      ']
M =
        My name is     Студенти люблять
          Yevgeny       MATLAB
```

Єдина відмінність текстових матриць від числових – але дуже суттєва! – кількість позицій (тобто символів і пропусків) в елементах одного стовпчика мають співпадати! Так, в обох текстах першого стовпчика '*My name is* ' і '*Yevgeny* ' кількість позицій становить 13, а в текстах другого – 18. Перевіримо:

```
>> length('My name is  ')
     ans =   13
>> length('  Yevgeny  ')
     ans =   13
```

```
>> length(' Студенти люблять')
    ans =    18
>> length('    MATLAB       ')
    ans =    18
```

Якщо це правило порушити, отримаємо скаргу MATLABу:

```
??? Error using ==> vertcat
CAT arguments dimensions are not consistent.
```

Наприкінці і на початку найдовших текстів одного рядка зроблено додаткові пропуски, аби була відстань між ними після видачі на екран. В інших текстах стовпчиків додані пропуски усередині апострофів, аби забезпечити однакову *length* кожному тексту стовпчика.

Команди MATLAB, що працюють з текстовими даними та їх матрицями, дозволяють налагодити діалог між машиною й користувачем, зробити програми інтелектуальними...

Та чи не здається вам, що означене правило дуже обтяжливе – пам'ятати про нього, рахувати кількість позицій? Тому, програмісти вигадали нові типи даних, аби життя стало простіше..

1.7.2.3. **Символьні дані** (ще можна сказати **аналітичні дані**) – це такі ***текстові рядочки***, які можуть бути інтерпретовані як коректні, правильні математичні вирази. Так,

- *G=sin x* може слугувати текстовим даним, але не є символьним; та
- *G=sin*(*x*) або
- *F=sin*(*x*)*/ x* можуть слугувати як текстом, так і символьними даними.

Дійсно, *F* можна розуміти як математичну функцію $y = \frac{\sin x}{x}$. Не кожна команда здатна проводити таку інтерпретацію. А ось команда ***ezplot*** – може. Оскільки в (1.12) *F* введено як текст, то тепер її використання вже ***не потребує використання лапок***,

```
>> ezplot(F)
```

як то було в розділі 1.6, формула (1.4). Якщо інтерпретація *F* як функції вдалася – програма ***ezplot*** будує її графік.

Зауважимо, що не кожна команда потребує інтерпретації *F* як математичної функції. Наприклад, команді ***title***(*F*) достатньо, аби *F* була просто текстом, навіть з математичними помилками.

Таким чином, аби повідомити MATLAB, що певна величина є **текстового типу**, беремо цей текст у лапки, як (1.12).

Якщо ж ми хочемо конструювати правильні математичні вирази, маємо використати оператор ***syms***. Такі вирази (текстові дані) назвемо ***дані символьного типу***. Оголосимо, наприклад, що *x* є символьного типу,

```
>> syms x
```

Тепер будь-який правильний математичний текст, що включає *x*, буде також даним символьного типу. Приклад:

```
>> T=cos(x)
```

(спробуйте, одначе, >> *T*=*cos*(*x* або >> *T*=*cosx*).

Можна побудувати графік такої функції, не використовуючи апострофів:

```
>> ezplot(T,[-5*pi, 5*pi])
```

Можемо сконструювати скільки завгодно символьних величин із змінною *x* та працювати з ними:

```
>> F1=sin(x)/x;   F2=x*sin(x);
>> dF1=diff(F1)
    dF1 = cos(x)/x - sin(x)/x^2
>> pretty(dF1)  %Надамо результату „шкільного" вигляду:
     cos(x)      sin(x)
     --------  -  --------
       x            2
                   x
```

1.7.3. ЧИ ЗНАЄ MATLAB АЛГЕБРУ І ТРИГОНОМЕТРІЮ?

З даними символьного типу (символьними *об'єктами*), з якими ми щойно ознайомилися, MATLAB

вміє багато чого робити. Перевіримо, по-перше, його знання шкільної алгебри і тригонометрії.

1. Оголосимо *a* і *b* символьними об'єктами

```
>> syms a b
```

Чи вміє MATLAB розкласти (expand) простий алгебраїчний вираз $(a+b)^2$? Перевіримо:

```
>> Expression1=expand((a+b)^2)
   Expression1 =  a^2 + 2*a*b + b^2
```

Вміє! Спробуємо новий символьний об'єкт *Expression1* записати у більш звичному для нас, „письмовому" вигляді. Це вміє команда ***pretty***:

```
>> pretty(Expression1)
```

$$a^2 + 2\,a\,b + b^2$$

Знайома формула квадрату суми, чи не так? Зауважимо: останній вираз – не об'єкт, на відміну від *Expression1*, з ним нічого робити не можна. Команда ***pretty*** створює лише запис у вікні MATLAB!

Надамо ще приклади. Але пояснювати часто будемо за допомогою лише коментарів:

2.

```
>> %П'ятий степінь різниці:
>>Expression5 =expand((a-b)^5)
```

```
Expression5=a^5-5*a^4*b+10*a^3*b^2-10*a^2*b^3+5*a*b^4- b^5
```

```
>>%Знайома формула? Ось її запис:
>> pretty(Expression5)
```

$$a^5 - 5\,a^4\,b + 10\,a^3\,b^2 - 10\,a^2\,b^3 + 5\,a\,b^4 - b^5$$

3. А може MATLAB спростити довжелезний алгебраїчний вираз? Інколи може:

```
>> Expression3 =simplify(a^3 - a^2*b + a*b^2 - b^3)

   Expression3 = (a - b)*(a^2 + b^2)
```

І вже очікувано, що MATLAB вміє „розкривати дужки" будь-якого складного добутку:

```
>> expand((a - b^2)*(a^2 + b^2)*(a+b))
ans=a^4-a^3*b^2+a^3*b-a^2*b^3+a^2*b^2-a*b^4+a*b^3-b^5
>> pretty(ans) % У письмовому вигляді:
```

$$a^4 - a^3 b^2 + a^3 b - a^2 b^3 + a^2 b^2 - a b^4 + a^3 b - b^5$$

4. Перейдемо до тригонометрії. Якщо оголосити *Fi* символьної змінною (позначка для кута φ),

```
>> syms Fi
```

то вже знаємо, що *s1=sin*(*Fi*) та *s2=cos*(*Fi*) – також символьні об'єкти MATLAB. Перевіримо основну тригонометричну тотожність:

```
>> simplify(sin(Fi)^2+cos(Fi)^2)
   ans = 1
```

„Доведено"! Можна й інакше. Наступна команда цікава тим, що називає всі правила, які MATLAB випробовує[12]:

```
>> simple(sin(Fi)^2+cos(Fi)^2)
```

Деякі правила („combine", „factor", „collect" тощо) до такого відомого результату не привели, проте кілька інших були успішними. Цікаво, що правило „rewrite(tan)" дало навіть розкладення на тангенс половинного кута. Також:

```
>> simple(sin(Fi))  %Чи знайде перетворення до tgφ ?
```

Так, „rewrite(tan)" дало: (2*tan(*Fi*/2))/(tan(*Fi*/2)^2 + 1)

```
>> pretty((2*tan(Fi/2))/(tan(Fi/2)^2 + 1)) %Як це у запису?
```

$$\frac{2\tan\left(\frac{Fi}{2}\right)}{\tan\left(\frac{Fi}{2}\right)^2 + 1}$$

Відома зі школи формула!

5. А от формулу логарифма добутку творці MATLAB забули запрограмувати! Завдання

```
>> simple(log(a*b))
```

результату не дає! А от навпаки – все гаразд:

```
>> simple(log(a)+log(b))
```

бо кілька „правил" дають відповідь *log*(*a**b*)!

---

[12] Незнайомим з англійською: *simplify* означає „спрости", *simple* – „простий". Радимо запитати Help для цих команд!

6. Задамо MATLAB доволі складне запитання: „чому дорівнює $e^{i\varphi}$, тобто експонента від кута, помноженого на комплексну одиницю i”:

>> *simple*(*exp*(*Fi**i))

Кілька з правил дають відповідь:

cos(*Fi*) + *sin*(*Fi*) i

На підставі цього запишемо остаточно цю „доведену” формулу:

$$e^{i\varphi} = cos\varphi + i \cdot sin\varphi \qquad (1.13)$$

Це – славнозвісна формула Ейлера[13], що пов’язує тригонометрію з логарифмами за допомогою уявної одиниці $i = \sqrt{-1}$.

Питання до читача: так що ми тепер відповім на запитання у заголовку даного розділу?

1.7.4. ЧИ ЗНАЄ MATLAB ВИЩУ МАТЕМАТИКУ?

Диференціювання MATLAB знає, як показано у розділі 1.7.1. Покажемо застосування символьної математики до інтегрування та інших задач вищої математики. Вони – не лише для першокурсників, та стануть у нагоді й магістрам, і науковцям.

Приклад 1.5. *Знайти похідну функції* $y = \left(\frac{x}{1+x}\right)^x$

(*№ 663* із збірника задач Бермана [32]*).*

Приклад 1.6. *Знайти первісну функції* $y = \frac{1}{1+\sqrt{x+1}}$.

(*№ 1869* [32]).

Утворюємо два символьних об‘єкти. Їхні імена мають нагадувати походження задач:

>> *syms x* %Тобто, все наступне – символьні об’єкти:

>> *Berman663*=(*x*/(*1*+*x*))^*x*; *Berman1869*=*1*/(*1*+*sqrt*(*1*+*x*));

Знаходимо похідну

[13] Див. https://uk.wikipedia.org/wiki/, стаття "Формула Ейлера".

>> d663=diff(*Berman663*)

d663=(*x*/(*1+x*))^*x**(*log(x*/(*1+x*))+(*1*/(*1+x*)-*x/*(*1+x*)^2)*(*1+x*))

і невизначений інтеграл:

>> *int1869=int*(*Berman1869*)

int1869=-*log*(*x*)+2*(*1+x*)^(*1/2*)+*log*(-*1*+(*1+x*)^(*1/2*))-*log*(*1*+(*1+x*)^(*1/2*))

(Нововведені імена *d663* і *int1869* нагадують одночасно зв΄язок з диференціюванням і інтегруванням, та походження задач із збірника Бермана [32]).

Отримані вирази записано „у рядочок”. Слід переглянути їх за допомогою команди ***pretty***:

>> *pretty*(*d663*)

```
    / x  \ /  x  \x    /  x        1    \ /  x  \x - 1
log| ----- | | ----- |  - x | -------- - ----- | | ----- |
    \x + 1/ \x + 1/     |     2   x + 1 | \x+1 /
                        \  (x + 1)      /
```

>> pretty(int1869)

```
         1/2                1/2
2 (x + 1)    - 2 log((x + 1)    + 1)
```

Можна бачити, що одержані вирази дещо відрізняються від відповідей підручника:

$$d663 = \left(\frac{x}{x+1}\right)^x \left(\frac{1}{x+1} + \ln\frac{x}{x+1}\right),$$

$$\mathrm{int}1869 = 2\left(\sqrt{x+1} - \ln(1+\sqrt{x+1})\right) + C.$$

Проте, у кожному випадку наші результати тотожні відповідям. Можна спробувати доручити MATLAB самому знайти і запропонувати нам тотожні вирази. Так, команда *simple*(*d663*) видасть певний перелік можливих тотожних записів для похідної *d663*, кожний з яких використовує ті чи інші запрограмовані правила алгебри, тригонометрії, математичного аналізу тощо. Та не всі правила вдалося запрограмувати фірмі *MathWorks*… Тотожність двох результатів можна довести „руками та головою”, та не вдається (поки що?) довести комп’ютером...

Приклад 1.7. *Обчислити визначений інтеграл* $\int_0^\infty e^{-x^2} dx$ (інтеграл Гаусса).

Наводимо команди і відповіді MATLAB на них:

```
>> syms t k                                          (1.14)
 >> Gauss = int (exp(-t^2),0,inf)
        Gauss = 1/2*pi^(1/2)
 >> pretty(Gauss)
                    1/2
                1/2 pi
```

що є відомим результатом $\frac{1}{2}\sqrt{\pi}$. (Чи це не диво? Звідки тут зв'язок з числом π, відношенням довжини круга до діаметра!?)

Приклад 1.8. Відомо, що не від кожної функції „*можна взяти інтеграл*". Точніше висловитись так: є такі аналітично задані функції, для яких інтеграли **існують, але *не можуть бути виражені*** через відомі нам елементарні функції $\sin x$, $\cos x$, $\ln x$, x^α тощо. Ось відомі приклади:

$$\int \frac{\sin x}{x} dx \qquad \text{і} \qquad \int e^{-x^2} d\,x\,. \qquad (1.15)$$

І якщо вони *існують*, і конче потрібні у практиці, математики сказали: надаємо кожній функції власну назву і створюємо для кожної алгоритм розрахунку (а коли ще ЕОМ не було, створювали таблиці для ручного розрахунку). Першу назвали „*Інтегральний синус*" з позначенням у MATLAB ***sinint*(*x*)**, а другу назвали „*Інтеграл помилок*" з позначенням *erf*(*x*). Точніше:

$$sinint(x) \overset{Df}{=} \int_0^x \frac{\sin x}{x} dx \qquad \text{і} \qquad erf\,(x) \overset{Df}{=} \frac{2}{\sqrt{\pi}} \int_0^x e^{-x^2} d\,x$$

MATLAB вміє обчислювати ці функції, або навіть будувати їхні графіки:

```
>> help sinint
>> ezplot('sinint(x)', [-4*pi , 4*pi])      або
```

>> help erf
>> figure, *ezplot*('*erf* (*x*)', [*-4*pi* , *4*pi*])

Результати на рис. 1.6.
А як MATLAB їх розраховує практично? Адже звичний нам аналітичний розрахунок неможливий...

Ідея така: функції $y = \frac{\sin x}{x}$ і $y = e^{-x^2}$ спочатку перетворюються у таблиці їх значень у певних „вузлах" аргументу *x*, і до останніх застосовується чисельний метод, див. 2.4.

Коментар: у математиці є і інші функції, що не виражаються через елементарні, як кажуть – „*спеціальні функції математичної фізики*". Деякі ви зустрінете у лабораторній роботі 3.

1.7.5. НЕСКІНЧЕННІ СУМИ І РЯДИ В MATLAB

Приклад 1.9. *Обчислити наближено і точно суму знакозмінного ряду* $1-\frac{1}{2}+\frac{1}{3}-\frac{1}{4}+\ldots$. (1.16)

k-й доданок у даному ряді можна записати як $a_k = (-1)^{k+1}\frac{1}{k}$.

Перевіряємо: для *k*=1 маємо $a_1 = (-1)^2\frac{1}{1} = 1$ – дійсно, перший член суми; *k* =2 маємо $a_2 = (-1)^3\frac{1}{2} = -\frac{1}{2}$ – відповідає формулі; *k* =3 дає $a_3 = (-1)^4\frac{1}{3} = \frac{1}{3}$, і так далі. Наближене значення обчислюємо, взявши 5 і 10 (та будь-яку іншу скінчену кількість) доданків у частинній сумі, а точне значення – використовуючи ідентифікатор ***inf*** (від *infinity*, *нескінченність*):

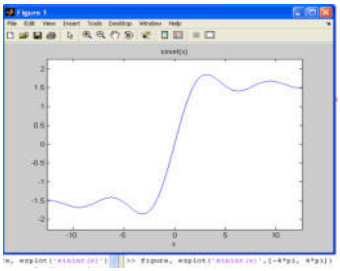
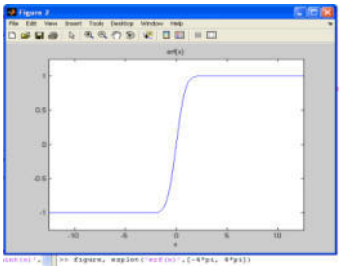



Рис. 1.6. Графіки інтегрального синусу і функції Гауса (1.15).

>>*S5=1+**symsum**((-1)^k/k, 1, 5), S10=1+symsum((-1)^k/k, 1, 10),*
S5 = 13/60
S10 = 893/2520
>> *S=1+symsum((-1)^k/k,1,inf)*
S = 1-log(2)

(Тобто ***сума нескінченої кількості*** таких доданків – ***скінчена***, та ще якимось дивним чином пов'язана з натуральним логарифмом числа 2, *ln*2!)

Таким чином, п'ять доданків дають *13/60*=0.21667, десять – *893/2520*=0.35437, а точне значення суми ряду є $1-\ln 2 \approx 0.30685$. При цьому враховано, що символьну змінну *k* попередньо було вже введено в (1.14). Значення змінних *S5*, *S10* і *S* MATLAB видав у "раціональній формі", як раціональні числа з цілими у чисельнику і знаменнику; про інші можливі формати див. *help sym*. В даному випадку слід скопіювати ці дробові значення, розмістити у командний рядочок і отримати значення у десятковій формі.

А тепер – „вільний експеримент", і подивимось на результат! Візьмемо такий-саме ряд, але усі його члени – у якомусь цілому степені *n=2*, або *n=3*, або *n=4*, і т.д.,

$$1-\frac{1}{2^n}+\frac{1}{3^n}-\frac{1}{4^n}+\frac{1}{5^n}-\frac{1}{6^n}\ldots$$

Ми вже з'ясували, чому дорівнює нескінченна сума таких доданків у випадку степені *n=1*. Дослідимо для інших.

n=2
>> *S2=1+symsum((-1)^k/k^2,1,inf)*
*S2 = 1-1/12*pi^2*

2
1 - 1/12 pi

n=4

```
>>  S4=1+symsum((-1)^k/k^4,1,inf)
S4 =   1-7/720*pi^4
>> pretty(S4)
```

$$1 - \frac{7\pi^4}{720}$$

Зрозуміло, що у цих усіх випадках спільний член виглядав як $a_k = (-1)^k \frac{1}{k^n}$. Продовжуємо:

n=3

```
>>  S3=1+symsum((-1)^k/k^3,1,inf)
    S3 =  1-3/4*zeta(3)
```

n=5

```
>>  S5=1+symsum((-1)^k/k^5,1,inf)
    S5 =  1-15/16*zeta(5)
```

А в останніх випадках що за диво, ***zeta***(*3*) чи ***zeta***(*5*)? Запит допомоги MATLAB *>> help zeta* дозволяє встановити, що це є «*Symbolic Riemann zeta function*», одна з багатьох *спеціальних* функцій як (1.15). Вона на виражається через *елементарні*, та може використовуватись поряд з ними. MATLAB обчислить її якимось чисельним методом:

```
>> S5 =  1-15/16*zeta(5)
```

Числові ряди (тобто нескінченні суми на зразок (1.16)) студенти мають вивчати на початку другого курсу університету. Та ви вже вмієте працювати з ними, хоча й без доведень☺...

Приклад 1.10. *Розкласти функцію* $y = e^{-t^2}$ *в ряд Тейлора, отримавши 10 членів розкладу.*

Результат досягається MATLAB-командою

```
>> T10=taylor(exp( - x^2,'Order',11), 11)
   T10 =- x^10/120 + x^8/24 - x^6/6 + x^4/2 - x^2 + 1
>> pretty(T10)
```

$$-\frac{x^{10}}{120}+\frac{x^{8}}{24}-\frac{x^{6}}{6}+\frac{x^{4}}{2}-x^{2}+1$$

(Символьна змінна *x* була вже створена; команді *taylor* дано завдання створити *11*, а не *10* членів ряду, бо команда сприймає таке завдання як "*не більше ніж…*". У відповіді *T10* присутні насправді 6 доданків, бо коефіцієнти при непарних степенях – нулі). ***Ряди Тейлора*** вивчають на початку другого курсу університету. Це дуже важливий математичний інструмент. Тому, ми приділяємо йому велику увагу в цій книжці. Аналогічно, велике значення мають й ***ряди Фур'є***...

1.7.6. РОЗВ'ЯЗУВАННЯ РІВНЯНЬ В MATLAB

Розглянемо можливості аналітичного розв'язування трансцендентних і диференціальних рівнянь від однієї невідомої.

Приклад 1.11. *Розв'язати систему лінійних алгебраїчних рівнянь* (СЛАР) *другого порядку*

$$a_1x+b_1y=c_1,\ \ a_2x+b_2y=c_2.$$

Оголосимо спочатку коефіцієнти та невідомі символьними об'єктами:

```
>> syms x y a1 a2 b1 b2 c1 c2
```

Використаємо команду ***solve*** та назвемо розв'язок ім'ям *Sol* (від англійського *solution*, розв'язок):

```
>> Sol=solve(a1*x+b1*y==c1, a2*x+b2*y==c2)
```

(Важливо, що використовується „==", а не „="!). Отримали щось не дуже зрозуміле:

```
Sol =
    x: [1x1 sym]
    y: [1x1 sym]
```

Тут використовується ще невідомий вам тип даних ***struct***. Докладно про нього пізніше, а зараз без пояснень. Цей тип даних має два ***поля*** – *x* та *y*. Отримати замість них обидва розв'язки можна так:

```
>> X=Sol.x,  Y=Sol.y
```

Маємо дві формули; перша з них така:

X = – (*b1***c2* - *b2***c1*)/(*a1***b2* - *a2***b1*)
>> pretty(X)

$$-\frac{b1\ c2 - b2\ c1}{a1\ b2 - a2\ b1},$$

тобто розв'язок пов'язаний з заданими коефіцієнтами через дріб. Такого роду формули отримували ще у школі. Так само можна отримувати ***формули*** для розв'язку СЛАР третього та вищих порядків.

Ми знайшли аналітичний розв'язок, формули для будь-яких значень коефіцієнтів. Якщо ж їхні чисельні значення відомі, за тім же самим алгоритмом MATLAB знайде числовий розв'язок.

Приклад 1.11а. *Розв'язати систему лінійних алгебраїчних рівнянь* (СЛАР) *другого порядку*

$$2x - y = 3,\quad x + 2y = -4.$$

У школі ми *підставляли* (*substitute*) числові коефіцієнти або в отримані формули розв'язку, або в рівняння та розв'язували їх тим самим алгоритмом. Підемо останнім шляхом:

>>*Eq1=a1*x+b1*y-c1*;*Eq2=a2*x+b2*y-c2*; %Символьні об'єкти
>> *Eq1n=**subs**(a1*x+b1*y-c1*,[*a1*,*b1*,*c1*],[2,-1,3]) %Підстановка
Eq1n = 2*x - y - 3
>> *Eq2n=subs*(*a2*x+b2*y-c2*,[*a2*,*b2*,*c2*],[1,2,-4]);
>> [*Xn*, *Yn*]=*solve*(*Eq1n*, *Eq2n*)
Xn = 2/5 *Yn* = -11/5

Отримано розв'язок у вигляді раціональних чисел.

Як відомо, розв'язування СЛАР пов'язане з обчисленням визначників (метод Крамера, див. 2.1).

Приклад 1.11б. *Обчислити визначник* $\begin{vmatrix} a_1 & b_1 \\ a_2 & b_2 \end{vmatrix}$.

Оскільки всі потрібні символьні об'єкти оголошені, безпосередньо запускаємо символьну математику MATLAB:

>> *D=**det***([*a1* a2; *b1 b2*])

Отримуємо знайомий вираз.

Приклад 1.12. *Розв'язати рівняння* $\arcsin x = p(1-x)$ *для* $p > 0$.

Ліва частина цього рівняння є тригонометрична функція, а права – алгебраїчна лінійна. З причини такого „змішування", рівняння належить до класу ***трансцендентних***. Відомо, що це для трансцендентних рівнянь *аналітичного* розв'язку не існує. Та розв'язок взагалі існувати може! Більше про такі рівняння – у розділі 2.2. „Знаючи" це, MATLAB швидко знаходить чисельні розв'язки для визначеного наперед значення параметра p. Використовуємо ту ж команду ***solve*** (*розв'язати*):

```
>> % Для p=0.5 :
>> X1=solve('asin(x) = .5*(1-x)')
X = . 32916
>> % Для p= 4 :
>> X2=solve('asin(x) - 4*(1-x)')
X = . 77738
>> syms x p  %немає розв'язку для довільного p:
>> X1=solve(asin(x) == p*(1-x))
Warning: Explicit solution could not be found.
        X1 = [ empty sym ]
>> X1=solve( subs( 'asin(x) = p*(1-x)' ,p,4)) %тепер p=4
        X1 = 0.77737822981390499749724206735112
>> %%перевіримо, чи це дійсно корінь?
>> subs( 'asin(x) - p*(1-x)' ,[p,x],[4, X1]) %"-" замість "="!
ans = 2.87816302049983092660105858952 27e-34
```

Отримане число 2,878..e^{-34} дуже мале, „машинний нуль", тобто дане значення *X1* „обертає" функцію на нуль, дійсно є розв'язком.

Зверніть увагу: рівняння можна писати у різних формах; використовувати лапки – якщо *x* **не** була оголошена як символьна змінна, та не використовувати лапки, але тоді *>>syms x* – обов'язково.

Спробуємо тепер новий *сольвер* (так тепер називають програми розв'язування рівнянь) – команду ***fzero*** (від *find zero*). На відміну від попередньої, тут 1) не можна задавати рівняння через „=", а треба писати ліву частину, яку треба „обернути у нуль", та 2) додатково слід задавати число – *перше наближення* до розв'язку. Спробуємо:

Приклад 1.13. *Розв'язати рівняння* $\cos x = 0{,}2x$.

```
>> X=fzero( 'cos(x) - .2*x ', 0)  %Початкове наближення=0
                X =   1.3064
>> X=fzero('cos(x) - .2*x',-1) %Початкове наближення=-1
                X =   -1.9774
>> X=fzero('cos(x) - .2*x',-3) %Початкове наближення=-3
                X =   -3.8375
```

Кожне з отриманих значень слід перевірити – чи воно дійсно корінь? На прикладі останнього пересвідчимось у цьому:

```
>> subs('cos(x) - .2*x',X)
          ans =  -2.2204e-16
```

Отримали „машинний нуль" 2,22..e^{-16}, тобто X =-3.8375 дійсно є коренем рівняння, що досліджується.

Приклад 1.14. *Розв'язати диференціальне рівняння* $y'''+9y'=0$ [32], *№4301*. (Хоча таку тему вивчають на другому році навчання, та для фізики потреба може виникнути раніше!).

Диференціальні рівняння і навіть системи диференціальних рівнянь запрограмоване розв'язувати командою ***dsolve***. В її аргументах перелічують рівняння системи і граничні умови (кожне – в лапках, через кому)**:**

```
>> Berman4301=dsolve(' D3y+9*Dy=0 ')
     Berman4301= C1+C2*sin(3*t)+C3*cos(3*t)
```

Тобто рівняння записують в певній умовній формі: *Dy* означає першу похідну і *D3y* третю похідну функції *y*(*t*), аргументом вважають "час" *t*, якщо інше не вказано (див. нижче). Можна пересвідчитись у правильності отриманого розв'язку, який включає три довільні константи. А ось як виглядали б завдання і розв'язок за наявністю граничних умов $y(0)=1,\quad y'(0)=1,\quad y(10)=1$**:**

```
>>Berm4301=dsolve('D3y+9*Dy=0','y(0)=1','Dy(0)=1','y(10)=1')
     Berm4301=1/3*(3*cos(30)+sin(30)-3)/(-1+cos(30)) –
                    1/3*sin(30)/(-1+cos(30))*cos(3* t)+1/3*sin(3*t)
```

Щоб полегшити прочитання розв'язку, застосуємо знайому вже команду:

```
>> pretty(Berman4301)
```

```
sin(3 t)    3 cos(30) + sin(30) - 3   cos(3 t) sin(30)
--------- + ------------------------- - ----------------
   3                3 (cos(30) - 1)         3 (cos(30) - 1)
```

Бачимо, що MATLAB знайшов константи *C1*, *C2*, *C3* ніби користуючись відомими з вищої математики методами. Для побудови графіку розв'язку слід скористатись отриманим виразом для *Berm4301*, тобто

>> ezplot(*Berm4301*,[0 2**pi*])

Однак, у даній версії MATLAB поки ще не кожне рівняння або система рівнянь можуть бути розв'язані аналітичною математикою MATLAB, не все зуміли запрограмувати програмісти фірми MathWorks. Може, ви доповните? Оскільки готового результату вам не гарантовано – будьте готові трохи експериментувати з пакетом!

У цьому розділі надано лише основну інформацію про розв'язування рівнянь у MATLAB. Тут ми робимо наголос на тому, аби розрізняти *аналітичні* та *чисельні* обчислення, бо вони потребують зовсім іншого програмування у MATLAB.

1.7.7. МНОГОЧЛЕНИ В MATLAB

Різницю між числовими і символьними об'єктами MATLAB, принципову різницю між аналітичними та чисельними алгоритмами (діями) покажемо також на прикладі многочленів, які можна розглядати і опрацьовувати як з одної, так і з іншої точки зору.

1.7.7.1. Многочлени як числові об'єкти

З векторами $p=[a_1,a_2,...,a_n]$, де a_i – числа, ми вже працювали – множили на скаляр або на матрицю вимірів n*1, будували графік командою *plot*. Та певні програми MATLAB сприймають цей числовий об'єкт зовсім іншим чином. Вони так створені, що числовий вектор p представляє многочлен у формі, відомій ще зі школи

$$p(x) = a_1 x^n + a_2 x^{n-1} + a_3 x^{n-2} + ... + a_n x + a_{n+1} \qquad (1.17)$$

(елементи вектора відповідають коефіцієнтам многочлена у порядку спадання степенів; замість слова „многочлен" використовують також синоніми „багаточлен", „поліном"; це – многочлен степеню n, бо вважається $a_n \neq 0$). Так, команда ***polyval***(*p,x*$_0$) (від ***poly****nomial* ***val****ue*) обчислює цей многочлен в точці x_0, тобто отримує $p(x_0)$.

Команда (програма) ***roots***(*p*) знаходить всі корені[14] многочлена $p(x)$. Якщо, навпаки, задати вектор-стовпчик з коренів многочлена *r*, то команда ***poly*** знаходить вектор-рядок коефіцієнтів полінома. Тобто, команди ***roots*** і ***poly*** – взаємо обернені. Наведемо приклади:

1. Нехай дано многочлен $P(x) = x^5 + x^4 - x^2 + 3x - 2$. Його можна представити у MATLAB як числовий вектор

```
>> p5=[1, 1, 0, -1, 3, -2];
```

(Давайте домовимось про використання імен. Многочлени позначаємо латинськими літерами *P*, *Q*, *R*, *T*, *p*, *q*, *r*. Використовуємо великі літери, якщо мова йде про многочлен у символьному представленні, та маленькі літери – якщо у чисельному представленні, як вектор. Числовий індекс за літерою свідчить про степінь полінома – п'ятий степінь у даному випадку.)

Знайдемо, чому дорівнює цей багаточлен *P*5, коли *x*=0:

```
>> x=0;   p5_0=polyval(p5 , x)
        p5_0 =    -2
```

А тепер знайдемо корені многочлена – назвемо їх *Korni*:

```
>> Korni=roots(p5)
        Korni =
          -1.3478 + 1.0289i
          -1.3478 - 1.0289i
           0.5000 + 0.8660i
           0.5000 - 0.8660i
           0.6956
```

У даного многочлена 5 коренів, з них 4 комплексних і один дійсний. Скільки має бути коренів і яких саме – ці важливі

[14] Пригадайте визначення, що таке „корінь многочлена"! Як можна „корінь" пов'язати з командою ***polyval***(*p,x*$_0$)?

питання див. нижче, а зараз вчимося лише працювати з многочленами, поданими у числовій формі.

Перевіремо, чи це справді корені. Наприклад, другий з них:

>> *polyval*(*p*5 , *Korni*(2))

ans = -3.5083e-14 - 7.5495e-15i

Оскільки значення полінома складається з дійсної частини порядку 10^{-14} і уявної частини порядка 10^{-15}, і ці числа дуже малі, робимо висновок, що це й насправді корінь з точністю, з якою комп'ютер обраховує. Так само й інші *Korni*(*i*), *i*=1,..,5.

2. Розглянемо протилежну задачу: задано числа, наприклад -3, 1, 4; знайти многочлен, що має такі корені.

У школі таку задачу розв'язували наступним чином. Це – многочлен третього (скільки коренів!) степеня. Тому[15]:

$$P3(x)=(x+3)(x-1)(x-4)= x^3 -2x^2-11x+12.$$

Та ось як вона розв'язується з MATLAB командою poly:

```
>> roots3=[-3 , 1 , 4];
>> p3=poly(roots3)
    p3 =    1    -2   -11    12
```

– співпадає з „ручним" розрахунком! Перевіримо цей результат:

```
>> Korni3=roots(p3)
        Korni3 =   -3.0000
                    4.0000
                    1.0000
```

– тобто такі корені, які „закладено у розрахунок"!

Зауваження: задача знаходження многочлену за коренями має нескінченну кількість розв'язків. Проте, усі можливі многочлени еквівалентні; один можна отримати з іншого множенням на число. Довести можете?

3. Питання: чи можна „розкриття дужечок", яке робилося для отримання многочлена *P*3, зробити у MATLAB? Тут у нагоді команда ***conv***, яка також працює з многочленами у числовій формі. Приклад:

[15] Аби кожне з них обертало поліном в нуль.

Многочлени першого степеня, що складають *P*3, назвемо p1, r1 і q1. Їх вже знаємо як отримати:

>>*p1*=*poly*(*roots*3(1)); *r1*=*poly*(r*oots*3(2)); *q1*=*poly*(*roots*3(3));

Тепер отримаємо многочлен, що дорівнює добутку першого й другого:

>> *p*2=*conv*(*p*1, *r*1);

І нарешті, маємо добуток *p*2 і *q*1, остаточний многочлен:

>> *p*3=*conv*(*p*2,*q*1)

*p*3 = 1 -2 -11 12

Той самий многочлен можна було б отримати й однією дією, а саме:

>> *p*3=*conv*(*conv*(*p*1, *r*1), *q*1)

p3 = 1 -2 -11 12

Це були дуже прості приклади. Але вони дозволяють розв'язувати у MATLAB дуже складні задачі, що часто-густо виникають у математиці й прикладних науках! Це були *обчислювальні* команди і алгоритми. Проте, добре було б діяти і за звичними нам правилами алгебри.

1.7.7.2. Многочлени як символьні об'єкти

Для цього перш за все слід оголосити аргумент *x* ***символьним об'єктом***, аби MATLAB не наполягав призначати йому якесь певне значення. Це можна зробити командами ***sym*** або ***syms*** у форматі

>> *syms x*; *% оголошуємо x символьною змінною*

>> *P5=x^5+x^4-x^2+3*x-2 %Тепер P також є символьною*

*P5 = x^5+x^4-x^2+3*x-2*

Або:

>> *syms x a1 b1 c1*

>> *P= a1*x^2+b1*x+c1*;

Тепер MATLAB не вважатиме коефіцієнти *a1*, *b1*, *c1* і змінну *x* невизначеними величинами і сприймає *P* як якийсь цілісний об'єкт. Можна замовити MATLAB показати цей об'єкт у більш „*приємному*" (*pretty*) вигляді:

>> *pretty*(*P*)

$$a1\,x^{2} + b1\,x + c1,$$

тобто звичний нам запис многочлена!

Так само введемо об'єкт *Q= a2*x^2+b2*x+c2* – інший символічний многочлен другого степеня. Спробуємо помножити ці многочлени:

```
>> syms a2  b2  c2
>>  Q= a2*x^2+b2*x+c2
>> PQ=P*Q
```

(*PQ* – це лише нововведена назва для многочлена-добутку). Відповідь MATLAB:

PQ = (*a1*x^2+b1*x+c1*)*(*a2*x^2+b2*x+c2*).

Він просто замінив *P* і *Q* на їх символічні значення (вирази), тобто працює з цілісними об'єктами, утворивши символьний об'єкт *PQ*, ще „не знаючи", що з останнім можна поводитись за правилами алгебри і розкрити дужки. Команда *expand* (*розкласти*) роз'яснює йому це наше бажання:

```
>> PQ=expand(PQ)
    PQ =a1*x^4*a2+a1*x^3*b2+a1*x^2*c2+b1*x^3*a2+
            b1*x^2*b2+ b1*x*c2+c1*a2*x^2+c1*b2*x+c1*c2
```

Як бачимо, MATLAB виконав множення, але ще не „розуміє", що в результаті, який отримано, можна привести подібні (з однаковими степенями від *x*) члени. Підкажемо йому далі командою ***collect*** (*збери*):

```
   >> PQ=collect(PQ)
PQ=a1*x^4*a2+(a1*b2+b1*a2)*x^3+(a1*c2+b1*b2+c1*a2)*x^
                2+ (b1*c2+c1*b2)*x+c1*c2
```

Нарешті отримали символічний вираз *PQ*, що представляє добуток двох многочленів:

```
>> pretty(PQ)
                4                3                2
 a1 a2 x  + (a1 b2 + a2 b1) x  + (a1 c2 + a2 c1 + b1 b2) x  +
                                          (b1 c2 + b2 c1) x + c1 c2
```

Зрозуміло, такий результат дійсний для будь-яких коефіцієнтів *a1*, *a2*, *b1*, *b2*, *c1*, *c2*, як нас вчили у школі.

Зауважимо: команда ***conv*** тут не працює, бо вона створена для многочленів у числовій формі, а не у символьній!

Багато корисного можна одержати, володіючи символьною математикою MATLAB. Припустимо, треба

пригадати, чому дорівнюють корені квадратного рівняння, які назвемо *Roots* (не плутати цю нашу назву з однойменною командою ***roots***!). Допоможе команда *solve*(*P*) (*розв'язати* рівняння *P=0*). Отримуємо, що масив розв'язків *roots* складається з двох елементів:

```
>> Roots=solve(P)
        Roots =  [ 1/2/a1*(-b1 +(b1^2 -4*a1*c1)^(1/2))]
                 [ 1/2/a1*(-b1 -(b1^2 -4*a1*c1)^(1/2))]
```

(Можна скомандувати й в іншому форматі
>> *Roots=solve*(*P*)).
Візьмемо з нього перший елемент

```
>> root1= Roots(1)
        root1 = 1/2/a1*(-b1+(b1^2 -4*a1*c1)^(1/2))
```

і використаємо команду запису в більш звичній нотації:
>> *pretty*(*root1*)
Впізнаєте відому формулу розв'язку квадратного рівняння:

$$\frac{-b1 + (b1^2 - 4\,a1\,c1)^{1/2}}{a1} \quad ?$$

1.7.7.3. Перехід з одного представлення в інше

Часто потрібно переходити від символьного задання многочленів до чисельного. Для цього слід використовувати команди ***sym2poly*** і ***poly2sym*** (хто звик до американської мови, той зрозуміє зміст роздільника ***2***: *to*, „*перетворити у*", тобто ***sym***-об'єкт, символьний, перетворити у ***poly***-об'єкт, тобто поліном у чисельному представленні, чи навпаки). Так і є, перша команда перетворює символьний многочлен у числовий, наступна – навпаки, поліном як чисельний об'єкт перетворює в символьний.

Увага: Перетворювати поліномами з нечисловими коефіцієнтами неможливо! Пояснимо вищесказане на прикладах.

Раніше отримано *PQ* – многочлен у символьній формі з довільними (символічними) коефіцієнтами *a1*, *b1*,*c1* і *a2*, *b2*, *c2*. Чи можна перетворити його у чисельний об'єкт? Ні,

неможливо, бо чисельні значення сталих *a1*, *b1* і *c1* та інших не визначено!

Надамо спочатку першій групі, а потім другій групі коефіцієнтів певних значень і підставимо (***subs**titute*) їх у многочлен:

```
>>  a1=1; b1=2; c1=3; PQ2 =subs(PQ)
                    PQ2 =
    x^4*a2+(b2+2*a2)*x^3+(c2+2*b2+3*a2)*x^2+
                                        (2*c2+ 3*b2)*x+3*c2
>> a2=2; b2=1;c2=3;   PQ1=subs(PQ2)
                    PQ1 = 2*x^4+5*x^3+11*x^2+9*x+9
```

Перша команда з символьного многочлена-добутку *PQ* створила символьний многочлен *PQ2*, що залежить лише від другої групи коефіцієнтів, а друга команда підставила числові значення для цієї групи і створила многочлен *PQ1* вже з числовими коефіцієнтами. Однак, *PQ1* – все ще символьний об‘єкт, що залежить від *x*. Знайдемо його чисельний аналог командою ***sym2poly***:

```
>> pq1=sym2poly(PQ1)
      pq1 =    2    5   11    9    9
```

Отримано вектор *pq1*, складений з коефіцієнтів многочлена. Команда

PQ1=poly2sym(*pq1*),

навпаки, перетворить цей числовий вектор в символьний об’єкт.

1.7.7.4. ПІДСУМОК

Для подальшого нам потрібно, аби ви вільно володіли як символьними, так і чисельними типами даних (до речі, ї усіма попередніми також! Пам’ятаєте раніше подані типи даних?). Розглянемо в цілому, що ми знаємо тепер про многочлени.

1. Вище був вами створений символьний об’єкт *P5*, поліном п’ятого степеня. Побудуйте його графік.

Найкраще підходить команда , що працює саме з символьними об’єктами:

```
>> ezplot(P5)
```

Отримали графік рис. 1.7,*а*. (Трохи відійдемо від теми та зробимо важливе для студентів **зауваження**. Не все те добре, що зроблено автоматично. Цей рисунок, зокрема, створює враження, що будь які x у діапазоні між -2 та 2 – нулі функції. Причина полягає у тому, що дивимось на функцію біля осі Ox „з висоти” $y \propto 6\,000$. Такі результати здавати викладачеві не радимо. Краще поглянути на графік з „нижчої висоти зору”, >> *ezplot*(*P5*, [-2 2]). Рис. 1.7,*б* показує, що функція *P5*(*x*) доволі сильно змінюється біля осі та має лише одну точку перетину, корінь).

2. Таку саме задачу поставимо, але вимагатиме використати *чисельну* побудову графіку за допомогою команди . Ось як це зробити:

```
>> p5=sym2poly(P5); %чисельний поліном
>> t=-2 : .01 : 2; %чисельні значення аргументу
>> y=polyval(p5,t); %чисельні значення поліному в точках t
>> figure, plot(t,y)
```

Графік рис. 1.7,*б* побудовано. Нагадаємо, що й такий шлях є можливим:

```
>>  y=subs(P5, x, t); figure, plot(t, y)
```

Такими є початкові кроки з символьною математикою. Таким чином, перш ніж її використовувати, в MATLAB треба створити символьні об‘єкти. Цьому служать команди *sym* і *syms*.

Засвоїти глибше роботу з ними можна за допомогою лабораторних робіт № 3 і № 4.

1.8. КОНТРОЛЬНІ ПИТАННЯ ДО РОЗДІЛУ 1

1. Які "вікна" MATLAB ви знаєте? Яке з них, власне, потрібне для роботи? Як викликати вікно Help та як його використовувати?
2. Як викликати довідку про побудову графіків?

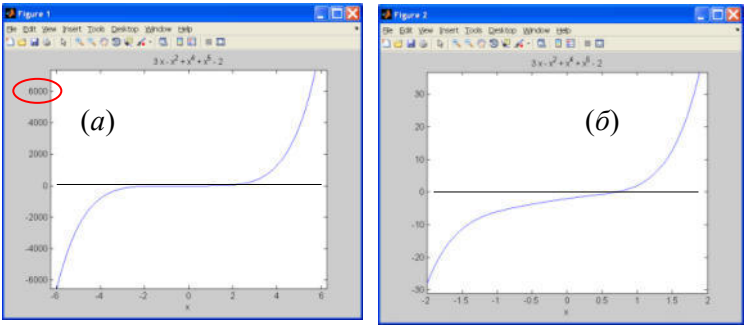


Рис. 1.7. Графіки поліному P5(x): (*а*) діапазон обрано автоматично; (*б*) діапазон взято $-2 \le x \le 2$, аби краще роздивитися поведінку кривої поблизу осі *Ox*.

3. Для чого в MATLAB призначені коментарі, як вони вводяться?
4. Які використовуються формати для дійсних і комплексних чисел? Як вводяться масиви чисел – вектори і матриці?
5. Які імена можна надавати змінним в MATLAB? Які НЕ можна? Чи можна надавати імена *ans*, *pi*, *i*, *j*? Як надавати змінним ті чи інші значення?
6. Які ви знаєте дії із змінними (з величинами) в MATLAB? За яких умов дії **+**, –, *****, **/** і **^** можна застосовувати до матриць? Чи можна застосувати операції **^** і *sqrt*, *sin*, і такі інші до матриць? Чим відрізняються деякі з названих операцій від їх відповідників "з крапкою" **.***, **./** і **.^**? Як отримати довідку про арифметичні оператори?
7. Як можна отримати значення тригонометричних функцій від заданих аргументів? Експоненти і логарифма (натурального, десяткового, з довільною основою)? Гіперболічних функцій? Чи може бути аргументом названих функцій вектор або матриця? А комплексне число?
8. Як відокремлювати команди у командному вікні MATLAB, якщо їх декілька? Яка різниця між роздільниками "**,**" (кома) і "**;**" (крапка з комою)?
9. Як можна повторити команду, попередньо вже виконану?

10. Яке призначення функцій від масивів *length*, *size*, *inv*, *norm* і *diag*?
11. Що означає команда MATLAB типу *x=-pi:pi/100:3*pi*? Навіщо вона потрібна для побудови графіків функцій? А така команда *x=-pi : 3*pi* ? Який тут буде крок?
12. Які команди для побудови графіків функцій ви знаєте? Чим відрізняються команди *plot*, *fplot*, *ezplot* і *comet*?
13. Побудуйте графік функції $y = \sin x + 0{,}5\sin 2x - 0{,}3\sin 3x$ у різних вікнах за допомогою команд *ezplot* і *plot*.
14. Яким чином команди *legend*, *title*, *grid*, *xlabel*, *ylabel*, *axis* "прикрашають" графіки? Як можна задати або змінити колір кривих і позначки на них? Як „включити" координатну сітку?
15. Для чого використовують команду *figure*? Як кілька графіків побудувати в одному вікні?
16. Нехай *a* – дуже великий вектор-стовпчик. Отримайте вектор-рядочок *a*1, що складається з останнього елемента *a*, передостаннього, і т.д. до першого?
17. Нехай матриця (масив) *A* дуже велика. Як отримати матрицю *A*1, що є частиною *A*, а саме: включені лише кожний третій стовпчик? Кожний парний рядочок і непарний стовпчик?
18. Нехай числовий вектор *b*, що збережений у MATLAB, дуже великий. Як визначити кількість елементів у ньому? Як отримати з нього новий вектор з елементами від останнього до першого?
19. Маємо двовимірну матрицю M і вектор-рядочок *b*. Як в цій матриці замінити числа 3-го рядочка на цей вектор? За яких умов це можливо?
20. За яких умов можливі дії +, *, –, ^, / з двома матрицями *A* і *B*?
21. За яких умов можливі дії **.***, **.^**, **./** з двома матрицями *A* і *B*?
22. Як ви можете пояснити різницю між числовими і символьними об'єктами MATLAB? Як знайти корені многочлена? Скільки їх має бути?
23. Дано многочлен $P_3(x)= x^3- 3x^2- x+3$. Знайти його корені та побудувати графік цієї функції на проміжку [-2, 4],

користуючись командами MATLAB, що розуміють лише чисельні дані.

24. Розв'язати попередню задачу, користуючись командами лише командами MATLAB, що розуміють символьні дані.
25. Дано многочлен третього степеня у загальному вигляді $P_3(x)= ax^3+ bx^2+cx+d$. Чи можна знайти його корені? Яким з методів, чисельним чи аналітичним, діяти? Скільки в нього коренів? А якщо многочлен другого степеня, четвертого, п'ятого, шостого? Чи є тут якісь обмеження? Як побудувати графік того чи іншого многочлена?
26. А якщо попередню задачу перевірити для коефіцієнтів із заданими числовими значеннями – як знаходити корені? Скільки їх слід очікувати? Як побудувати графік?
27. Як знайти похідну і первісну заданої функції? Як обчислити визначений інтеграл? Суму ряду?
28. Дайте визначення, що таке „символьні дані" („аналітичні")? Які ще знаєте „типи даних", чим вони відрізняються?
29. Чи є матриці в MATLAB „типом даних"? Чому? Що саме пов'язане з поняттям „тип даних"?
30. Яка з двох команд у командному рядку є правильною, яка неправильною (поясніть):
    (1) >> *ezplot*(' *sin*(*x*)**x* ') (2) >> *ezplot*(*sin*(*x*)**x*) ?
31. Як можна відобразити у Word-документі результати вашої роботи з програмою MATLAB – введені команди і отримані результати, побудовані графіки?

Якщо не змогли відповісти на більшість питань – радимо проробити Розділ 1 з початку

2. "РУЧНА" АЛГОРИТМІЗАЦІЯ ТИПОВИХ МАТЕМАТИЧНИХ ЗАДАЧ

В даному розділі зосередимось на ***алгоритмах*** розв'язування найбільш поширених задач математики. Як правило, це будуть *чисельні* алгоритми, і ми вже знаємо, чим вони відрізняються від *аналітичних* методів: вони оперують з числовим об'єктами; конечним числом певних дій можуть досягти лише наближеного розв'язку. Головне питання тут – як, якими методами, розв'язуються ті чи інші задачі з переліку основних задач першого курсу університету. Тому, якщо метою студента є не математика, а навчитися програмуванню – цей розділ у першому читанні можна пропустити, і відразу переходити до наступного. Та ми не радимо цього робити, бо багато з задач програмування спираються на такий матеріал.

Універсальний і потужний комп'ютерний пакет MATLAB використовується тут лише як швидкий і розвинений калькулятор, а головну увагу звернено на зміст математичних методів, що розглядаються. Відібрано методи обчислень, які найчастіше використовуються в типових задачах математичного моделювання, таких як системи лінійних алгебраїчних рівнянь (СЛАР), нелінійні рівняння з одним невідомим, знаходження емпіричних формул методом найменших квадратів, диференціювання та інтегрування функцій. Матеріал викладено у короткій, довідковій формі з посиланням на більш ґрунтовну літературу. Студенту важливо володіти ідеєю метода, розуміти коректність кожної даної задачі – за яких умов гарантовані існування та єдиність її розв'язку. Бо якщо в математичній моделі немає розв'язку – як таке зрозуміти фізично? Якщо ж розв'язків декілька – який з них використовувати в практиці? Важливо також знати умови збіжності методу, способи перевірки достовірності і точності результату.

"Ручне" обчислення цих задач доволі громіздке і тривале. Проте, за допомогою MATLAB воно стає швидким і необтяжливим. Студент орієнтований на те, аби всі дії проводити за єдиним ***алгоритмом***. Таке виконання

обчислень має підготувати вас до програмування тих самих задач засобами MATLAB, чому присвячений наступний розділ 3.

Так, для всіх цих методів у MATLAB існують внутрішні, власні команди. Саме вони мають застосовуватись у Ваших курсових роботах, у дипломній, та у Вашій наступній професійній роботі. Дехто запитає "А чому не можна ці „остаточні" методи MATLAB вивчати відразу? Навіщо вчити, припустимо, метод Гауса, якщо в MATLAB є готова команда, що його реалізує?"

Якщо коротко, то можна відповісти: з тієї самої причини, чому ми навчаємо множенню і діленню чисел незважаючи на наявність у студента (навіть у школяра) зручного кишенькового калькулятора.

Бездумне використання математичних пакетів може призвести до „мавпячої хвороби": можна не вчити визначення диференціювання, якщо знаходимо потрібне командою *diff*(*y*); можна не запам'ятовувати правил обчислення визначників, якщо домашні завдання виконувати командою *det*(*A*), тощо. Та зрозуміло, що фахівцем своєї справи так не стати! Бо фахівець розуміє, що він робить.

З іншого боку, чисельна математика є джерелом задач для науки програмування, яка починається із розділу 3.

2.1. РОЗВ′ЯЗУВАННЯ СИСТЕМ ЛІНІЙНИХ АЛГЕБРАЇЧНИХ РІВНЯНЬ

Системи лінійних алгебраїчних рівнянь – це найпростіша, але ж найбільш поширена з математичних задач. Ця задача має важливе самостійне значення, оскільки до неї безпосередньо приводять багато практичних моделей фізичних, технічних та економічних явищ. Крім того, до таких рівнянь та необхідності їхнього розв'язування призводять задачі більш складні. Системи лінійних алгебраїчних рівнянь (скорочено СЛАР) – фундамент обчислювальної математики.

Студенти інколи запитують, „чи всі математичні задачі розв'язані наукою?" На таке питання можна відповісти: ні, лише ***лінійні задачі*** розв'язані усі та до кінця, лише у лінійних проблемах не лишилося білих плям. Деякі нелінійні задачі також розв'язані до кінця, та їх небагато (приклад – многочлени, розділ 1.7.7). Та переважна більшість задач є ***нелінійними***, і для них більше невідомого, аніж відомого...

Будь-яка математична операція f, що може бути застосована до елементів x, y,… якоїсь множини, називається ***лінійною***, якщо результатом застосування f до $x+y$ та до Cx (де C стала) буде відповідно $f(x)+f(y)$ та $C\,f(x)$. З курсу лінійної алгебри [15,16,34-36,41] відомо, що множення матриці A вимірності (m,n) на вектор-стовпець x вимірності (n,k) є саме лінійною операцією, результатом якої є нова матриця y вимірності (m,k). *Матричним рівнянням* називають запис

$$Ax=b, \tag{2.1}$$

де A матриця (m,n), b – деякий *n*-вимірний вектор-стовпець[16], а x – *n*-вимірний вектор-стовпець, що розшукується:

$$A=\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix}, \quad b=\begin{pmatrix} b_1 \\ b_2 \\ \dots \\ b_m \end{pmatrix} \quad x=\begin{pmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{pmatrix}. \tag{2.2}$$

Подане матричне рівняння може також розглядатись як *система m лінійних рівнянь з n невідомими*:

[16] Ще раз просимо виконувати домовленість: велика літера *A*, *B*, .. – для матриць; маленька *a*, *b*, .. – для векторів та чисел!

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + ... + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + ... + a_{2n}x_n = b_2 \\ . \ \ldots\ldots\ldots\ldots\ldots\ldots \\ a_{m1}x_1 + a_{m2}x_2 + ... + a_{mn}x_n = b_m . \end{cases} \qquad (2.3)$$

Розв'язком матричного рівняння (2.1), або **розв'язком системи** (2.3) називають, як відомо, такий вектор x, що обертає їх в тотожності. В більшості випадків мають справу *з квадратними матрицями*, $n=m$, тобто з випадком, коли *кількість рівнянь збігається з кількістю невідомих*. Умови існування та єдиності розв'язку такої задачі точно встановлені. Відомо, що СЛАР (2.3), або матричне рівняння (2.1) може мати (*i*) один єдиний розв'язок, (*ii*) може не мати жодного, або (*iii*) нескінченну кількість розв'язків [15,45]. Скажімо, лише 2 розв'язки, або 3 – таке неможливо!

Розглянемо методи розв'язування таких систем лінійних рівнянь (СЛАР). Теоретично, розв'язати їх можна за допомогою визначників. З курсу вищої математики [36,45] відоме ***правило*** (***метод***) ***Крамера***. За цим правилом, при розв'язуванні системи (2.1), k-та компонента x_k вектора x визначається згідно з формулою

$$x_k = \frac{\Delta_k}{\Delta}, \qquad (2.4)$$

де Δ –визначник матриці A, Δ_k – визначник допоміжної матриці A_k, в якій k-тий стовпчик замінюється вектором b.

Інший спосіб (його називають **матричний**) полягає в обчисленні оберненої матриці A^{-1}, після чого

$$x = A^{-1} b . \qquad (2.4A)$$

В MATLAB, як вже знаємо з 1.4, є команда швидкого і ефективного обчислення оберненої матриці. Якщо надати їй ім´я *A_inv*, то *A_inv= inv*(*A*). Розв'зком (2.3) буде вектор

*x= A_inv*b.*

Найбільш ефективним методом чисельного розв'язування СЛАЕ – особливо великої вимірності! –

вважають метод Гауса, винайдений цим великим німецьким математиком XIX сторіччя.

Увага: завжди перевіряйте отриманий розв'язок, як це показано у прикладі 1.12, розділ 1.7.5! Якщо отриманий вектор x помножили на матрицю A і не отримали вектор правих частин b, то у вашому розв'язку помилка! Або обчислити A**x*- *b* – має бути нуль хоча б з „машинною точністю", „машинний нуль", див. приклад 1.12.

Теоретичним шляхом встановлені умови існування і єдиності розв'язку даної задачі: **Матриця *A* має бути невиродженою, тобто** $\Delta \neq 0$**.**

Розглянемо два з названих методів, звертаючи особливу увагу на їх алгоритмічну сторону.

2.1.1. Розв'язування СЛАР методом Крамера

Припустимо, треба розв'язати СЛАР (2.3) з конкретними числовими коефіцієнтами

Наступний ***алгоритм*** (послідовність дій), здається, є найбільш раціональним в MATLAB:

$$\begin{aligned} 4x_1 + 2{,}4x_2 - 0{,}8x_3 + x_4 &= 9, \\ x_1 + 3x_2 - 1{,}5x_3 - x_4 &= 8, \\ 0{,}4x_1 - 0{,}8x_2 + 4x_3 + 2x_4 &= 20, \\ 2x_1 - x_2 - 2x_3 - 3x_4 &= 7. \end{aligned}$$

```
>> % Створення матриці системи
>> A=[4, 2.4, -0.8, 1 ;1, 3, -1.5 -1 ;0.4, -0.8, 4, 2 ;2, -1, -2, -3];
>>% Створення вектора-стовпчика правої частини СЛАР
>> b=[9 ; 8 ; 20 ; 7]
>> D=det(A); % Визначник СЛАР
>>% Допоміжна A1 з b у першому стовпчику, її визначник
>> A1=A; A1( : ,1)=b; D1=det(A1) ;
>>% визначник D2
>> A2=A; A2( : ,2)=b; D2=det(A2)
>> % визначник D3
>> A3=A; A3( : ,3)=b; D3=det(A3) ;
>> % визначник D4
>> A4=A; A4( : ,4)=b; D4=det(A4) ;
```

>> % Отримуємо розвязок:
*>> x1=D1/D***;** *x2=D2/D***;** *x3=D3/D***;** *x4=D4/D***;**
>> % Обов'язково перевіряємо розв'язок:
>> A * [*x1***;** *x2* **;** *x3* **;** *x4*]

ans = 9.0000
8.0000
20.0000
7.0000

Дійсно, отримали вектор правих частин – СЛАР розв'язано вірно!

Зроблено було кілька однакових, однотипних дій; їх можна робити, викликаючи попередній командний рядочок та редагуючи його „руками". Таким чином можна працювати не замислюючись, як робот, із СЛАР довільного порядку.

А тепер – тому самому алгоритму надамо трохи іншу форму, більш „автоматизовану":

>>% Чисто формальна дія:
>> ***I=1***; *% Увага*: *важливо для програмістів!*
>>% Допоміжна матриця і її визначник D1
>> ***I= I +1***, *A1=A*; *A1*(**:** , *I*)=*b*; *D1*(*I*)=*det(A1*) **;**
>> матриця і її визначники D1(2)*, D1*(3)*, D1*(4)
>> ***I=I+1***, *A1=A*; *A1*(**:** , *I*)=*b*; *D1*(*I*)=*det(A1*) **;** *%*
>> ***I=I+1***, *A1=A*; *A1*(**:** , *I*)=*b*; *D1*(*I*)=*det(A1*) **;**
>> ***I=I+1***, ***;*** *A1=A*; *A1*(**:** , *I*)=*b*; *D1*(*I*)=*det(A1*) **;**

Алгоритм став зовсім „автоматизованим": вектор *b* „сам" вставляється у потрібний стовпчик, „руками" слід лише натискувати „↑" та „Ввід", аж поки не отримаємо *I=4*, вимірність даної СЛАР.

Увага, використано стандартний прийом програмування: введення штучної змінної ***I***, що спочатку дорівнює нулю а потім рахує номер рівняння (номер невідомої x_i) ***I=I+1***, використано для програмування такої реалізації методу Крамера... . Остаточно:

>> % Отримуємо розв'язок:
*>> x=D1/D***;**

2.1.2. МЕТОД ГАУСА РОЗВ'ЯЗУВАННЯ СЛАР

Однак для практичних задач, коли кількість рівнянь *n* велика (а в сучасних технічних або економічних задачах *n* дорівнює кільком тисячам) методи Крамера та оберненої матриці стають надто громіздкими і потребують значних затрат машинного часу, тому в обчислювальній практиці застосовуються не завжди. Метод Гауса є зручним саме для ЕОМ. Цей метод відомий вам ще зі школи як *метод виключення невідомих*. Розглянемо систему рівнянь (2.3), у якій вільні члени будемо позначати як $a_{k,n+1} = b_k$. Алгоритм розв'язку складається з двох етапів. Перший з них називається ***прямий хід***.

Розділимо перше рівняння системи на a_{11} і результат запишемо у вигляді

$$x_1 + a_{12}^{(1)} x_2 + a_{13}^{(1)} x_3 + ... = a_{1,n+1}^{(1)}. \qquad (2.5)$$

Тут і надалі верхній індекс означатиме номер перетворення, яке здійснене. Помножимо це рівняння на $-a_{21}$ і додамо його до другого рівняння системи (2.3). Коефіцієнти отриманого другого рівняння мають тепер вигляд $a_{2j}^{(1)} = a_{2j} - a_{21} a_{1j}^{(1)}$, де індекс j пробігає значення $j = 1,2,..., n+1$. При цьому перший коефіцієнт нового рівняння $a_{21}^{(1)}$ обертається в нуль, бо $a_{11}^{(1)} = 1$. Так само поводимось із наступними рівняннями, тобто від *k*-го рівняння віднімаємо рівняння (2.5), попередньо помножене на a_{k1}. Його нові коефіцієнти мають вигляд[17]

$$a_{kj}^{(1)} = a_{kj} - a_{k1} a_{1j}^{(1)} \text{ для всіх } j = 1,2,..., n+1.$$

---

[17] Пояснимо складну для новачків систему позначень: $a_{kj}^{(r)}$ означає коефіцієнт з k-го рівняння (рядочка матриці *A*), j-го рядочка *A* після r-го перетворення.

Після виконання таких перетворень для всіх рівнянь системи, $k = 2,3,..., n$, маємо нову систему рівнянь

$$x_1 + a_{12}^{(1)} x_2 + a_{13}^{(1)} x_3 + ... + a_{1n}^{(1)} x_n = a_{1,n+1}^{(1)}$$
$$a_{22}^{(1)} x_2 + a_{23}^{(1)} x_3 + ... + a_{2n}^{(1)} x_n = a_{2,n+1}^{(1)}$$
$$\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots$$
$$a_{n2}^{(1)} x_2 + a_{n3}^{(1)} x_3 + ... + a_{nn}^{(1)} x_n = a_{n,n+1}^{(1)},$$

де всі коефіцієнти в першому стовпчику матриці $A^{(1)}$ дорівнюють нулю, за винятком $a_{11}^{(1)} = 1$.

На наступному кроці перше рівняння нової системи (відповідно, перший рядочок матриці $A^{(1)}$) лишаємо незмінним, а аналогічну процедуру застосовуємо до рівнянь від другого до *n*-го, останнього. Маємо формули для нових коефіцієнтів (*i* – номер рівняння, *j* – номер стовпчика):

$$a_{2j}^{(2)} = a_{2j}^{(1)} / a_{22}^{(1)}, \quad j = 2,3,...,n+1,$$
$$a_{ij}^{(2)} = a_{ij}^{(1)} - a_{i2}^{(1)} a_{2j}^{(2)}, \quad i = 3,4,...,n, \quad j = 3,4,...,n+1.$$

Таку саму процедуру повторюємо для всіх рядків матриці *A*. Так, для *k*-го кроку ($k = 1,2,...,n-1$) маємо для елементів *k*-го і наступних *i*-х рівнянь:

$$a_{kj}^{(k)} = a_{kj}^{(k-1)} / a_{kk}^{(k-1)}$$
$$a_{ij}^{(k)} = a_{ij}^{(k-1)} - a_{ik}^{(k-1)} a_{kj}^{(k)}, \qquad (2.6)$$

де $i = k+1,...,n, \quad j = k+1,...,n+1$.

Після $n-1$ кроків система рівнянь лишається еквівалентною вихідній, але набуває такого "трикутного" вигляду:

$$x_1 + a_{12}^{(1)} x_2 + a_{13}^{(1)} x_3 + a_{14}^{(1)} x_4 + ... + a_{1n}^{(1)} x_n = a_{1,n+1}^{(1)}$$
$$x_2 + a_{23}^{(2)} x_3 + a_{24}^{(2)} x_4 + ... + a_{2n}^{(2)} x_n = a_{2,n+1}^{(2)} \qquad (2.7)$$
$$x_3 + a_{34}^{(3)} x_4 + ... + a_{3n}^{(3)} x_n = a_{3,n+1}^{(3)}$$
$$\ldots\ldots\ldots\ldots\ldots\ldots\ldots$$
$$x_{n-1} + a_{n-1,n}^{(n-2)} x_n = a_{n-1,n+1}^{(n-2)}$$
$$a_{n,n}^{(n-1)} x_n = a_{n,n+1}^{(n-1)}$$

На цьому **прямий хід** методу виключення Гауса закінчено. З трансформованої системи (2.7) безпосередньо маємо x_n, а наступні компоненти розв'язку ***рекурентно*** отримуємо „знизу вгору”:

$$x_n = \frac{a_{n,n+1}^{(n-1)}}{a_{n,n}^{(n-1)}}, \qquad x_{n-1} = \frac{a_{n-1,n+1}^{(n-2)} - a_{n-1,n}^{(n-3)} x_n}{a_{n-1,n-1}^{(n-1)}}, \quad \ldots, \quad (2.8)$$

$$x_2 = a_2^{(2)} - a_{23}^{(2)} x_3 - \ldots - a_{2n}^{(2)} x_n,$$

$$x_1 = a_1^{(1)} - a_{12}^{(1)} x_2 - \ldots - a_{1n}^{(1)} x_n.$$

Цей процес послідовного обчислення значень невідомих називається ***зворотним ходом*** методу Гауса.

Алгоритм ***методу виключення Гауса*** складається, таким чином, з двох етапів. Якщо на етапі прямого ходу виникає випадок, коли $a_{kk}^{(r)} = 0$, то k-й рядок неможливо використовувати для виключення елементів k-го стовпчика. В цьому випадку k-й рядок замінюється іншим, що лежить нижче. Так здійснити неможливо лише у тому випадку, коли всі елементи $a_{p,k}^{(r)} = 0$, $p > k$, а це відповідає тому, що вихідна матриця A є виродженою і система не має єдиного розв’язку.

Розглянемо тепер реалізацію метода Гауса в середовищі MATLAB. А дали ознайомимось ще з одним методом.

Приклад 2.1. Візьмемо конкретну СЛАР

$$\begin{cases} x_1 + 2x_2 + x_3 + 4x_4 = 13 \\ 2x_1 + 4x_3 + 3x_4 = 28 \\ 4x_1 + 2x_2 + 2x_3 + x_4 = 20 \\ -3x_1 + x_2 + 3x_3 + 2x_4 = 6. \end{cases} \quad (2.9)$$

Використовуючи систему MATLAB, подамо матрицю A і вектор-стовпець b у вигляді рядочків

```
>> A=[1  2  1  4; 2  0  4  3; 4  2  2  1; -3  1  3  2]
>> b=[13  28  20  6]'
```

(**Увага:** апостроф поза матрицею **'** означає операцію *транспонування*, яка вектор-рядочок перетворює у стовпчик, і навпаки, вектор-стовпчик у рядочок!).

На екрані монітора після натискання клавіші *Enter* з'явиться таке зображення

```
A =   1   2   1   4
      2   0   4   3
      4   2   2   1
     -3   1   3   2
b =  13
     28
     20
      6
```

– це дана матриця і стовпчик вільних членів вихідної СЛАР.

Покажемо, як у MATLAB побудувати простий алгоритм її розв'язування методом виключення Гауса. Спочатку командою

>> Ab=[*A*, *b*]

створимо розширену матрицю, яка включає як коефіцієнти рівнянь, так і праві частини (назвали її *Ab*):

```
Ab =
   1   2   1   4  13
   2   0   4   3  28
   4   2   2   1  20
  -3   1   3   2   6
```

Тепер командою

>> Eq1=Ab(*1*,**:**)

створюємо допоміжний вектор-рядок, що відповідає першому рівнянню:

```
Eq1 =   1   2   1   4  13
```

Так само командами

>> Eq2=Ab(*2*,:),
>> Eq3=Ab(*3*,:),
>> Eq4=Ab(*4*,:)

створюємо вектор-рядки, що відповідають трьом останнім рівнянням:

```
Eq2 =   2   0   4   3  28
```

Eq3 = 4 2 2 1 20
Eq4 = -3 1 3 2 6

Тепер можна починати еквівалентні перетворення рівнянь відповідно до прямого ходу методу Гауса.

Після команди

>> Eq1=Eq1/Eq1(1,1)

перший коефіцієнт першого рівняння дорівнює одиниці. Виконуємо тепер три (скільки лишилося рівнянь) команди

*>> Eq2=Eq2-Eq1*Eq2(1,1), Eq3=Eq3-Eq1*Eq3(1,1),*
*Eq4=Eq4-Eq1*Eq4(1,1)*

Система рівнянь перетворена до вигляду (2.5), тобто перший коефіцієнт у першому з них дорівнює 1. Тепер рівняння з другого по четверте перетворюємо командами

>> Eq2=Eq2/Eq2(1,2)

та

*>> Eq3=Eq3-Eq2*Eq3(1,2),*
*>> Eq4=Eq4-Eq2*Eq4(1,2).*

(Можна бачити, що ці дії, подібні до попередніх. Тому, клавішею ↑, стрілка вверх, у командному вікні MATLAB можна викликати попередню команду і відредагувати її вручну – змінити відповідні індекси).

Тепер перетворюємо рівняння з третього по четверте

*>> Eq3=Eq3/Eq3(1,3), Eq4=Eq4-Eq3*Eq4(1,3).*

Прямий хід на цьому завершено. Допоміжні вектори-рядки тепер мають такий вигляд на екрані комп'ютера:

Eq1 = 1 2 1 4 13;
Eq2 = 0 -4 2 -5 2;
Eq3 = 0 0 -5 -7.5 -35
Eq4 = 0 0 0 -9 -18

Таким чином, за формулами (2.6) отримано трикутну систему рівнянь вигляду (2.7).

Проводимо **зворотний хід методу** за формулами (2.8). Відразу маємо

>> x4=Eq4(1,5)/Eq4(1,4)=2.

Далі:

*>> x3=Eq3(1,5) – Eq3(1,4)*x4=4,*

*>> x2=Eq2(1,5) – Eq2(1,4)*x4-Eq2(1,3)*x3= –1,*

*>> x1=Eq1(1,5) – Eq1(1,4)*x4 – Eq1(1,3)*x3 – Eq1(1,2)*x2=3.*

Вектор-стовпець *X*=[*x1*; *x2*; *x3*; *x4*] і буде розв'язком вихідної системи рівнянь даного прикладу.

Розв'язки слід обов'язково перевіряти. Це можна зробити таким чином: матрицю *A* слід помножити на стовпчик *X* ',

*>> A*X*

Результатом має бути стовпчик правих частин рівняння *b*'. Розв'язок отримано!

Даний алгоритм зручний тим, що повторює однакові дії. Але ж його недолік у тому, що не видно, як цей алгоритм узагальнити на систему довільного порядку *N*. Цьому заважає те, що кожне з рівнянь *Eq1*, *Eq2* і т.д. треба утворювати окремо. Надамо модифікацію алгоритму, яке ще краще показує шлях автоматизації обчислень.

>> I=0; *Abm=Ab*; *%Прямий хід*; *утворено допоміжну Abm*

>> I=I+1, *J=I, % I - кількість перетворень, J номер рівняння*

>> Abm(*I*, **:**)=*Abm*(*I*, **:**)*/Abm*(*I,I*)

Пересвідчимось, що в модифікованій розширеній матриці *Abm* перший елемент стає рівним 1. Наступну дію

>> J=J+1, Abm(*J*, **:**)=*Abm*(*J*, **:**)-*Abm*(*J,I*)**Abm*(*I*, **:**)

повторюємо кілька разів, поки не стане *J= N =4*. Отримаємо матрицю *Abm* з „нульовим" першим стовпчиком:

J = 4

Abm =

1	*2*	*1*	*4*	*13*
0	*-4*	*2*	*-5*	*2*
0	*-6*	*-2*	*-15*	*-32*
0	*7*	*6*	*14*	*45*

Перетворення для *I= 1* закінчено. Проводимо низку перетворень для *I= 2* – повторюємо другий та третій рядочки:

>> I=I+1, J=I, % I - кількість перетворень, J номер рівняння

>> Abm(*I*, **:**)=*Abm*(*I*, **:**)*/Abm*(*I , I*)

```
I =     2
J =     2
```

І знов четвертий рядочок

```
>> J=J+1, Abm(J, :)=Abm(J, :)-Abm(J,I)*Abm(I, :)
```

повторюємо поки не стане *J= N =4*. Отримаємо матрицю *Abm* з „нульовим" другим стовпчиком:

```
J =     4
Abm =
   1.0000   2.0000   1.0000   4.0000   13.0000
        0   1.0000  -0.5000   1.2500   -0.5000
        0        0  -5.0000  -7.5000  -35.0000
        0        0   9.5000   5.2500   47.5000
```

Перетворення для *I= 2* закінчено, повторюємо знов для *I= 3*:

```
>>  I=I+1, J=I, % I - кількість перетворень, J номер рівняння
>>  Abm(I, :)=Abm(I, : )/Abm( I , I)
I =     3
J =     3
```

і

```
>> J=J+1, Abm(J, :)=Abm(J, :)-Abm(J,I)*Abm(I, :)
J =     4
Abm =
   1.0000   2.0000   1.0000   4.0000   13.0000
        0   1.0000  -0.5000   1.2500   -0.5000
        0        0   1.0000   1.5000    7.0000
        0        0        0  -9.0000  -18.0000
```

Прямий хід закінчено коли *I=3= N-1*. Маємо в результаті трикутну матрицю.

Зворотній хід починаємо з останнього рядочка матриці *Abm*:

```
>> I=5;  %Початок зворотнього ходу
>> I= I -1, J=I, Abm(I,:)=Abm(I,:)/Abm(I,I)
```

і наступні дії повторюємо, доки *J=1*:

```
>> J=J-1,  Abm(J, :)=Abm(J, :)-Abm(I, :)*Abm(J,I)
```

Маємо „всі нулі" у четвертому стовпчику. Повторюємо другий рядочок для *I=3*:

```
>> I= I -1, J=I, Abm(I,:)=Abm(I,:)/Abm(I,I)
```

і відповідний цикл по *J*, доки *J=1*:

*>> J=J-1, Abm(J, :)=Abm(J, :)-Abm(I, :)*Abm(J,I)*

Маємо всі нулі (окрім головної діагоналі) у третьому стовпчику. Знов повторюємо другий рядочок для *I=2* і відповідний цикл по *J*, доки *J=1*:

>> I= I -1, J=I, Abm(I,:)=Abm(I,:)/Abm(I,I)

*>> J=J-1, Abm(J, :)=Abm(J, :)-Abm(I, :)*Abm(J,I)*

Ось і маємо остаточну трикутну розширену матрицю, де є лише одиниці на головній діагоналі!

```
J =     1
Abm =
          1    0    0    0    3
          0    1    0    0   -1
          0    0    1    0    4
          0    0    0    1    2
```

У ній останній стовпчик – розв'язок системи. Отримуємо його

```
>> x=Abm( : ,4+1)   %Отримуємо розв'язок
```

і перевіряємо:

```
>>  A*x
ans =    13
         28
         20
          6
```

Розв'язок правильний, бо добуток дорівнює правій частині вихідного рівняння!

Ми бачили тут, що алгоритм прямого ходу складається з двох циклів: зафіксували *I=4*, провели цикл обчислень для *J=1, 2, 3* і *4*; потім зафіксували *I=3*, провели цикл обчислень для *J= 2, 3* і *4*; зафіксували *I=2*, провели обчислення для *J= 3* і *4*... Аналогічно, „цикли по *J*" мають місце у середині циклів по *I* для зворотного ходу. Алгоритм у такій формі легко піддається програмуванню, і це ми зробимо далі.

2.2. РОЗВ'ЯЗУВАННЯ НЕЛІНІЙНИХ РІВНЯНЬ

Досить часто виникає потреба знати корені нелінійного рівняння загального виду

$$f(x) = 0\,. \qquad (2.10)$$

Коли функція $y = f(x)$ є поліномом, рівняння (2.10) називається ***алгебраїчним***; для аналізу і розв'язування таких рівнянь розроблені спеціальні методи, див. [34,35]. У випадку алгебраїчного рівняння, коли $f(x)$ є поліномом будь-якого степеня, теорія завжди може відповісти на питання про існування і кількість розв'язків рівняння (2.10), див. 1.7.7.

Якщо функція $f(x)$ складається лише з тригонометричних, степеневих та інших елементарних функцій, то рівняння (2.10) називають відповідно тригонометричним, степеневим, логарифмічним тощо. Якщо у його запису присутні функції з різних класів – рівняння називають *трансцендентним*. Загальної теорії для таких рівнянь немає. Інколи ми навіть не знаємо аналітичний вираз функції $f(x)$ (отримуємо її, скажімо, з чисельного розв'язку якоїсь іншої складної задачі в залежності від параметра x) – все одно можна ставити питання про наявність коренів рівняння (2.10) та їхнє знаходження з потрібною для тих чи інших практичних цілей точністю. Далі коротко викладено кілька найбільш поширених методів.

2.2.1. ГРАФІЧНИЙ МЕТОД, ВІДОКРЕМЛЕННЯ КОРЕНІВ

"Шкільний" метод визначення наявності кореня та його грубого знаходження за допомогою побудови графіка функції $y = f(x)$ є найбільш простим і переконливим. Як показано в 1.7.6, команда *solve*( ) видає лише один корінь, *fzero*( ) може видати кілька (якщо міняти початкове наближення x_0), та завжди лишається питання – чи всі? І лише графічний метод дозволяє дати повну відповідь. Його застосування полегшується ефективністю сучасних

комп'ютерів і великої кількості простих для застосування графічних програм. Наведемо два приклади.

Приклад 2.2. Нехай потрібно знайти корінь трансцендентного рівняння

$$\cos x = k\,x, \qquad (2.11)$$

де параметр k – дійсне число, додатне чи від'ємне. У даному випадку функцію $f(x) = \cos x - k\,x$ доцільно розглядати складеною з двох функцій $\varphi(x) = \cos x$ і $\psi(x) = k\,x$. Рівняння (2.11), очевидно, еквівалентне рівнянню $\varphi(x) = \psi(x)$. Графіки кожної з цих функцій неодноразово будували у школі: це буде, відповідно, косинусоїда і пряма лінія через початок координат і з тангенсом кута нахилу k. З MATLAB значно легше:

```
>> % Косинусоїда
>> syms x
>> LeftSide=cos(x)
>> ezplot(LeftSide, [-10, 10])
>> % Права частина для k=0,5
>> RightSide=0.5*x
>> figure(1), hold on, ezplot(RightSide,[-10, 10])
>> % Права частина для k=0,1
>> RightSide=0.1*x
>> figure(1), ezplot(RightSide,[-10, 10])
>> % Права частина для k=-0,3
>> RightSide=0.3*x
>> figure(1), ezplot(RightSide,[-10, 10])
>> % Надписи на рисунку:
>>  figure(1), legend('cos(x)', 'k=0,5', '  0,1', ' -0,3')
```

Графіки показані на рис. 2.1, де пряма побудована для кількох значень k. Перетин її з косинусоїдою й дає відповідний розв'язок рівняння (2.11). **Можна зробити висновок**, що кількість розв'язків даного рівняння залежить від значення k, а саме:

а) при $k = 0{,}5$ (пряма 1) маємо лише один корінь рівняння, що лежить між 0 і $\pi/2$ (можна знайти „чисельним

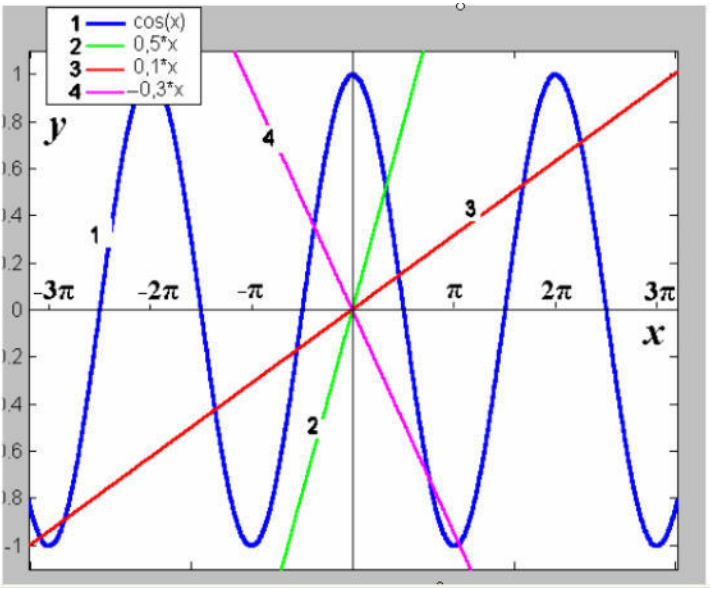


Рис. 2.1. До графічного розв'язування рівняння cos $x = k\,x$ для кількох значень k: крива **1** – косинусоїда; **2** – пряма для k=0,5; **3** –для k=0,1; **4** – пряма для k= -0,3

експериментом", що один корінь буде для усіх k, таких, що $-0{,}35 < k < 0{,}35$);

б) при $k = 0{,}1$ (червона пряма) рівняння має аж сім коренів, які знаходяться у інтервалах: $x_1 \in (0, \frac{\pi}{2})$, $x_2 \in (\pi, 2\pi)$, $x_3 \in (2\pi, 3\pi)$, $x_4 \in (-\pi, 0)$, $x_5 \in (-2\pi, -\pi)$, $x_6 \in (-3\pi, -2{,}5\pi)$ і $x_7 \in (-3{,}5\pi, -3\pi)$.

в) при $k = -0{,}3$ маємо лише три корені: $x_1 \in (-\frac{\pi}{2}, 0)$, $x_2 \in (\frac{\pi}{2}, \pi)$ і $x_3 \in (\pi, 1.5\pi)$.

Таким чином, у кожному випадку графічна побудова дозволяє принципово з'ясувати кількість коренів рівняння, що досліджується. Однозначно можна бачити, що інших коренів, окрім вказаних для певного значення k, не має, бо

далі пряма уходить на нескінченність, а тригонометрична функція лишається коливатись між -1 та 1. Знаходимо, окрім того, і приблизні значення коренів, тобто вказуємо інтервал $a \le x \le b$, де знаходиться лише один корінь. Така процедура називається *локалізацією* коренів. За допомогою іконки графічного вікна можемо знайти корені й більш точно. У більшості „шкільних" випадків цього достатньо. Побудову графіка часто використовують для того, аби лише грубо встановити, чи існує корінь рівняння на певному проміжку $[x_{\text{п}}, x_{\text{к}}]$. Таку процедуру називають *відокремленням кореня*. Важливо відмітити, що, оскільки *неперервна* функція $f(x)$ з рівняння (2.10) має різні знаки на кінцях інтервалу $x_{\text{п}}$ і $x_{\text{к}}$ (індекси обрано від слів "Початок" і "Кінець"), то критерієм існування на ньому коренів може служити умова

$$f(x_{\text{п}}) \cdot f(x_{\text{к}}) < 0. \qquad (2.12)$$

Її зручно використовувати в комп'ютерних програмах для „автоматичного" розв'язку рівнянь.

В такий спосіб неможливо, однак, знаходити комплексні корені. Щодо останніх, то повна теорія існує лише для многочленів, та це лишається поза нашої уваги.

2.2.2. МЕТОД ПОДІЛУ ВІДРІЗКА НАВПІЛ

Не завжди можна скористатись нашим зором, і треба, буває, "доручити" програмі "самій" обрати рішення щодо наявності кореня рівняння. Для цього обчислювач повинен "алгоритмізувати" процес обчислень.

На першому етапі треба *відокремити корінь* рівняння (2.10), тобто знайти такі кінці інтервалу $x_{\text{п}}$ і $x_{\text{к}}$, для яких виконується умова. В багатьох випадках їх можна знайти графічним методом, як у попередньому розділі. Бувають випадки, коли кінці інтервалу $x_{\text{п}}$ і $x_{\text{к}}$ для певної задачі відомі теоретично. Часто, для неперервної функції $f(x)$, допомагає наступний алгоритм.

Крок 1, *початковий*: довільно обирають *початкове наближення* x_0 і *крок пошуку* $\Delta x = \Delta x_0$ і обчислюють функцію в точках x_0 і $x_1 = x_0 + \Delta x$.

Крок 2, *визначення напрямку пошуку*: якщо $|f(x_1)| < |f(x_0)|$ (наступне значення ближче до нуля, ніж попереднє) – напрямок вірний, знак Δx не змінюють; якщо ж $|f(x_1)| > |f(x_0)|$ (ще далі віддалилися від нуля) – напрямок невірний, змінюють його на протилежний, тобто приймають $\Delta x = -\Delta x$.

Крок 3, *локалізація кореня*: обчислюють послідовно $x_2 = x_1 + \Delta x$ і $f(x_2)$, $x_3 = x_2 + \Delta x$ і $f(x_3)$, …, $x_n = x_{n-1} + \Delta x$ і $f(x_n)$, кожного разу перевіряючи умову $f(x_{n-1}) \cdot f(x_n) < 0$, $n = 1,2,...$. Як тільки це сталося – щойно отримані точки і дають кінці інтервалу $x_{\text{п}} = x_{n-1}$ і $x_{\text{к}} = x_n$, на якому знаходиться корінь рівняння (2.10). Можна переходити до наступного етапу.

Треба мати на увазі, що описаний алгоритм не завжди дозволяє локалізувати корінь. Якщо на кроці 2 маємо рівність $f(x_1) = f(x_0)$, то визначитись з напрямком пошуку ще не можна. Тоді треба зробити ще крок у тому ж напрямку $x_2 = x_1 + \Delta x$, порівняти $f(x_2)$ і $f(x_1)$ і знову спробувати визначитись. На кроці 3 для довільної функції $f(x)$ не гарантовано, що умова (2.12) відбудеться – це означає, що у програмі треба передбачити "*аварійну зупинку*" для запобігання її нескінченної роботи.

<u>На другому етапі</u>, коли $x_{\text{п}}$ і $x_{\text{к}}$, тобто кінці інтервалу, що вміщують корінь, вже відомі, треба *уточнити корінь*, тобто знайти його остаточно з потрібною точністю. Найбільш простим і надійним способом розв'язування нелінійних рівнянь вважають *метод ділення відрізка навпіл*. Наступні наближення обирають тепер за правилом

$$\Delta x = \Delta x / 2, \qquad x' = x_{\text{п}} + \Delta x. \tag{2.13}$$

Знов перевіряють умову (2.12). Оскільки для кінців інтервалу $x_{\text{п}}$ і $x_{\text{к}}$ вона виконана, то обов'язково буде виконуватись одне із співвідношень,

$$\text{або } f(x_{\text{п}})\cdot f(x')<0, \qquad \text{або } f(x')\cdot f(x_{\text{к}})<0. \tag{2.14}$$

Перше з них означає, що корінь знаходиться у лівій половині інтервалу, тобто x' можна вважати тепер за кінець уточненого інтервалу, тобто покласти $x_{\text{к}} = x'$. У другому випадку корінь лежить у правій половині інтервалу, між новим початком $x_{\text{п}} = x'$ і кінцем $x_{\text{к}}$. Знаємо, тобто, новий, вдвічі вужчий інтервал $[x_{\text{п}}, x_{\text{к}}]$, де знаходиться корінь рівняння.

Уточнення кореня за формулою ділення навпіл (2.13) і логічними умовами (2.14) продовжуємо доти, поки довжина інтервалу Δx не стане меншою за потрібну точність ε. Виконання останньої умови означатиме, що корінь рівняння (2.10) з точністю ε знайдено.

Треба зазначити, що під час перевірки умови (2.14) може статися і "точне попадання в корінь" $f(x_{\text{п}})\cdot f(x')=0$. На цей доволі рідкісний випадок треба передбачити дострокове припинення обчислень, бо x' є корінь.

Більш детально про цей метод дивіться у [2,34,35]. Цей метод з кожним кроком уточнення дає удвічі краще наближення до кореня, що вважається доволі повільним. Деякі з наступних методів мають швидшу збіжність.

2.3. ПІДБІР ЕМПІРИЧНИХ ФУНКЦІЙ МЕТОДОМ НАЙМЕНШИХ КВАДРАТІВ

Проблема пошуку емпіричного рівняння чи не найчастіше зустрічається в практиці інженера або наукового працівника-експериментатора в будь-якій галузі науки і техніки. Йдеться ось про що.

Нехай в експерименті, науковому або інженерному, встановлено залежність між вхідним параметром *x* (аргументом) і вихідним параметром *y* (функцією). Результат такого експерименту фіксують у вигляді таблиці 2.1 з даними *N* дослідів:

Таблиця 2.1: Результати дослідів

Номер досліду	Значення x	Значення y
1	x_1	y_1
2	x_2	y_2
…	…	…
N	x_N	y_N

Ці ж дані можна представити у графічній формі, скажімо – значками **o** , як показано на рис. 2.2. Якщо послідовні експериментальні точки $\{x_i, y_i\}, i = 1,..., N$ з'єднати прямими, то отримаємо ламану лінію, яка буде тим "гірше", чим більше похибка експериментальних вимірів. Область використання таких даних обмежується лише умовами даного експерименту. Однак, вона буде значно ширшою, якщо знайти *аналітичну формулу* $y = f(x)$, яка наближує дані експериментальні точки. Тоді легко робити *інтерполяцію* даних експерименту (тобто наближено визначати значення функції y **між** сусідніми значеннями аргументу x_k і x_{k+1}) і навіть *екстраполяцію* (прогноз **за межі** експерименту, тобто для $x < x_1$ або $x > x_N$). Знаходження такої ***емпіричної функції*** і є задачею даного розділу. Додамо, що задачі такого роду можуть виникнути і в теоретичній роботі, коли, скажімо, результати складного розрахунку треба подати наближеною, але простою і зручною в користуванні формулою. Інший випадок: вигляд формули встановлено теорією, як це було, наприклад, із законом всесвітнього тяжіння, але значення коефіцієнтів невідомі і мають бути знайдені зі спеціальних експериментів.

Викладемо ***метод найменших квадратів*** (***МНК***) розв'язання такої задачі. Хоча цей красивий метод

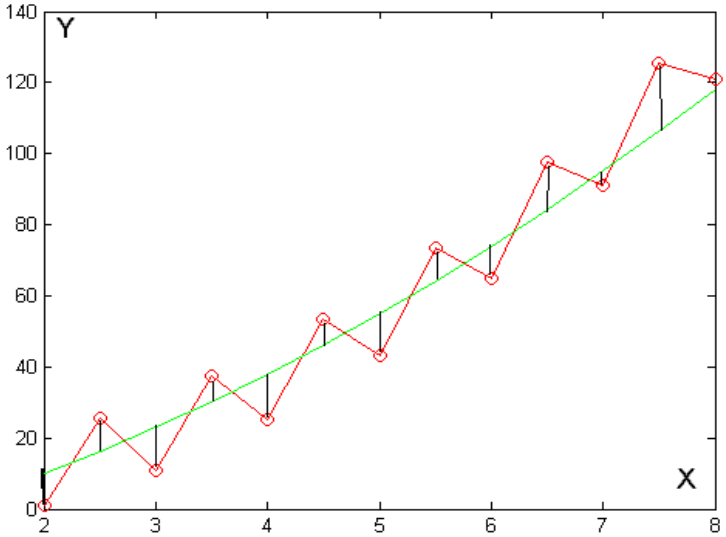


Рис. 2.2. Емпіричні дані позначені значком **o** і з'єднані ламаною, разом із кривою, що їх наближує

запропоновано ще Гаусом і Лежандром в 19-му сторіччі [34,35,38] він плідно використовується у багатьох розділах математики і, безумовно, має складати науковий багаж сучасного фахівця.

На першому етапі побудови емпіричної формули треба встановити (запропонувати) вигляд цієї формули. Для цього треба поєднати візуальний вигляд вже побудованої ламаної з вашими знаннями про функції і їх графіки. Якщо ламана, що на графіку сполучає експериментальні точки, близька до прямої лінії, то аналітичну залежність слід шукати у вигляді лінійної функції $y = Ax + B$ з деякими коефіцієнтами A і B, які треба знайти. Якщо ламана зростає або спадає приблизно як парабола $y = \pm Ax^2 + B$, то останню функцію і слід взяти як емпіричне рівняння. Більш загально, буває корисним і зручним дослідити степеневу функцію $y = \pm Ax^{\alpha} + B$ або многочлен n-го степеня $y = a_1 x^n + a_2 x^{n-1} + a_3 x^{n-2} + \ldots + a_{n-1} x + a_n$, які мають

параметри (коефіцієнти) відповідно A, B, α і a_1, a_2, …, a_n. Якщо функція дуже швидко необмежено зростає – використовують показникову функцію $y = Ae^{\alpha x}$. Якщо візуально видно певну періодичність функції – доцільно звертатись до тригонометричних функцій $y = A\sin(\omega x)$, $y = B\cos(\omega x)$, або до їхніх сум. У "важких випадках" радимо переглянути довідник [33].

На другому етапі слід знайти "найкращі" значення параметрів, коефіцієнтів вибраної емпіричної функції. Нехай обрана функція має вигляд

$$y = f(x; A_1, A_2, ..., A_M), \qquad (2.15)$$

тобто залежить, крім аргументу x, від M параметрів $A_1, A_2, ..., A_M$. Буває, що кількість параметрів співпадає із кількістю експериментальних даних, $M = N$. Тоді, підставляючи у формулу (2.15) пари експериментальних значень, маємо N рівнянь з N невідомими

$$y_i = f(x_i; A_1, A_2, ..., A_N), \quad i = 1,2,..., N.$$

Існують різні способи *точно* провести криву (2.15) через експериментальні точки, див. підручники [34,38]. Тут викладемо принципово інший метод.

Дійсно, якщо експериментальні дані неминуче включають деяку похибку вимірювань, навіщо ж проводити криву (2.15) точно через відповідні точки?

Нехай крива (2.15) при певному виборі значень параметрів $A_1, A_2, ..., A_M$ і при *i*-му значенні аргументу x_i приймає значення функції $\tilde{y}_i$ (з "тільдою"), яке відрізняється від експериментального значення y_i, і так – для кожного з вимірювань $i = 1,..., N$. Тоді сума всіх відхилень

$$\sum_{i=1}^{N} (\tilde{y}_i - y_i)^2 \overset{Df}{=} S(A_1, A_2, ..., A_M) \qquad (2.16)$$

відрізняється від нуля. Така сума називається *нев'язкою* (англійською – *residual*) і є функцією коефіцієнтів $A_1, A_2, ..., A_M$. Природно обирати такі значення цих коефіцієнтів, за яких нев'язка має найменше значення. Геометрично це означає, що крива (2.15) проходить посередині між експериментальними точками. В цьому і полягає ідея **методу найменших квадратів**. Знайти мінімум функції (2.16) можна шляхом диференціювання нев'язки $S(A_1, A_2, ..., A_M)$ по кожному з параметрів і отримання M рівнянь з M невідомими.

Із загальною реалізацією методу найменших квадратів можна ознайомитись у книгах [35,37-40]. Тут обмежимось лише певними випадками.

1. <u>Емпіричне рівняння є лінійним.</u>

Залежність між аргументом і функцією шукаємо у вигляді лінійної функції

$$y = ax + b .$$

Тоді, нев'язка (2.16) має такий вигляд:

$$S(a,b) = \sum_{i=1}^{N} (y_i - ax_i - b)^2 .$$

Прирівняємо до нуля частинні похідні цієї функції по кожному з аргументів a і b, аби знайти мінімум нев'язки. Маємо два рівняння відносно a і b:

$$\frac{\partial S}{\partial a} = 2\sum_{i=1}^{N} (y_i - a x_i - b)(-x_i) = 0 ,$$

$$\frac{\partial S}{\partial b} = 2\sum_{i=1}^{N} (y_i - a x_i - b)(-1) = 0 .$$

Після простих спрощень і перетворень приходимо до лінійної системи двох рівнянь

$$a\sum_{i=1}^{N} x_i^2 + b\sum_{i=1}^{N} x_i = \sum_{i=1}^{N} x_i y_i ,$$

$$a\sum_{i=1}^{N} x_i + nb = \sum_{i=1}^{N} y_i .$$

Можна довести строго, що така система **завжди має**, і притому **єдиний розв'язок**, якщо кількість вимірів $N > 1$. Цей розв'язок легко знайти:

$$a = \frac{N\,S(x_i\,y_i) - S(x_i)\,S(y_i)}{N\,S(x_i^2) - S(x_i)^2}, \quad b = \frac{S(y_i)\,S(x_i^2) - S(x_i)\,S(x_i\,y_i)}{N\,S(x_i^2) - S(x_i)^2}, \qquad (2.17)$$

де позначено $S(x_i) = \sum_{i=1}^{N} x_i$ – сума перших степенів аргументів, $S(x_i^2) = \sum_{i=1}^{N} x_i^2$ – сума других степенів аргументів, $S(x_i\ y_i) = \sum_{i=1}^{N} x_i\ y_i$ – сума попарних добутків аргументів і функції і, нарешті, $S(y_i) = \sum_{i=1}^{N} y_i$.

Приклад 2.3. Нехай "експериментальні" дані подані таблицею 2.2 (власне, саме ці дані показано на рис. 2.2).

Таблиця 2.2. Результати умовного експерименту

x_i	2	2.5	3	3.5	4	4.5	5	5.5	6	6.5	7	7.5
y_i	1	25.5	11	37.5	25	53.5	43	73.5	65	97.5	91	125.5

Обчислюємо вказані вище суми:

$S(x_i) = 65$; $S(y_i) = 770$;

$S(x_i^2) = 370{,}5$ і $S(x_i\ y_i) = 4760$

(це, безперечно, можна обчислити "вручну", але легко робити в MATLAB командами на зразок

>> *Sx=sum(x), Sx2=sum(x .^2), Sxy=sum(x .* y),*

якщо *x* і *y* – відповідні вектори даних). За формулами (2.17) маємо $a = 2{,}66$, $b = -5{,}42$. Таким чином, лінійним емпіричним рівнянням для даних табл. 2.2 буде $y = 2{,}66x - 5{,}42$.

Увага: використовуються "операції з крапкою" **.^** і **.***!

2. <u>Емпірична формула у вигляді многочлена.</u>

Так само легко знайти "найкращий" (у сенсі найменшої середньоквадратичної нев'язки) серед многочленів заданого

степеня $n < N$, що наближує дані емпіричні дані з N вимірювань. На цьому шляху отримаємо систему n лінійних рівнянь з n невідомими, яка завжди однозначно розв'язується. Покажемо це на прикладі емпіричного рівняння другого степеня $y = a x^2 + b x + c$.

При так обраному емпіричному рівнянні, функція нев'язки має вигляд

$$S(a,b,c) = \sum_{i=1}^{N} (y_i - a x_i^2 - b x - c)^2 \cdot$$

Прирівнюючи до нуля частинні похідні $\frac{\partial S}{\partial a}$, $\frac{\partial S}{\partial b}$ і $\frac{\partial S}{\partial c}$, приходимо до системи трьох лінійних рівнянь з трьома невідомими:

$$a\sum_{i=1}^{N} x_i^4 + b\sum_{i=1}^{N} x_i^3 + c\sum_{i=1}^{N} x_i^2 = \sum_{i=1}^{N} x_i^2 y_i \ ,$$

$$a\sum_{i=1}^{N} x_i^3 + b\sum_{i=1}^{N} x_i^2 + c\sum_{i=1}^{N} x_i = \sum_{i=1}^{N} x_i y_i \ ,$$

$$a\sum_{i=1}^{N} x_i^2 + b\sum_{i=1}^{N} x_i + Nc = \sum_{i=1}^{N} y_i \ \cdot$$

Формула для розв'язку цієї системи має доволі громіздкий вигляд. Для знаходження чисельних значень коефіцієнтів a, b і c слід використати визначники, або операцію MATLAB

$$C=A\backslash b,$$

де матриця

$$A = \begin{pmatrix} S(x_i^4) & S(x_i^3) & S(x_i^2) \\ S(x_i^3) & S(x_i^2) & S(x_i) \\ S(x_i^2) & S(x_i) & N \end{pmatrix}$$

і вектор

$$b = \begin{pmatrix} S(x_i^2\ y_i) \\ S(x_i\ y_i) \\ S(y_i) \end{pmatrix},$$

а також введено додаткові позначення

$$S(x_i^4)=\sum_{i=1}^{N}x_i^4\,,\quad S(x_i^3)=\sum_{i=1}^{N}x_i^3$$

і $\quad S(x_i^2\,y_i)=\sum_{i=1}^{N}x_i^2\,y_i\,.$ (2.18)

Приклад 2.4. Нехай для тих самих "експериментальних" даних табл. 2.2 шукаємо емпіричне рівняння у вигляді квадратичної функції. Тоді, додатково до значень прикладу 2.2, маємо:

$$S(x_i^4)=15234\,,\quad S(x_i^3)=2307{,}5\quad \text{і}\quad S(x_i^2\,y_i)=31580\,.$$

Визначники системи рівнянь дорівнюють

$$\Delta_0=\begin{vmatrix}S(x_i^4) & S(x_i^3) & S(x_i^2)\\ S(x_i^3) & S(x_i^2) & S(x_i)\\ S(x_i^2) & S(x_i) & N\end{vmatrix},\ \Delta_0=\begin{vmatrix}S(x_i^2y_i) & S(x_i^3) & S(x_i^2)\\ S(x_i\,y_i) & S(x_i^2) & S(x_i)\\ S(y_i) & S(x_i) & N\end{vmatrix}$$

$$\Delta_2=\begin{vmatrix}S(x_i^4) & S(x_i^2y_i) & S(x_i^2)\\ S(x_i^3) & S(x_i\,y_i) & S(x_i)\\ S(x_i^2) & S(y_i) & N\end{vmatrix},\quad \Delta_3=\begin{vmatrix}S(x_i^4) & S(x_i^3) & S(x_i^2y_i)\\ S(x_i^3) & S(x_i^2) & S(x_i\,y_i)\\ S(x_i^2) & S(x_i) & S(y_i)\end{vmatrix}$$

і для умов прикладу дорівнюють

$\Delta_0=74\,011\,,\qquad \Delta_1=106\,620\,,\qquad \Delta_2=414\,050\qquad$ і

$\Delta_3=-725\,110\,.$

Нарешті маємо коефіцієнти емпіричного рівняння у вигляді квадратичної функції:

$$a=\Delta_1\big/\Delta_0\approx 1{,}44\,,\qquad b=\Delta_2\big/\Delta_0\approx 5{,}59\qquad \text{і}$$

$$c=\Delta_3\big/\Delta_0\approx -9{,}80\,.$$

Аналогічно можуть бути знайдені коефіцієнти полінома степеня більшого, ніж $n=2$.

Математичне середовище MATLAB дозволяє легко і швидко обчислювати ті громіздкі суми (2.18), знання яких потрібно для знаходження емпіричних рівнянь. Для цього доцільно використати оператор MATLAB *sum*(*v*) з вектором *v* як аргумент, як це було у Прикладі 2.3. Для отримання суми

$$S(x_i^2\ y_i) = \sum_{i=1}^{N} x_i^2\ y_i$$

слід наказати MATLAB:

>> Sx2y=sum((x .^2) . y).*

Проте, ***універсальність*** математичного середовища MATLAB в тому і полягає, що воно включає широкий набір внутрішніх, готових засобів для типових задач математичного моделювання. Пошук емпіричних рівнянь буде використано у розділі 4.

2.4. ІНТЕГРУВАННЯ ФУНКЦІЙ

У розділі 1.7.4 ми обчислювали інтеграли як символьні об'єкти MATLAB, не вдаючись навіть у питання „а що таке інтеграл?". Для чисельного обчислення з цього треба починати.

Як відомо [36], визначеним ***інтегралом на проміжку*** $a \le x \le b$ від функції $f(x)$ називають граничне значення суми

$$\sum_{i=1}^{n} f(\bar{x}_{i-1})(x_i - x_{i-1}), \qquad \bar{x}_{i-1} \in [x_{i-1}, x_i]. \tag{2.19}$$

де точки $x_0 = a$, x_1, x_2, …, $x_n = b$ (*вузли*) розподіляють проміжок $[a,b]$ на відрізки довжиною $\Delta x_i = x_i - x_{i-1}$ (останні найчастіше беруться рівної довжини Δx), коли кількість точок *n* нескінченно зростає і довжини відрізків, таким чином, прагнуть до нуля, $\Delta x \to 0$. Нагадаємо, що користуватись саме граничним значенням, яке позначають

$$\int_a^b f(x)dx, \tag{2.20}$$

замість формули (2.19) запропонували І. Ньютон і Г. Лейбніц, і з цим пов'язані найбільші успіхи фізико-

математичних наук, починаючи з XVII сторіччя. Геометричний смисл інтеграла (2.20) – площа криволінійної трапеції, обмеженої ліворуч і праворуч прямими $x = a$ і $x = b$, кривою $f(x)$ зверху і віссю Ox знизу.

У курсі вищої математики ви вивчили способи *інтегрування*, тобто пошуку аналітичних формул для інтегралів (2.20) від багатьох функцій $f(x)$. Проте, далеко не всі функції можуть бути "аналітично проінтегровані", наприклад – інтеграл $\int_a^b \frac{\sin x}{x} dx$ не може бути виражений у вигляді формули. Ще приклад – коли функціональна залежність f від x в аналітичному вигляді відсутня, а її отримано в результаті роботи якогось алгоритму. Практика поставляє багато саме таких задач. Для обчислення інтегралу від таких функції треба зробити "крок назад" від формули (2.20) до формули (2.19).

2.4.1. ФОРМУЛА ПРЯМОКУТНИКІВ ОБЧИСЛЕННЯ ІНТЕГРАЛІВ

Дійсно, виходячи з уявлення про інтеграл (2.20) як про площу, можна бачити, що формула (2.19) дає наближене значення цієї площі (рис. 2.3,А). При цьому *i*-й доданок в формулі (2.20) відповідає площі елементарного прямокутника $\Delta S_i = f(x_{i-1})(x_i - x_{i-1})$, який заштриховано на рисунку (використано сторону прямокутника f_{i-1} ліворуч від точки x_i). В даному випадку, для зростаючої функції, сума площ прямокутників наближає площу криволінійної трапеції "знизу".

Якщо площу елементарного прямокутника визначити інакше, $\Delta S_i = f(x_i)(x_i - x_{i-1})$, за стороною f_i праворуч від x_i, то сума площ наближає інтеграл від даної функції "зверху". В кожному випадку такі формули дають спосіб наближеного обчислення інтегралу і мають назву "*формули прямокутників*". Інший варіант обчислення площі, так звану

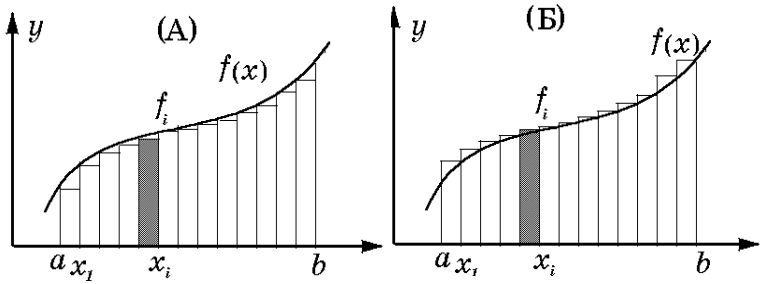


Рис. 2.3. Дві схеми інтегрування методом прямокутників: (А) наближення "знизу"; (Б) наближення "зверху"

„*формулу трапецій*", отримаємо, якщо значення функції f брати у середині відрізку $[x_{i-1}, x_i]$. Тоді маємо $\Delta S_i = f(\frac{1}{2}(x_i + x_{i-1})) \cdot (x_i - x_{i-1})$, а для інтеграла в цілому –

$$\int_a^b f(x)\,dx \approx \sum_{i=1}^{n} f(\tfrac{1}{2}(x_i + x_{i-1})) \cdot (x_i - x_{i-1}).$$

Кожна з формул зводить задачу інтегрування до арифметичних дій із значеннями функції f в певних точках інтервалу, *вузлах*, і дає доволі точні результати, якщо функція $f(x)$ є неперервною і змінюється "не дуже сильно". Вони легко програмуються, а для рівномірного кроку інтегрування $\Delta x = \frac{b-a}{n}$, формули мають особливо простий вигляд. Наприклад, формула (2.19) перетворюється у таку:

$$\int_a^b f(x)dx \approx \Delta x \sum_{i=0}^{n-1} f(x_i). \qquad (2.21)$$

Таким чином, для обчислення інтеграла від $f(x)$ за допомогою MATLAB треба утворити масив вузлів поділу інтервалу

>> *x*= *a* **:** *Dx* **:** *b* – *Dx*,

де *Dx*=(*b* – *a*)/*n* крок інтегрування, обчислити значення функції в них (весь масив значень в MATLAB утворюється лише однією командою *F*=*f*(*x*)), застосувати внутрішню функцію сумації *S*=*sum*(*F*) і, зрештою, помножити на *Dx*, тобто

*>> Integral=Dx * S.*

Приклад чисельного (відрізняти від аналітичного!) обчислення інтегралу дивіться в [2].

Відома також формула трапецій [2,15,16,20,34-39]; вона обраховує інтеграл точніше, аніж цей **метод прямокутників**. Та нам зараз важливо не це.

2.4.2. ПОХИБКИ ПРИ ОБЧИСЛЕННІ ІНТЕГРАЛІВ

Користувач того чи іншого методу обчислень повинен давати собі (і можливим замовникам) раду щодо точності результату. При обчисленні інтегралів маємо нагоду показати, з чого може складатись похибка обчислень і як це враховувати в роботі.

Якщо метод обчислення (в даному випадку – інтегралів) обрано вірно, то має бути принципова можливість наближатись до "вірного значення" як завгодно близько шляхом "кращого" обрання параметрів методу. В останньому прикладі таким параметром методу інтегрування слугувало число *N*, кількість поділів проміжку інтегрування або – це зручніше – крок чисельного інтегрування $\Delta x = {(b-a)}/{N}$.

Похибкою (*абсолютною*) ***розрахунку*** R називають різницю між наближеним значенням, отриманим чисельним алгоритмом, і точним значенням. Абсолютна похибка має дві складові – одна виникає внаслідок заміни інтегралу сумою (*похибка методу* R_1), друга зумовлена похибками обчислення значень f_i і діями над ними (*похибка округлення* R_2). Очевидно, що для грубих f_i результат не буде точним навіть для найкращого методу інтегрування. Така *неусувна похибка* оцінена як $|R_2| \le (b-a)\Delta f$, де Δf – абсолютна похибка обчислення підінтегральної функції [38].

А як обрання методу впливає на похибку? Чи гарантує метод, що зменшення Δx зменшує похибку? Для методів чисельного інтегрування доведено [38], що точність отриманих формул дійсно покращується, якщо функція $f(x)$

поводить себе "добре" у тому сенсі, що її перша або друга похідні обмежені по модулю, тобто $|f'(x)| \le M_1$, $|f''(x)| \le M_2$ на всьому проміжку інтегрування.

Як швидко зменшується похибка методу R_1 із зменшенням Δx? Доведено, що похибка формул лівих або правих прямокутників зменшується "зі швидкістю" $\frac{1}{2} M_1 (b-a) \cdot \Delta x$ (першого степеня Δx); похибка формули середніх прямокутників і формули трапецій зменшується як $k M_2 (b-a) \cdot \Delta x^2$ (другого степеня Δx), де $k = \frac{1}{24}$ для першої і $k = \frac{1}{12}$ для другої формул, тобто набагато швидше. Наведені оцінки часто дозволяють визначити наперед кількість точок поділу n, яка забезпечує потрібну точність. Часто використовують таке практичне правило:

- інтеграл обчислюють двічі, з n і з $2n$ точками поділу;
- порівнюють обидва результати I_n і I_{2n} і оцінюють точність як кількість спільних десяткових знаків, або кількість нулів після десяткової коми у різниці $I_n - I_{2n}$.
- При цьому точність обчислення проміжних величин, підінтегральної функції f_i, має бути на один-два порядки краще.

Якщо таке виконується – добре, розрахунок є остаточним. Якщо не виконується – продовжуємо розрахунок із збільшеним n, тобто I_{4n}.

Слід мати на увазі, що інженер інколи стикається і з "поганими" інтегралами, для яких названі методи не дають точного результату. Такими є, наприклад, інтеграли з теорії рядів Фур'є $\int_a^b f(t)\sin(kt)\,dt$, оскільки для великих k підінтегральна функція багато разів змінює знак на проміжку

інтегрування, отже $f'(x)$ і $f''(x)$ майже необмежені. Для таких інтегралів розроблені спеціальні методи.

Лабораторні роботи № 1 – № 4 „тренують" навички, викладені у двох розділах, які використовуються у наступних розділах з програмування.

2.5. КОНТРОЛЬНІ ПИТАННЯ ДО РОЗДІЛУ 2

1. На які питання щодо „якості" (коректності) ви маєте відповісти, отримавши для розв'язання (а можливо – і для самостійного дослідження) ту чи іншу математичну задачу?
2. Що таке СЛАР? Які методи розв'язування СЛАР ви знаєте? Чому СЛАР мають фундаментальне значення для математичного моделювання фізичних, технічних, економічних явищ? За яких умов СЛАР не має розв'язку? Має нескінченну кількість розв'язків? Лише два розв'язки? Лише єдиний?
3. У чому полягає метод Крамера розв'язування систем лінійних алгебраїчних рівнянь? Оберіть будь-яку СЛАР з додатку 1 та розв'яжіть методом Крамера.
4. У чому полягає метод Гауса розв'язування систем лінійних алгебраїчних рівнянь? Оберіть будь-яку СЛАР з додатку 1 та розв'яжіть методом Гауса.
5. У чому полягає різниця між чисельними і символьними (аналітичними) діями у MATLAB? Методи Крамера та Гауса – чисельні чи аналітичні?
6. Що називають алгебраїчним рівнянням з однією невідомою? Скільки розв'язків може мати математична модель якогось явища у формі многочлена *n*-го степеня? У чому фундаментальність такого роду математичних задач?
7. Розв'яжіть рівняння $x^4 - 5x^3 + 5x^2 + 5x = 6$ двома методами MATLAB – символьним та чисельним (розділ 1).
8. Що називають трансцендентним рівнянням з однією невідомою? Чи існують якісь універсальні методи

розв'язування таких задач? Скільки розв'язків може мати таке рівняння?

9. Отримавши якесь трансцендентне рівняння $\varphi(x)=0$, як ви почнете його досліджувати? Чи може бути корисною побудова графіка функції $y=\varphi(x)$? На що звертати увагу для визначення коренів?
10. Оцінивши значення того чи іншого кореня грубо, наближено, яким алгоритмом можна уточнити його більш точно? З якою точністю?
11. У чому полягає метод "поділу пополам"? Навіщо висувають ***умову збіжності*** чисельного методу і чи завжди збіжність гарантована?
12. Проведено експерименти із киданням кульки під кутом до горизонту. За умов *r=6*; *V0=10*; вітер відсутній змінювали кут „стрільби" *Alfa* від 0^o до 90^o і отримали наступні залежності дальності *L* і висоти *h* польоту:

Alf	10^o	20^o	30^o	40^o	50^o	60^o	70^o	80^o	90^o
L	*3.08*	*5.24*	*6.57*	*7.11*	*6.91*	*6.01*	*4.49*	*2.42*	*0*
h	*0.14*	*0.53*	*1.08*	*1.73*	*2.41*	*3.03*	*3.54*	*3.88*	*3.99*

Знайдіть емпіричні залежності: (**а**) $L=f(Alfa)$;

(**б**) $h=f(Alfa)$. В останньому випадку винайдіть кут *Alfa*, за яким досягається найдовший політ.

13. Чому не вистачає аналітичних методів обчислення інтегралів, чому виникає потреба у їх чисельному обчисленні? Які інтеграли не можна обчислити „аналітично"? Які ви знаєте методи чисельного обчислення визначених інтегралів? Як оцінюють точність (похибку) обчислення інтегралу?

Якщо не змогли відповісти на більшість питань – радимо проробити все з початку

3. ОСНОВИ ПРОГРАМУВАННЯ - З MATLAB

Система MATLAB дозволяє користувачеві надавати окреме ім'я певним ланцюжкам команд таким чином, що це ім'я починає працювати як команда MATLAB, виконання якої в командному рядку викликає виконання того самого ланцюжка команд. Команди, створені користувачем, називаються ***програмами***, процес їхнього створення – ***програмуванням***. Якщо користувач створив певну команду (програму), то та в свою чергу може бути використана в якійсь більш складній програмі. Програмування, таким чином, дозволяє складні, розвинені й довготривалі обчислювальні процеси складати з більш простих.

Система MATLAB розрізняє програми двох типів – програми-функції (*functions*) і програми-сценарії (*scripts*), які ми для скорочення будемо називати *функціями* і *скриптами*. Для їхнього створення MATLAB має спеціальний текстовий редактор, який зберігає написані програми у файлі із вказаним користувачем іменем і з розширенням "***.m***". (Можна використовувати і якийсь інший текстовий редактор, та це навряд чи зручно та доцільно).

Даний розділ присвячено основам програмування в системі MATLAB. Структурне програмування, яке вивчатимемо в даному розділі, нічим принципово не відрізняється від такого програмування іншими алгоритмічними мовами, проте – MATLAB дозволяє швидше досягти певної майстерності і легко створювати захоплюючі програми.

Дано приклади функцій і скриптів, рекомендації щодо роботи з вбудованим текстовим редактором MATLAB (його називають ***редактором m-файлів***), команди логіки ***if***, ***elseif***, ***else***, ... ***end*** і команди управління обчислювальним процесом ***for . . end***, ***while . . . end***. Запропоновано лабораторну роботу **5**, яка дозволяє закріпити навички програмування. Наведені приклади і завдання цієї лабораторної роботи спираються на методи, викладені у попередньому, другому розділі.

Щоб запустити будь-яку програму MATLAB, необхідно набрати її ім‘я в командному рядку командного вікна MATLAB і натиснути клавішу *Ввод*. Приклади команд:

>> *visualize* >> *step1*(*z*)
>> *step_pi*(*t*) >> *MyDeterminant*(*A*),

де *visualize*, *step1*, *step_pi* і *MyDeterminant* – програми (команди), створення яких надано нижче. Зараз важливо сказати, що кожна програма для стороннього користувача (або для самого її автора після тривалого проміжку часу) – це "чорна скринька", яка створює (говорять, "*видає*", "*повертає*") певні величини (***вихідні*** дані) на підставі даних, які вже існують в математичному середовищі MATLAB або спеціальним чином вказані цій програмі (***вхідні*** дані).

Програма, яка видає дані *y*, *x1* та інші на підставі вже створених в математичному середовищі даних і не потребує ніяких вхідних даних, називається *скриптом*. Команда (програма) *visualize* є скриптом, бо не має вхідних параметрів. Програма, яка може використовувати вже створені дані, але потребує певних вхідних даних ***x***, *a*, ***b***,…, називається *функцією* (або *m-функцією*). Наприклад, *step1*(*z*) та інші є саме функціями, бо у них є якийсь вхідний аргумент у дужках. Різницю між двома видами програм MATLAB може пояснити рис. 3.1.

Таким чином, *скрипт* і *функція* є двома можливими різновидами програм MATLAB, яким надані певні імена. Вони зберігаються у файлах з саме такими іменами. Розглянемо правила створення таких програм.

3.1. ПРОГРАМИ-СЦЕНАРІЇ

Скрипт, тобто програма-сценарій, – це послідовність команд середовища MATLAB, яка зберігається у файлі з певним іменем і розширенням "***.****m*". Створювати скрипти доцільно тоді, коли необхідно неодноразово повторювати відповідну

послідовність команд.

Перш за все слід викликати редактор *m*-файлів. Для цього в меню MATLAB мишею обираємо

File \ New ► M-file,

або натискуємо іконку  . З‘являється вікно редактора *m*-файлів з порожньою сторінкою майбутнього *m*-файлу, який поки що озаглавлений "*Untitled*". Після набору кількох перших рядків майбутньої програми, новий *m*-файл слід зберегти. Для цього необхідно мишею у меню обрати ***File\Save*** або натиснути на іконку „Дискета”  , або – на клавіатурі комбінацію "гарячих клавіш" <*Ctrl* + *S*>. У вікні "***Save file as*:**", яке з’явиться, у першому рядку "*Ім’я файлу*" замість запропонованої назви ***Untitled.m*** ввести бажане ім‘я, наприклад ***script1.m***. Рис. 3.2 пояснює процес збереження й найменування програми.

Як же назвати програму? Дуже важливе питання! „Як назвеш, так і працювати буде....” Дозволяються будь-які імена, що

(*i*) не починаються з цифри;

(*ii*) не містять знаків арифметичних дій, крапок і коми (так, знак „-”, мінус, заборонений, але ж можна використати „_”, підкреслення);

(*iii*) не містять „заборонених” знаків @, **(**, **]**, **{**, **!** тощо.

В останньому Ви вільні. Однак, імена, що надають студенти, не завжди раціональні:

lab32, igorko3, zzz2, sop, positiv, asd1, z4p4.

Доречно відображати у цих назвах призначення програми, її версію. Наприклад:

solveSLAE, My2Taylor, Flower, SunSystem2.

Важливо: у назвах програм користуватися виключно латинськими літерами! MATLAB починаючи з v.7 розрізняє великі й малі літери (*SunSystem* і *sunSystem* – різні програми).

Далі можна продовжувати набір програми, час від часу зберігаючи її ( або <*Ctrs* + *S*>). Оскільки ім‘я вже надане, MATLAB зберігає нові набрані дані "мовчки", без

посередництва вікна "***Save file as*:**" ("*Зберегти як*:"). Лише у випадку, коли ви захочете змінити назву програми (файлу), слід через меню натиснути мишею "***File* \ *Save as . . .***" і у вікні "***Save file as*:**" замінити запропоновану назву файла на бажану.

Отже, пишемо програму-скрипт. Формат написання програм, взагалі кажучи, довільний, проте кожному слід виробити свій власний стиль і порядок, враховуючи надані нижче рекомендації.

1. В першому і, можливо, в кількох наступних рядках слід написати ***коментар*** – власний текст після знаку **%** на початку кожного рядку. В коментарі (у редакторі *m*-файлів вони відмічені, як правило, зеленим кольором) доцільно коротко записати, для якої задачі створено програму, навести певні формули і звідки їх взято, навести прізвище автора програми і дату її створення, наприклад:

% *Програма перетворення декартових координат* {x,y,z}
% *у сферичні* {*rho, theta, r*}
% *за формулами rho*= …, *theta*=…
% (*формули взято з книги* [*Курс вищої математики*])
%
% *Розробник програми Ваше прізвище та ім'я*
% *Дата створення . . . .*

Кожного разу, натискуючи на клавішу *Ввод* для переходу на новий рядок, редактор *m*-файлів нумерує рядки тексту, так що наведені рядки мають номери від *1*, *2*, *3* і т.д., рис. 3.1. У ***лістінгах***[18] програм, що наведено у цій книжці, ми теж нумеруємо, аби зручніше було пояснювати програму, але **Вам нумерувати рядки програми під час її створення – не треба!**

Написані на початку скрипту коментарі відіграють доволі важливу роль, допомагаючи сторонньому користувачеві узнати (або самому авторові – пригадати)

[18] ***Лістінг*** (англійською *Listing*) – це просто запис тексту програми у вашому зошиті або у файлі, його роздруківка. Такий термін використовується в інформатиці.

призначення даної програми. Припустимо, ви назвали свою програму ***script1*** (редактор *m*-файлів надасть їй розширення „*m*" автоматично). Якщо надалі в командному рядочку MATLAB вводити

>> help ***script1*** **,**

то отримуємо записаний під коментарем текст (рис. 3.1). Таким чином, щойно розпочата програма ще нічого не вміє робити – а допомогу, Help, вже дає!

2. Далі слід написати **тіло програми** – послідовність команд системи MATLAB, яка власне й утворює програму. Наведемо приклад скрипту візуалізації трьох функцій (побудови графіків), які, припустимо, вже створені в MATLAB-середовищі, називаються *f0*, *f1* і *f2* і приклад яких наведено далі. В командному вікні виконуємо:

```
>> Xbegin=0; Xend=10;  N=1000; % Початок і кінець табуляції
>> Step= (Xend – Xbegin)/N;          % Крок табуляції
                                                        (3.1)
>> X= Xbegin : Step :  Xend ; % Вектор значень аргументу
>> F0=f0(X); F1=f1(X); F2=f2(X); %Вектори значень 3 функцій
>> visualize    % А це – Ваша перша власноруч створена програма
```

Коментарі до кожної команди можна не писати, але вони корисні – пояснюють призначення певних дій: у першому рядку задаються початок і кінець відрізку, на якому будемо обчислювати значення функцій (табулювати) а також кількість точок поділу *N*; далі обчислюємо крок табуляції *Step*; у третьому рядку створюємо вектор значень аргументу *X*, який складається з *N+1* елементів, починаючи з числа *Xbegin* і закінчуючи числом *Xend*; у четвертому рядку обчислюються три вектори, що містять значення функцій *f0*, *f1* і *f2* для відповідних аргументів (маємо на увазі, що функції *f0*, *f1* і *f2* MATLAB "знає": або ми пишемо конкретні функції *sin*, *tan*, *exp* тощо, або вже створено файли *f0.m*, *f1.m* і *f2.m*). Команда ***visualize*** в останньому рядку раніше була невідома MATLAB, але ми попередньо створили файл *visualize.m* з наступним змістом:

Лістінг 3.1А (*m*-файл *visualize.m*)

```
% Програма побудови трьох графіків функцій (візуалізації)
```

```
% для їхнього співставлення між собою
%Copyright Gayev Ye.A.
figure; plot(X, F0, 'r',   X, F1, 'b',   X,F2, 'g');
title('Співставлення графіків трьох функцій')
grid on;
xlabel(' Незалежний аргумент x '); xlabel(' Функції від x ')
% Кінець програми visualize.
```
(3.2)

Прочитавши послідовність команд (3.2) з означеного файлу, MATLAB виконуватиме її кожного разу як єдину команду *visualize*. Відмітимо, що потрібні для її роботи дані *x, F0, F1, F2* MATLAB бере серед раніше створених даних у своєму середовищі. Якщо скрипт створює якісь нові дані, то вони поступають у середовище MATLAB, як показано на рис. 3.2, і стають доступними всім наступним командам.

Увага: коментарі з кириличними літерами (українськими або російськими) іноді спричинюють помилки у **MATLAB v.6.***. Хоча цього недоліку вже позбавлений **MATLAB v.7.***, все одно радимо писати коментарі або англійською мовою, або латинськими літерами.

Інші приклади скриптів надають програми *Welcome*, *Helicopter*, *Ball* (у додатку). Красивою є програма-скрипт годинник (*Clock*, рис. 0.1), створена студентом.

3.2. ПРОГРАМИ-ФУНКЦІЇ

Програми-функції, як і скрипти, створюють за допомогою редактора *m*-файлів і зберігають в файлі з певним іменем і розширенням *m* (редактор сам дописує це розширення), але їх структура трохи відрізняється від структури скриптів.

1. Першим рядком програми-функції має бути текст з ключовим словом *function* вигляду

function ***ВихіднийАргумент*** =*FunctionName*(***ПерелікВхіднихАргументів***)

де *FunctionName* – довільне ім'я, яке дозволене в MATLAB (див. 3.1). Рекомендовано зберігати програму-функцію у файлі з тим самим іменем *FunctionName*.

2. Наступні рядки програми доцільно заповнити коментарями на зразок наведених вище. Вони мають нагадувати користувачеві потрібну інформацію щодо функції, коли у командному рядочку ввести

>> help FunctionName

3. "*ПерелікВхіднихАргументів*", у якому вхідні аргументи перераховані через кому, використовується у наступному **тілі програми**. Цей перелік може виглядати таким чином:

FunctionName(*x1*, *x2*, *x3*)

4. Тіло програми-функції призначено для утворення нових змінних і надання їм певних значень за допомогою математичних операцій. Вказані аргументи мають бути використані в тілі програми для створення нових змінних, наприклад,

Sum=x1 + x2; A= sqrt(*x3*)*;* і так далі. . .

5. Наприкінці важливо надати потрібного значення ***Вихідному Аргументу***. Останній може бути одним іменем, наприклад, ***y***, і тоді перший рядок виглядає

function ***y*** = *FunctionName*(*x1*, *x2*, *x3*) ,

і цій змінній наприкінці програми обов'язково надається те чи інше значення, наприклад

y =*Sum* * *A*;

Бувають випадки, коли вихідний аргумент може складатись з кількох змінних, взятих у квадратні дужки, наприклад:

function **[** ***y1, y2*** **]** = *FunctionName*(*x1*, *x2*, *x3*) .

Тоді їм усім перед виходом з програми треба присвоїти ті чи інші значення, наприклад

y1 =*Sum* * *A*; ***y2***=*Sum* + *sqrt*(*A*);

Коли *m*-функція вже виконана і ми з неї повернулися до MATLAB-середовища, то – нагадаємо – усі змінні, що утворювалися (*Sum* і *A* у наведених прикладах) **забуваються**!

Всі ці загальні положення розберемо на прикладах.

Приклад 3.1. Побудувати функції, які будуть репрезентувати „***частинні суми ряду Фур'є***"

$$\frac{4}{\pi}\left(\frac{\sin x}{1}+\frac{\sin 3x}{3}+\frac{\sin 5x}{5}+\frac{\sin 7x}{7}+\ldots\right), \qquad (3.3)$$

який, як відомо [32,36,48], на відрізку $x\in[-\pi,\pi]$ наближує розривну функцію

$$f(x)=\begin{cases}-1, & якщо\ x\in[-\pi,0]\\ 1, & якщо\ x\in(0,\pi)\end{cases}. \qquad (3.4)$$

З такою задачею ми вже стикалися у Лабораторній роботі 1; там було надано зразок „ручного" виконання аналогічної роботи. Тепер можемо автоматизувати ті обчислення!

Програма-функція для обчислення математичної функції $f_1(x)=\frac{4}{\pi}\frac{\sin x}{1}$ (перша частинна сума ряду) виглядає наступним чином:

```
function F=G1(x)
% Перша частинна сума ряду Фур'є
%   4*(sin(x)/1 +sin(3*x)/3 +sin(5*x)/5  +...)/pi
%
% Copyright Гаєв Є.О.
F= 4*(sin(x)/1)/pi;
```

Програма функції $f_2(x)=\frac{4}{\pi}\left(\frac{\sin x}{1}+\frac{\sin 3x}{3}\right)$ (друга частинна сума ряду) має вигляд:

```
function F=G2(x)
% Друга частинна сума ряду Фур'є
%   4*(sin(x)/1 +sin(3*x)/3 +sin(5*x)/5  +...)/pi
%
% Copyright Гаєв Є.О.
```

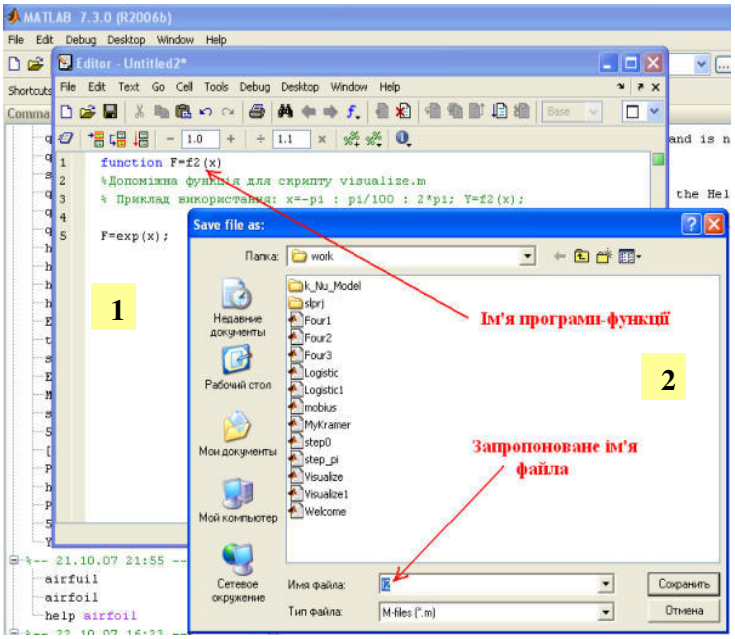


Рис. 3.1. Початок створення власної програми у редакторі *m*-файлів: 1 – вікно редактора *m*-файлів; 2 – вікно збереження програми. Скрипту, що створюється, давайте будь-яке дозволене ім'я; ім'я ж *m*-функції має співпадати з іменем файлу!

$$F = G1(x)+4*(sin(3*x)/3)/pi;$$

Утворивши файли *G1.m*, *G2.m* і, аналогічно, *G3.m* з аналогічним змістом, ми навчаємо MATLAB "розуміти" ці функції як його власні, внутрішні. Саме тому друга програма і працює, що вона посилається на *G1.m* як на внутрішню функцію. В свою чергу, третя програма викликає другу, і т.д. Можемо тепер:

(*i*) обчислити значення новостворених функцій для будь-якого аргументу, навіть для векторного;

(*ii*) обчисливши масив аргументів *x* і масив відповідних значень функції *G1* або *G2* командою *plot* можна побудувати їхні графіки;

(*iii*) застосувати також команду автоматичної побудови графіку *fplot*.

Ось приклад використання новостворених функцій для побудови графіків з командного вікна:

```
>> fplot('G1(x)', [-2*pi 2*pi],'b')
>> hold on; fplot('G2(x)', [-2*pi 2*pi],'g')
>> hold on; fplot('G3(x)', [-2*pi 2*pi],'r')
```

– отримаємо графіки першої (синя крива), другої (зелена), третьої (червона) і т.д. частинної суми ряду (3.3). Це – „автоматизація", одна з можливих, завдання 15.2 лабораторної роботи 1. Інші можливі підходи наведено нижче.

Увага: аби функція працювала для векторних аргументів, не забувайте пересвідчитись, що кожна операція тіла функції правильно сприймає вектори, тобто де потрібно – використано "операції із крапкою"!

Вважаємо створеними аналогічні функції від аргументу x, а саме *G4.m*, *G5.m* і т. д. до *G10.m*, що являють собою відповідні частинні суми ряду (3.3) до десятої частинної суми включно. Цікаво дізнатися, чи дійсно вони наближають розривну граничну функцію (3.4) і наскільки близько. Раніше, у лабораторній роботі № 1, ми не могли побудувати їхні графіки програмно, лише „від руки".

Звернемося тепер до програмування функції (3.4).

Приклад 3.2. Запрограмувати розривну функцію (3.4) для відрізку $x \in [-\pi, \pi]$.

Наводимо програму (*Лістинг*[19]), що частково вирішує проблему (далі зрозуміло, чому лише "частково"). Дамо функції (3.4) ім'я *step*(*x*), враховуючи її вигляд.

Лістинг 3.2А (*m*-файл *step0.m*)

```
function  F=step0(x)
% Функція ступінчата з рівнянням  (3.4)
%              I = -1;   якщо x \in [-pi, 0]
%   F(x)=
```

[19] Нагадаємо, що Лістинг – це „роздруківка" тексту програми.

```
%            \ =  1,    якщо x \in (0, pi]
%
% Copyright Гаєв Є.О.
   if x<=0
      %   disp( ' SubProgram X<0' )
      F=-1;
   else
      %   disp( ' SubProgram X>0' )
      F=1;
   end
```

Перевіримо, як працює і що може робити щойно отримана функція середовища MATLAB *step0*:

1. За командою з командного рядочка

```
>> help step0
```

отримуємо допомогу у вигляді

Функція ступінчата з рівнянням (3.4)

$$F(x)=\begin{cases} -1, & \text{якщо } x \in [-pi,\ 0] \\ 1, & \text{якщо } x \in (0,\ pi] \end{cases}$$

2. Програма дозволяє одержати значення функції (3.4) для будь-якого поодинокого значення аргументу. Наприклад,

```
>> x=2.1; y= step0(x)
   y=1
```

Або

```
>> x=–2.1; y= step0(x)
   y=–1
```

В той же час, ця функція працює лише для скалярного (поодинокого) аргументу. Для вектора *t= - 3 : 3* замість, відповідно, семи значень маємо *step0*(*t*) *=1* (тобто значення лише для останнього елементу вектора *t*)! Це не дозволить побудувати графік цієї функції командою *plot*, проте,

3. команда

```
>> fplot('step0',[-5 5 -1.1 1.1])
```

потрібний графік побудує. (Спробуйте! Подумайте ще раз над різницею між двома командами *plot* і *fplot*).

Зауваження 1: розглянута проста програма включає невідомі ще читачеві команди середовища MATLAB. Так, зустрічаємо команду *disp*, корисну для налагоджування програм. Про неї, та чому вона "закоментована" знаком % (тобто система все одно її в обчисленнях не враховує) розповімо в розділі 3.4.

Зауваження 2: остання команда *fplot* будує потрібний графік (3.4) лише на інтервалі $x \in [-\pi, \pi]$. Аби узагальнити цю розривну функцію на весь числовий ряд $x \in (-\infty, \infty)$ з періодом 2π потрібні нові оператори *for*.... *end*, розділ 3.3.2.

Таким чином, маємо вже приклади скриптів і *m*-функцій. Так чим же відрізняються два вида МАТЛАБ-програм? Це узагальнює наступна таблиця:

(*i*) Скрипти не мають ні вхідних аргументів, ні вихідних (приклад *Welcome* далі); *m*–функції обов'язково мають вхідні і вихідні (результат роботи програми-функції) аргументи (приклад: *step0*(*x*)).

(*ii*) Скрипти „бачать" усі змінні, що вже створені у MATLAB-середовищі. Так само, з середовища „видно" усі змінні, що створюються у скрипті, і інші програми можуть їх використовувати. На відміну від цього, *m*-функції „не бачать" жодної змінної у MATLAB-середовищі, окрім тих, що підставляються у вхідні параметри. Аналогічно, усі змінні, створені у тілі *m*-функцій, окрім вихідних параметрів, „не видні" з MATLAB-середовища і тому не можуть використовуватись іншими програмами. (Це зручно, бо розробник програми може не перевіряти, чи вже є таке-то ім'я десь зовні чи у якійсь іншій програмі. Якщо ж ;виникла потреба у якихось зовнішніх змінних – звертайтеся до правила *iv*).

(*iii*) Файли з *m*-функціями можуть зберігати у собі також і підпрограми, до яких вони звертаються (приклад – Лістинг 3.3 нижче). Якщо ж у скрипті є потреба звернутися до якоїсь допоміжної програми (тобто, до підпрограми), то її треба записати як окрему програму в окремому *m*-файлі.

(*iv*) Якщо треба порушити правило (*ii*) відносно лише окремих змінних, скажімо *x1*, *a*, *Diss*, то їх треба перелічити за допомогою оператора *global*:

```
global x1  a  Diss
```

Всі ці правила, що відрізняють обидва типи програм, пояснює рис. 3.2.

Для закріплення навичок програмування, пропонуємо дві задачі для самостійної роботи (із зразками програм).

Задача 1. В 1.7.1 та у лабораторній роботі 4, завдання 3, вже роз'яснювали сутність чисельного диференціювання, та обчислення проводили вручну. Зробіть програму *MyNumDiff*_1, на вхід якої подається якась аналітична функція (наприклад *sin*(*x*)**x*), інтервал для її аналізу (наприклад [*a*, *b*]) і ціле число розбиттів цього інтервалу *N*. А на виході цієї програми треба отримати таблицю наближених значень похідної вхідної функції в точках розбиття і графіки для порівняння наближеної і точної (аналітичної) похідної.

Наш зразок програми *MyNumDiff*_1 див. у Додатку 3. Пізніше буде створена аналогічна програма *MyNumDiff*_2, що таке саме робить у діалоговій формі. Ось який науковий висновок можна зробити за допомогою цієї програми: чим більше *N*, точок поділу інтервалу $[a,b]$, на якому функція досліджується, тим ближче співпадають аналітичний і чисельний розрахунок похідної, рис. 3.3.

Задача 2. Аналогічно, створити програму *MyNumInt*_1, що порівнює аналітичне і чисельне знаходження первісної для заданої функції *F* (наприклад тієї самої *F*= *sin*(*x*)**x*) на інтервалі [*a*, *b*] і цілому числі розбиттів *N*.

Не поспішайте дивитися у відповідь у Додатку! Пізніше буде створена аналогічна програма *MyNumInt*_2, що таке саме робить у діалоговій формі.

3.3. ЕЛЕМЕНТИ СТРУКТУРНОГО ПРОГРАМУВАННЯ

Ті програми, скрипти і *m*-функції, з якими Ви вже ознайомилися, мали *лінійну структуру*, тобто використані у

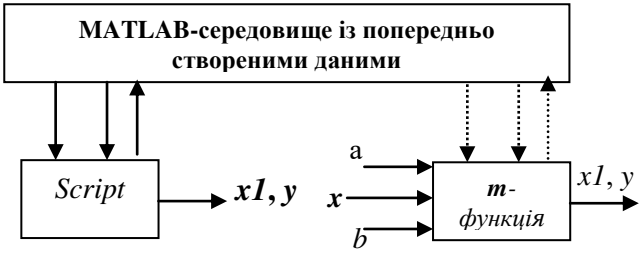


Рис. 3.2. Пояснення щодо різниці між двома типами програм, *скрипт* і *m-функція*: суцільні стрілочки – безпосередній зв'язок; бліді стрілочки – зв'язок, опосередкований описом *global*; a, *x b* – вхідні параметри; *x1*, *y* – вихідні результати.

них оператори (команди) виконувалися один за одним, наступний за попереднім. Однак вже на зорі програмування виникла потреба порушувати цей лінійний порядок, пропускати виконання деяких операторів, повертатись до попередніх операторів, багаторазово виконувати певні групи операторів (цикли) тощо. Значне ускладнення програм – до тисячі, мільйона і більше рядочків! – висунуло вимоги до їх „правильної", зрозумілої організації. Так виникло ***структурне програмування*** – використання певних „цеглинок", з яких складаються необмежено великі програми. Саме таке сучасне програмування починаємо тут вивчати. Антитезою структурного програмування є вже згадана лінійна організація програм та, з протилежної сторони, ***об'єктно-орієнтоване програмування***.

Треба також розуміти, що MATLAB є *інтерпретатором* команд, так само як, припустимо, програмне середовище Basic і сучасні мови Java і C#, і не є ***компілятором*** з мови програмування. Це означає, що Ваші програми можуть виконуватись лише у MATLAB-середовищі, і не можуть розповсюджуватись як окремі незалежні *exe*-файли. Проте, „ще не вечір": в MATLAB є деякі можливості „покинути обмеження" компіляторів, які розробники програми розширюють у кожній наступній версії...

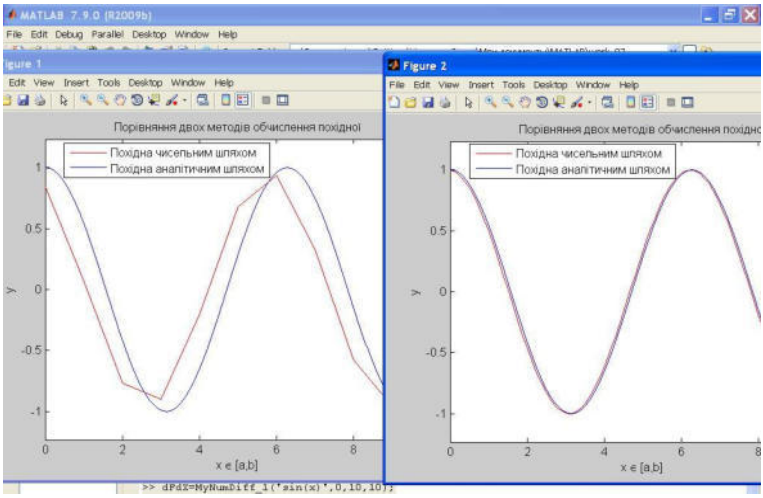


Рис. 3.3. Дослідження двох методів диференціювання, чисельного і аналітичного, програмою `MyNumDiff_2`: зліва кількість точок поділу *N=10*, праворуч *N=100*.

Все одно, вивчити **програмування у MATLAB** – дуже корисно з багатьох позицій:

1. з практичної точки зору (бо він використовується у тисячах університетів, дослідницьких лабораторій різноманітного профілю, конструкторських бюро і виробничих фірм);
2. з навчальної точки зору, бо воно дуже просте, не потребує багато зусиль, і, на відміну від інших мов програмування, за короткий час дозволяє зробити розвинені і цікаві програми;
3. з іншого боку, програмування в MATLAB включає майже все, що буде Вам корисне і в інших мовах програмування, всі конструкції цієї науки – різноманітні типи даних, парадигми програмування (структурне і об'єктно-орієнтоване програмування) тощо.

Методами структурного програмування, які Ви тут вивчите, розроблені доволі складні програми, що будуть запропоновані далі. Починаємо!

3.3.1. ОПЕРАТОРИ УМОВНОГО ВИКОНАННЯ

Оператори умовного виконання *if*, *else* та пов'язаний з ними ***elseif*** – важливий інструмент програмування, що дозволяє надати програмі риси штучного інтелекту. Загальна форма оператора умовного виконання є такою:

```
if Умова 1
      Група операторів 1
elseif Умова 2
      Група операторів 2                    (3.5)
else
     Група операторів 3
end
```

В складеному операторі (3.5) *Група операторів* – це послідовність будь-яких команд MATLAB (дії над величинами, побудова графіків тощо), а *Умова* – це якийсь математичний запис, що може бути ***вірним*** (*true*) або ***невірним*** (*false*). Ще кажуть, що *Умова* може приймати лише два логічних значення, відповідно ***1*** або ***0***.

Оператор (3.5) – це єдина структура, єдиний блок! Він завжди має починатися з ***if*** і закінчуватися ***end***, і працює наступним чином. Коли MATLAB, послідовно виконуючи всі попередні команди програми, дійшов до ***if***, з якого починається блок команд (3.5), то перш за все перевіряється *Умова 1*. Якщо вона має логічне значення ***1***, тобто ***вірно***, то MATLAB виконує *Групу операторів 1*, після чого приступає до операторів, що написані нижче за ***end*** (виходить з блоку). Всі інші умови і оператори (3.5) тоді ніякої ролі не відіграють.

Якщо *Умова 1* приймає значення ***0*** (тобто ***false***, ***невірно***), то MATLAB пропускає без всякої уваги *Групу операторів 1* і звертається до перевірки *Умови 2*. Якщо остання має значення ***1*** (***true***, ***вірно***), то виконується *Група операторів 2* і далі знову "*управління передається*" на оператор, наступний за ***end***. *Група операторів 3* в цьому випадку ніякої ролі не відігравала.

Проте, коли *Умова 2* також ***невірна*** (має значення ***0***, *false*), то *Група операторів 2* пропускається без уваги і

MATLAB виконує *Групу операторів 3*, а потім знов-таки "покидає оператор" (блок) ***if … end*** і виконує наступний за ним оператор у програмі.

З наведеного детального опису має стати зрозумілою і робота кожної із скорочених форм оператора умовного виконання (3.5):

```
if Умова
        Група операторів
end
```

```
if Умова
        Група операторів 1
else
        Група операторів 2
end
```

В усіх випадках кожне ***if*** повинно закінчуватись відповідним ***end***! Друга скорочена форма оператора умовного виконання пояснюється блок-схемою на рис. 3.4. З оператором у такій формі ми вже стикалися у програмі *step0*.

Увага: після ключового слова ***else*** ніякої умови не треба!

Редактор *m*-файлів автоматично розміщує складові цього та інших структурних операторів у вигляді "сходинок", аби покращити сприйняття тексту програми. Кожному

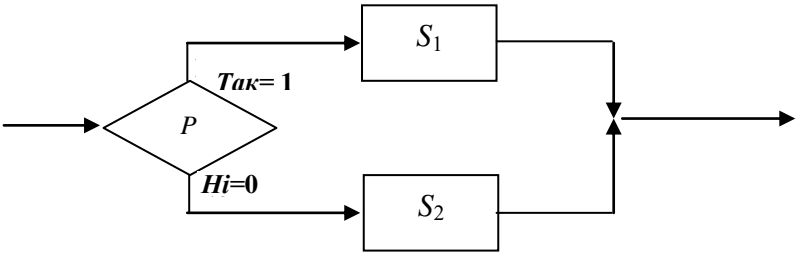


Рис. 3.4. Блок-схема, що пояснює оператор логіки
if *P ...* ***else*** ***end***

радимо дотримуватись такого ***правильного стилю програмування***! Крім того, редактор *m*-файлів „фарбує” ці та всі інші „ключові слова” ***if***, ***else***, ***end*** тощо у голубий колір. Тому, хоча і дозволяється використовувати будь-який інший текстовий редактор, та редактор *m*-файлів є найзручнішим.

Корисно також переглянути приклади, подані в *help* ***if***.

Наведена теорія дозволяє остаточно зрозуміти програму функції *step0*. Якщо десь у командному рядку або у програмі MATLAB зустріне вираз *step0*(*t*), то він звертається до своєї робочої області пам‘яті і шукає там *m*-файл із іменем *step0*. Не знайшов – видає інформацію

??? Undefined function or variable ' step0 '.
(невідома функція або змінна *' step0 '*)

Це означає, що зробили щось невірне; потрібно знайти і виправити помилку. Можливе пояснення і виправлення – див. розділ 3.4. Знайшов MATLAB потрібну функцію – замість аргументу програми *x* (*формального аргументу*) підставляє надану величину (*фактичний аргумент*), тобто *x*=*t*, і виконує запрограмовані дії:

- якщо надане значення *x* менше або дорівнює нулю, то вихідній змінній *F* надається значення *F= -1*, і з цим значенням виходить із "внутрішнього середовища команди" у загальне середовище MATLAB;
- якщо ж умова *x <= 0* була невірною, то вихідній змінній надається значення *F= 1*, що якраз і відповідає нижній гілці *x > 0* функції (3.4).

Умова, яка є аргументом операторів *if* або *elseif*, може бути **простою умовою**, що використовує один з можливих ***операторів відношення***

$$A == B, \quad A < B, \quad A <= B,$$
$$A > B, \quad A >= B, \quad A \sim= B^{20}. \tag{3.6}$$

де *A* і *B* – це довільні арифметичні вирази, яким програма вже надала числових значень. Тоді написані умови означають відповідно "*дорівнює*", "*менше*", "*менше або дорівнює*", "*більше*", "*більше або дорівнює*", "*не дорівнює*". Розглянемо, наприклад, останню умову *A~= B* ("*A не дорівнює B*"). Вона приймає значення ***1*** (***true***), якщо *A* і *B* дійсно такі, що $A \neq B$; проте, умова приймає значення ***0*** (***false***), якщо $A = B$.

[20] Цю умову та знак „~” називають „заперечення”.

Увага: існує принципова різниця між використанням знака „=” (дорівнює) в математиці і в інформатиці! **В математичних виразах** пишемо **„=”**, і це означає „*дорівнює*”. В інформатиці знак **„=”** означає власне „*значення праворуч заслати у клітинку пам’яті з іменем, що вказано ліворуч*”. А от **в Умовах пишемо „= =” (два знака поспіль!)**, і лише це означає „*дорівнює*”! Сутність інших форм **простої умови** (3.6) так само легко зрозуміти. Їх значенням може бути лише 0 або 1, як сказано вище.

Однак, ***Умова*** може бути і складною, складаючись з виразів (3.6) за допомогою ***операторів логіки***

Умова1 **&** *Умова2*, *Умова1* **|** *Умова2*, **~** *Умова1*,

які читаються відповідно

"*Умова1* **і** *Умова2*", "*Умова1* **або** *Умова2*", "**не** *Умова1*".

Це означає, що перша з написаних вище складних умов (її називають ***логічним множенням*** або ***AND***) приймає значення ***1*** лише тоді, коли *Умова1* і *Умова2* мають значення ***1*** одночасно. Друга складна умова (***логічна сума*** або ***OR***) приймає значення ***1*** у двох випадках: коли *Умова1* має значення ***1*** незалежно від *Умови2* і коли *Умова2* має значення ***1*** незалежно від *Умови1*. Третя складна умова (її називають ***запереченням*** або ***NOT***) приймає значення ***1*** коли *Умова1* має значення ***0***, і приймає логічне значення ***0*** коли *Умова1* має значення ***1***. Більш докладно співвідношення між вхідними і вихідним значеннями функцій логіки надає Таблиця 3.1.

Таблиця 3.1. Значення функцій логіки ***AND***, ***OR*** та ***NOT***

Значення *A* та *B*		*A & B*	*A | B*	*~ A*
1	***1***	***1***	***1***	***0***
1	***0***	***0***	***1***	***0***
0	***1***	***0***	***1***	***1***
0	***0***	***0***	***0***	***1***

Приклад 3.3. Зробіть самостійно програму привітання *Welcome*, яка буде Вас вітати в залежності від часу доби: скаже

- "*Доброго ранку*!" між п'ятою і дванадцятою годинами,
- '*Доброго дня*!' між 12 і 17, і так далі,
- і додасть '*Я порадила б Вам спати у цей час!*' після опівночі.

Коли зробите (чи ні) – подивіться наш приклад такої програми, Додаток 4. Це – ще один зразок скрипту. Далі будемо використовувати його як звичайну команду MATLAB.

У наведеному прикладі структура *if . . . end* включає у собі ще таку ж структуру *if . . . end*; остання називається вкладеною, перша – зовнішньою.

Наступний приклад показує доволі незвичні властивості MATLAB'івської логіки**.** Майже в усіх мовах програмування заборонено проводити арифметичні операції одночасно над чисельними і логічними величинами, бо останні не є числами *0* та *1*, а „особливими" величинами *false* і *true*. А от MATLAB дозволяє таке! І в цій арифметиці, зрозуміло

$$false * X = 0, \quad \text{і} \quad true * X = X,$$

$$false + X = X, \quad \text{і} \quad true + X = X + 1 .$$

Таке невеличкий „винахід" розробників MATLAB може інколи дуже спростити програмування. Як приклад, надаємо відому вже функцію *step0*(*x*), але вже з використанням цього „винаходу". Аби відрізняти, назвемо модифіковану функцію *step02*(*x*). Програма набагато коротше:

Лістінг програми step01.m

```
function  y=step01(x)
% Функція ступенчата за рівнянням (3.4)
%            / = -1;   якщо x \in [-pi, 0]
%     F(x)=
%            \ =  1,    якщо x \in (0, pi]
% Це та ж програма step0.m,
%але – з використанням MATLAB-логіки
```

```
% Copyright Гаєв Є.О.

y1=(x < 0);
y2=(x >=0);
y= - y1 + y2;
```

Приклади використання складних логічних умов подано далі; ще більше прикладів можна знайти в довідковій системі *Help* та у книгах [4-9,12-14,16-20].

Розглянемо й інші оператори, які часто використовують у структурному програмуванні.

3.3.2. ОПЕРАТОРИ УПРАВЛІННЯ

Існує кілька **операторів управління** обчислювальним процесом –

for … end, ***while… end***,

switch … case … otherwise … end,

try … catch … end, ***break***

та деякі інші. Тут зупинимось на двох перших, найбільш поширених операторах – ***for … end*** і ***while… end***. Якщо, з набуттям досвіду програмування, виникне потреба в інших операторах, познайомитись з ними можна у довіднику *Help*. У якості прикладів візьмемо програмування розривної періодичної функції, бо це дозволить просунутись у вже розпочатому аналізі збіжності частинних сум рядів Тейлора і Фур'є, див. лабораторну роботу 1 та формулу (3.3).

Структурі ***switch … case … otherwise … end*** присвячено розділ 3.7.3.

1. Приклади використання структури *for … end*

Оператори управління потрібні для організації циклів повторювальних обчислень, наприклад – дозволяють модифікувати програму *step0*, аби вона правильно працювала і для векторного аргументу. Для цього текст програми слід трохи змінити, як показано у наступному лістінгу, який радимо зберегти у *m*-файлі *step1*:

Лістинг 3.2Б (*m*-файл *step1.m*)

```
function  F0=step1(x)
% Така сама Step-функція, як  "Step0.m",
```

```
% однак модифікована на врахування векторного аргументу.
%%
% РозробникГаєв Є.О., 10.01.2001

L=length(x); % Знаходимо довжину вектора-аргументу x
for i=1:L  % Все наступне робимо для кожного з  елементів x
        if x( i )<=0
           F0( i )=-1;
        else
           F0( i )=1; % Значень F0 саме стільки, скільки L
        end
end
```

З розташуванням „по сходинках” програма читається легко. Розглянемо її роботу. Перш за все, програма *step1* використовує функцію *length* (див. розділ 1.4), що визначає кількість елементів у вхідному векторі *x*. Таку саму кількість елементів повинен мати вихідний вектор *F0* (він може називатись і *F*, значення не має).

Кожний з елементів вхідного вектора *x* аналізується по черзі у циклі *for* (7-й рядок програми) **…** *end* (13-й рядок програми). Такий оператор називають **оператором з визначеною кількістю циклів**. Змінна циклу ***i*** приймає спочатку значення *1*, обирає *x*(*1*), перший з елементів вектора *x* і аналізує його так само, як в програмі *step0*, надаючи вихідній змінній *F0*(*1*) відповідного значення *–1* або *1*. Потім таку саму роботу програма проводить для *x*(*2*) і надає певного значення *F0*(*2*), і так – для всіх *L* елементів вхідного вектора. Коли вся робота виконана, програма повертає управління в зовнішнє середовище MATLAB разом із змінною *F0*, яка стає вектором з *L* елементів і указана як вихідний аргумент програми *step1*(*x*).

Тепер можна використати і команду *plot* для побудови графіку функції (3.4):

```
>> t= -3*pi : .01 : 3*pi;  F=step1(t);
>>  plot(t,F);  axis( [ - 3  3  - 1.1  1.1 ] )
```

Відмітимо, що змінними *i* і *L* можна користуватись усюди "в середині" програми *step1*, однак значення їх, як і кожної

іншої змінної, створеної в тілі програми-функції, зникають, як тільки управління повертається до командного рядка. Це зручно, бо жодна із зовнішніх змінних, навіть з таким самим іменем, не впливатиме на роботу функції. Проте існують програмні засоби для порушення цього загального правила (див. правило (*iv*) в розділі 3.2 та 3.3.3.).

З численними повтореннями майже однакових операції ми стикалися, коли „вручну" обчислювали розв'язки СЛАР методом Крамера чи то методом Гауса, розділ 2.1. То було б корисно, аби знайти „алгоритмічні" дії для наступного програмування. Тепер можете подивитись програми, які „автоматизують" метод Крамера (додаток 5) і метод Гауса поступового виключення невідомих(додаток 6). Такі програми дають додаткові приклади використання структури ***for … end***. Зверніть увагу на те, що у середині структури ***for … end*** (її називають ще *циклом*) може бути така ж структура ***for … end*** (*внутрішній цикл*).

2. Приклад використання структури *while… end*

Щойно створена програма *step1* будує функцію (3.4) фактично на інтервалі $x \in (-\infty, \infty)$, тобто надає значення $F = -1$ для всіх $x \le 0$ і значення $F = +1$ для всіх $x > 0$. Однак вона мала б задаватись рівнянням (3.4) лише на інтервалі $x \in [-\pi, \pi]$ і бути подовженою періодично (з періодом $T = 2\pi$) на всю числову вісь. Програма-функція такої математичної функції буде більш складною.

Приклад 3.3. Побудувати функцію (3.4), періодичну на усій числовій осі (див. графік).

Перш за все слід **виробити алгоритм** знаходження значення $F(x)$ для довільного значення аргументу x. Пропонується такий алгоритм: якщо нам надано якесь число x, то перевіряємо його належність до інтервалу $[-\pi, \pi]$; якщо x належить цьому інтервалу – відповідна функція *step0* або *step1* надасть $F(x)$ потрібного значення *-1* або *1* – це вже відома задача. Нехай тепер x не належить інтервалу $[-\pi, \pi]$. Треба визначитись, якого значення набуде F, якщо функцію (3.4) періодично пересувати на 2π вліво чи вправо. У цьому,

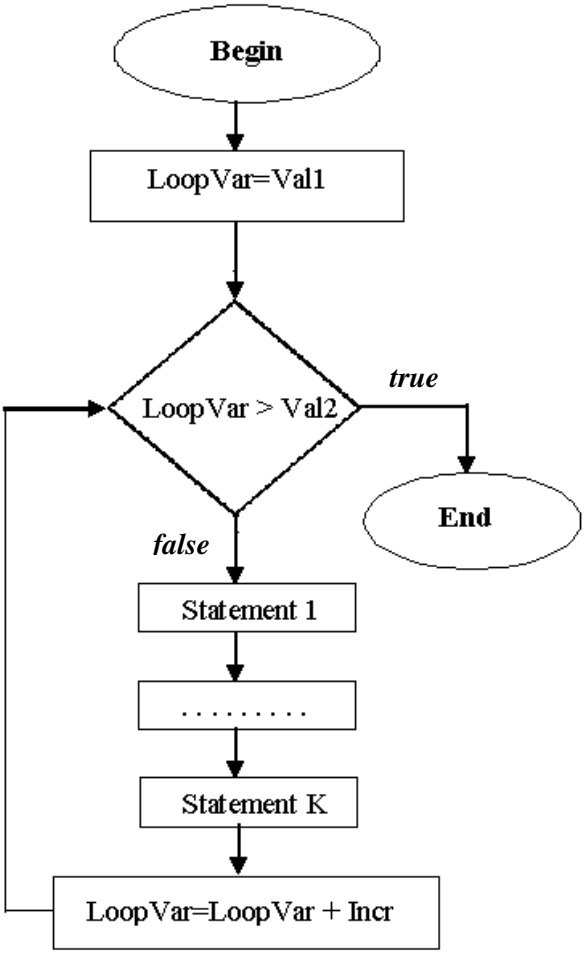


Рис. 3.5. Блок-схема циклу *for … end*.

фактично, і полягає алгоритм: якщо *x* додатній, то віднімаємо від нього період 2π і перевіряємо, чи потрапив результат в інтервал $[-\pi,\pi]$. Не потрапив? Ще раз віднімаємо і перевіряємо належність результату до інтервалу. Зрозуміло, що рано чи пізно потрапить, але кількість операцій віднімання заздалегідь невідома. Як тільки результат

віднімання потрапив у контрольний інтервал, то визначимо значення $F(x)$ вже створеним алгоритмом *step0*.

Якщо ж вихідний x від'ємний, $x < 0$, то додаємо до нього період $T = 2\pi$ і перевіряємо потрапляння в контрольний інтервал $[-\pi, \pi]$. Так робимо, аж поки не потрапимо, а тоді – визначимо $F(x)$ алгоритмом *step0*.

Оскільки кількість операцій віднімання–додавання невідома заздалегідь, у програмі слід використати оператор ***while*** – **оператор циклу з невизначеною кількістю циклів**. Наведемо текст програми, збережений, скажімо, в файлі "*step_pi.m*":

Лістинг 3.3 (*m*-файл *step_pi.m*)

```
function  F=step_pi(x)
% Періодична Step-функція,
%тобто періодичне, з періодом 2*pi
%подовження Step-функції на весь числовий ряд
%          / =-1; якщо  x \in [-pi, 0]
%  F(x)=
%          \ = 1, якщо  x \in (0, pi]
%% Розробник Гаєв Є.О., 10.01.01
L=length(x);
for i=1:L %Початок циклу № 1 по елементах вхідного вектора
    xx=x(i);
     if  (xx >= -pi) & (xx < pi) % Перевірка xx відносно [-pi, pi]
     %  disp('x потрапив в інтервал [-pi, pi]');
        F(i)=step(x(i));
     elseif   xx <= 0 %  Якщо x не потрапив та від'ємний
        % disp(' X < 0 ' );
          nT=0;
       %Цикл № 2 доки xx потрапить в інтервал [-pi, pi]
        while ~ ((xx >= -pi) & (xx < pi))
              nT=nT+1;   xx=xx+2*pi ;
        end                                    % Кінець циклу 2
          F(i)=step(xx);
     else              %   Якщо x не потрапив та додатній
      % disp(' X > 0 ' );
            nT=0;
      % Цикл № 3 доки xx потрапить в інтервал
```

```
        while ~((xx >= -pi) &(xx < pi ))
              nT=nT+1;   xx=xx-2*pi;
        end                                    % Кінець циклу 3
        F(i)=step(xx);
     end
%  Закінчення перевірки положення xx відносно [-pi, pi]
end          % Кінець циклу № 1


%-------------------------------------------------------------
%   SubFunction, допоміжна функція.
%Це можливо для m-функцій!
%-------------------------------------------------------------
function  F=step(x)
%  Step Function дає -1  якщо x \in [-pi, 0]  і дає  1, якщо x \in (0, pi]
   if x<=0
         F=-1;
   else
         F=1;
   end
```

Пояснення до програми *step_pi*.

Після коментарів у рядках з другого по восьмий, роль яких вже пояснено, іде тіло програми. Оскільки ця програма вже доволі складна, слід починати її аналіз із з'ясування того, із скількох циклів та логічних блоків вона складається, тобто який оператор *end*, що має закінчувати блоки та цикли, до якого "відкриваючого оператора" *for*, *while* або *if* відноситься. І тут такий „***правильний стиль програмування***”, розташування команд у середині блоків „сходинками”, стає до нагоди! Розташування операторів "сходинками", що його автоматично зробив редактор *m*-файлів, дозволяє легко встановити наступне:

- програма складається з зовнішнього циклу, що розпочинається з *for* в 10-му рядку і закінчується останнім *end* в 33-му рядку; названі оператори пояснено коментарями "Початок циклу № 1 по елементах вхідного вектора" і "Кінець циклу № 1".

- всередині цього зовнішнього циклу знаходиться оператор умовного виконання *if* (12-й рядок) **…** *end* (31-й рядок) в його повній формі (3.5).
- серед операторів, які виконуються після умови *elseif*, присутній цикл №2, що починається з *while* у 19-му рядку і закінчується *end* у 22-му.
- ще один "вкладений" цикл №3 виконується після умови *else*. Це також є цикл з невизначеною кількістю обертів *while* (27-й) **…** *end* (29-й рядок).

З урахуванням зазначеної структури програми *step_pi*( ) розберемо її роботу. Цикл №1 виконується однаково для кожного з L елементів вхідного вектора. Для чергового i-го елемента $x(i)$ утворюємо службову змінну *xx* (на всякий випадок, аби не змінити x) і вже її аналізуємо. Якщо $xx \in [-\pi, \pi]$, то умова оператора виконана і у 15-му рядку програми відповідному елементу вихідного вектора *F(i)* надається значення відомою вже програмою *step*, управління передається "закриваючому" операторові *end* у 31-му рядку і починається аналіз *i+1*-го елемента вхідного вектора. Якщо ж потрапляння в контрольний інтервал не сталося, то діємо в залежності від того, від'ємне *xx* чи додатне. У першому випадку виконується цикл № 2, у другому – цикл № 3. На виході з цих циклів змінній $F(i)$ надається значення програмою *step*. Блок-схема програми на рис. 3.7 надасть додаткового розуміння цієї програми.

Про роль "закоментованих" команд розповідається у розд. 3.4.

Тепер можна побудувати графік періодичної на всій числовій осі функції (3.4). Це можна зробити або командою для аналітично заданих функцій

```
>> ezplot('step_pi(t)',[ -5*pi, 3*pi])
```

або такою командою для чисельно-заданих функцій:

```
>> t=-5*pi:8*pi/1000:3*pi;
>> F=step_pi(t);plot(t,F),axis([-5*pi,4*pi,-1.1 1.1])
```

Цей графік показано на рис. 3.6. Стає зрозумілою дана тут назва програми-функції *step* (*сходинка*).

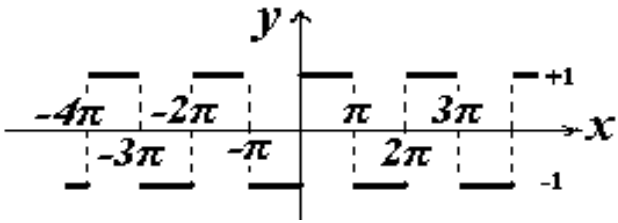


Рис. 3.6. Графік періодичної Step-функції

Після закінчення роботи програми *step_pi* в зовнішнє середовище видається вихідний вектор *F* такої самої довжини, як вхідний вектор *x*. А от змінні *L*, *xx*, *nT* (*лічильник циклів*) "забуваються" – у цьому суттєва відмінність програми-функції від програми-скрипту. Нижче буде дано приклад використання щойно створених функцій і скрипту.

Програми-функції *step1* і *step_pi* дають приклади використання двох типів операторів управління. Подальша програма обчислення чисел Фібоначі *My1Fibonachi* з розділу 4.3 надає ще один приклад використання структури *while* **. . .** *end*. Інформацію про інші, менш поширені, оператори управління разом з прикладами використання можна отримати командами допомоги *help switch*, *help try*, *help break*.

3.3.3. ПІДПРОГРАМИ. ГЛОБАЛЬНІ ЗМІННІ

Ще декілька важливих властивостей програм-функцій.

1. В програмі *step_pi* є звертання до функції *step*. Таке звертання *step*(*xx*) нічим не відрізняється від звертання до будь-якої внутрішньої функції системи MATLAB на зразок *sin*(*xx*) або *exp*(*xx*); припускається лише, що відповідна функція вже створена у середовищі MATLAB. Але не варто визначати функцію *step*(*xx*), чи будь-яку іншу, якщо вона не має самостійного значення і вживається лише у зв‘язку з даною функцією. Тому в версіях системи MATLAB починаючи з v.6, введено можливість використовувати **підпрограм** в *m*-функціях, і саме така можливість використана в Лістингу 3.3. Наприкінці програми-функції

step_pi, у тому ж файлі *step_pi.m* написано ще функцію *step*, яка і є підпрограмою і оформлена у відповідності з загальними правилами MATLAB (з єдиною відмінністю, що інформація про неї не буде видана командою *help step*). Таке ілюструє властивість *m*-функцій (*iii*). Підпрограм може бути кілька в одному *m*-файлі. Порядок їхнього розташування ніякої ролі не відіграє.

Тим не менше, запитайте MATLAB командою *help step* – і отримаєте інформацію про іншу функцію *step*, навіть про кілька функцій *step*, що належать до пакетів *LTI*, *IDmodel*, *IDdata*. Це вказує на важливу для нас властивість MATLAB: потрібну для виконання програми-функції інформацію він шукає спочатку всередині її тексту (і тому, в нашому випадку, виконує саме нашу функцію *step*!) і лише потім, якщо не знайде, шукає ще десь. Ось і знайшов якусь іншу функцію з тим самим іменем десь в інших пакетах... Це ім'я, як кажуть, *перегружене*, десь ще використовується, але конфлікту поки що немає.

Ще зверніть увагу: підпрограму *step* написано у найпростішому варіанті, розрахованому лише на поодинокі числа, а не вектори. Але ж складніше і не треба, бо звертання до неї *F*(*i*)=*step*(*xx*) іде саме для поодинокого числа *xx*!

2. Інколи виникає потреба порушити **конвенцію щодо локальної ролі змінних**, правило (*ii*), породжених усередині програми-функції. Наприклад, вам потрібно знати (або навіть використати в іншій програмі) скільки зсувів *nT* основного інтервалу було зроблено. Тоді на початку *кожної* з програм треба перерахувати всі спільні, **глобальні** змінні, тобто поставити рядок з описом змінних на зразок

```
global  nT  X  Y  Z
```

(перераховуються без ком, „пропуск" розділяє змінні).

Таку ж команду треба виконати в командному рядку, якщо передбачається використовувати там змінні *nT*, *X*, *Y* та *Z* як глобальні (спільні з якоюсь програмою). Приклад використання подає програма *Ball* з розділу 4.5, де це означення використано задля зв'язку з підпрограмами *Right2* та *LinearMotion*.

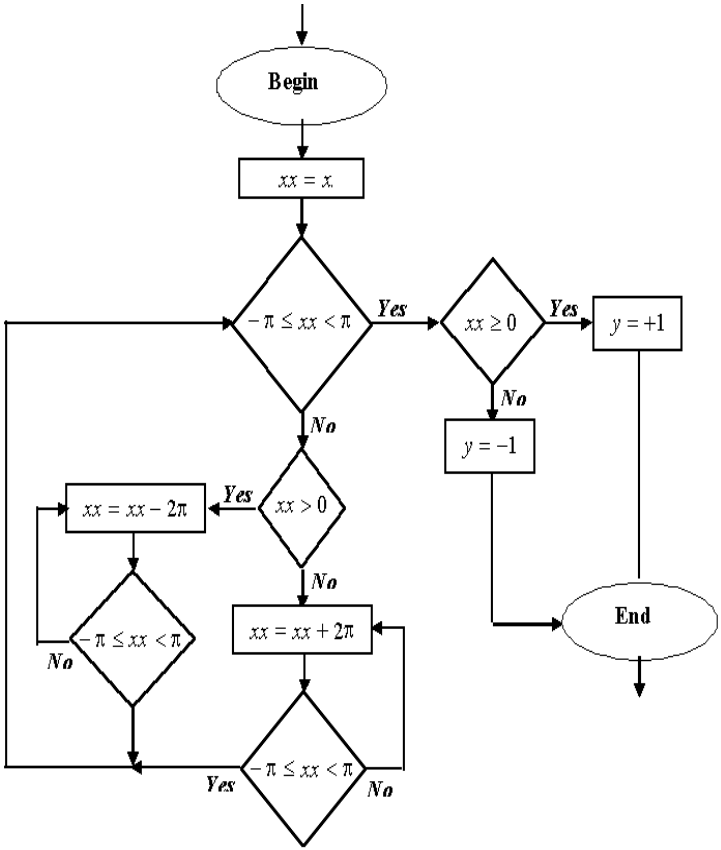


Рис. 3.7. Блок-схема програми *step_pi*.

Глобальні змінні використовують також при створенні графічного інтерфейсу користувача (GUI, Graphical User Interface), див. у другій частині Посібника. Подробиці можна знайти в довідковій системі *Help* за ключовим словом *global*, або прямо з командного вікна MATLAB:

help global та *help isglobal*.

3. Маючи справу з програмами і з підпрограмами, з їх написанням та викликом, треба розрізняти їхні вхідні та вихідні аргументи як **формальні параметри** і **фактичні**

параметри. Коли пишемо *m*-функцію, не має значення, як назвати її вхідні і вихідні параметри. Наприклад, створили *m*-функцію

$$[X, Y, Z]=MyProgram(x, y, z, a, b),$$

тобто у тілі цієї програми якимось чином пов'язали вихідні аргументи X, Y, Z з її вхідними аргументами x, y, z, a і b. Сутність і робота цієї програми не зміниться, якщо імена усіх цих аргументів (параметрів), як у її назві, так і у тілі програми, замінити на якісь інші, наприклад

$$[U, V, T]=MyProgram(X1, Y1, Z1, A, D).$$

Мабуть тому ці параметри (аргументи) називаються **формальними**.

Програма або підпрограма написана і збережена у файлі „*MyProgram.m*". Тепер виконуємо цю програму у якомусь розрахунку з практичною метою, для фізики, фінансової справи тощо. За кожним параметром стоїть певний зміст, наприклад, мають значення *Masa*, *Temperatura*, *Tisk*, *Vartist1*, *Vartist2*, від яких залежать *Shvidkist*, *Energy* і *Chas*. Тоді до програми треба звернутись з цими **фактичними параметрами** наступним чином:

$$[Shvidkist, Energy, Chas]=MyProgram(Masa, Temperatura, Tisk, Vartist1, Vartist2).$$

Що при цьому має місце: замість формального параметра *X1* всюди у тілі програми підставляється фактичний параметр *Masa*, замість *Y1* – *Temperatura*, і так далі, замість *D* – підставляється *Vartist2*. Тепер з фактичними змінними виконуються ті самі операції, що запрограмовані для відповідних формальних, у результаті отримують вихідні величини, і вони видаються для змінних *Shvidkist*, *Energy*, *Chas* замість формальних вихідних змінних *U*, *V*, *T*.

Не важливо, таки чином, як назвати формальні аргументи і які фактичні аргументи будуть підставлені замість них. А от що важливо – так це порядок розташування аргументів, їх знаходження у переліку вхідних або вихідних! Якщо змінну *Masa* поставити випадково на друге місце, то з

нею будуть формально виконуватись ті перетворення, що призначалися для *Temperatura*. Якщо *Chas* поставити на перше місце у списку вихідних аргументів, то ця змінна отримає значення, призначене для *Shvidkist*. Таке, може, не викличе якогось збою алгоритму, але дасть фізично-помилковий результат. **Тому дуже важливо слідкувати за місцем того чи іншого аргументу!**

На перших етапах розвитку алгоритмічних мов було також важливим слідкувати за кількістю аргументів, що підставляється у програму. Скажімо, у програму *MyProgram* слід було підставляти 5, і лише 5, вхідних аргументів – інакше правильність результату не гарантована та MATLAB може видати червоне повідомлення про некоректний виклик програми. Однак згодом, прагнучи звільнитись від нудних обмежень, розробники алгоритмічних мов високого рівня почали дозволяти більше свободи у використанні програм. Наприклад, до програми *ezplot* у MATLAB можна звернутися з одним або з двома параметрами,

ezplot(‘*sin*(*x*)/*x*’) або *ezplot*(‘*sin*(*x*)/*x*’, [**-***pi*, *pi*]).

А команда *plot* – ще більш варіабельна. Такі „інтелектуальні” програми зручніше користувачеві. А от як це забезпечити – таємниця програміста. Деякі з таких таємниць ми розкриємо у другій частині цього Посібника.

Порада: слідкуйте за тим, аби імена, які ви даєте змінним, „натякали” на їхній зміст (наприклад *Energy_0*). Інший шлях – залишати у програмі коментарі, що пояснюють їхнє призначення, наприклад:

% *U -- Shvidkist*,

% *V – Energy*,

% *D -- Vartist2*.

Крім того, латинські літери *X*, *Y*, *Z*, *x*, *y*, *z* використовуйте для величин, що можуть змінювати значення, а *A*, *B*, *C*, *D*, *a*, *b*, *c*, *d* – для сталих, незмінних параметрів процесів, *i*, *I*, *j*, *J*, *k*, *K*, *l*, *L*, *m*, *M*, *n*, *N* – для величин що приймають лише цілі значення (та *i*, *j* не використовуйте, якщо одночасно оперуєте з комплексними числами).

4. Ще однією важливою рисою **структурного програмування** є використання значної кількості підпрограм. Якщо ви аналізуєте чужу програму, то побачите, що вона викликає інші, ті у свою чергу – ще інші, ті – знов викликають, і так – до певної визначеної, конечної глибини. (Є лише один різновид програм, *рекурентні програми*, коли вона звертається сама до себе невизначену кількість разів – про це ми поговоримо пізніше у розділі 4.3). Якщо ж ви – розробник доволі складної програми у якомусь колективі, то вашу частину роботи ви маєте зробити у вигляді підпрограм, що задовольняють певним умовам (стикуються з підпрограмами колег-програмістів) і можуть бути викликані головною програмою. Подальші наші приклади мають показати усі означені властивості програмування.

3.4. НАЛАГОДЖЕННЯ ПРОГРАМ. ДЕЯКІ ПОРАДИ

Інколи студенти-програмісти ледь не плачуть: „Я все зробила, як Ви казали, а програма *не хоче* працювати!”

Навіть у досвідчених програмістів написана вперше програма не завжди починає правильно працювати відразу. Ще треба її налагодити (англійською – *to debug*). Редактор *m*-файлів MATLAB v.6 пропонує тут певну допомогу. Наведемо деякі поради щодо перевірки і налагодження ваших програм.

1. Якщо після запуску вашого скрипту або використання програми-функції ви отримуєте замість очікуваного результату якийсь коментар, що розпочинається словом "*Error*:", то це свідчить про ***синтаксичні помилки*** (такі тобто, які просто порушують правила написання) у вашій програмі. Цей коментар видається, як правило, червоними літерами і включає певну підказку, де шукати помилку. Наприклад:

??? Error: *File*: *D*:*Matemat**work**Berm4271.m Line: 7 Column: 11 *")" expected*, *";" found.*

Виданий текст означає, що інтерпретатор MATLAB (який перекладає вашу програму на машинну мову) чекав від

вашого тексту закриття дужечки, *")" expected*), але дійшов до кінця командного тексту, *";" found*, так і не знайшовши її. Інтерпретатор радить шукати помилку в 7-му рядку в 11-й позиції вказаного файлу (*Berm4271.m Line: 7 Column: 11*).

Інший можливий коментар

??? Undefined function or variable 'ya'.

означає, що інтерпретатор не може надати значення змінній *'ya'*: або ви забули написати оператор на зразок *ya*= , або програмі слід взяти її значення зовні, для чого змінну *ya* слід зробити глобальною. Можуть бути і більш складні причини; як їх знайти – див. поради нижче.

2. Коли всі помилки виправлено і програма нібито починає працювати, це ще не гарантує, що вона працює вірно. В більшості випадків слід пересвідчитись, що всі змінні обчислюються правильно і приймають вірні значення.

Складайте вашу велику програму з певних програмних модулів і підпрограм; кожний з модулів слід перевірити окремо, поки програма не виросла „як сніговий ком". Перевіряйте модулі на штучних комбінаціях вхідних даних, обраних таким чином, що певні проміжні величини легко обчислюються вручну. Тоді в програмі безпосередньо за цими "контрольованими" величинами слід тимчасово ставити оператори виводу даних (оператори друку). Такі оператори використано, наприклад, в програмі *step0* (Лістинг 3.2А):

disp(*'Текстовий_Рядок'*)

Вказана команда *disp* виводить на екран комп'ютера той *Текстовий_Рядок*, який в лапках стоїть аргументом команди. Така видача дозволяє зорієнтуватися, чи програма пройшла відповідне місце у ній, чи ні. Можливі і такі використання програми друку: *disp*(*A*), або *disp A*; буде надруковано числове значення величини *A*. Проте, той самий результат одержимо, якщо просто скажемо *A* в тексті програми, або після присвоєння значення не поставимо крапку з комою: *A=a***,** *B=в***;**. Такі "контрольні видачі" в програмі *step0* (лістинг 3.2А) дозволили нам упевнитись, що оператор логіки ***if … else … end*** дійсно працює так, як передбачалося. Після

перевірки програмного модуля контрольні видачі можна викреслити з тексту програми або "закоментувати" знаком %. Це можна зробити або вручну, або за допомогою меню редактора програм ***Text\comment***, або, навпаки, за допомогою ***Text\uncomment*** викреслити цей знак. Редактор програм також дозволяє дізнатися значення, яке набула та чи інша змінна після виконання програми. Для цього треба навести на цю змінну курсор, виділити її і в меню редактора обрати ***Text\Evaluate_Selection***. Команда *disp* може бути також використана для повідомлення користувачу про той чи інший випадок під час розрахунків. Так використано команду

disp('*Матриця не є квадратною! Помилка!*')

в програмі *MyDeterminant* розділу 3.6.

Аналогічна команда вводу може допомогти в організації простого. "діалогу" з програмою, про що буде у другій частині Підручника. Наприклад, виконання команди

X=***input***('*Please input value of X*')

призведе до того, що на екрані з'явиться запрошення ввести значення для *X*; далі програма чекає на таке введення, після цього надає *X* відповідного значення і продовжує роботу. Приклад з простим діалогом між програмою і користувачем, який здійснюється з командного рядочка, дамо далі. Більш складний діалог через спеціальні графічні вікна можна організувати на основі *Graphical User Interface* (*GUI*). Подробиці також дамо у другій частині Посібника; корисна інформація є [5,42,43].

3. Не налагоджена нова програма може інколи "зациклитися", тобто нескінченно довго "обертатись" по певному колу команд без видач будь-якого результату. Чекаєш результату, чекаєш – а його немає!...

Так буває при обчисленнях за ітераційною схемою, коли програміст забуває передбачити випадок розбіжності ітерацій. Аби перервати роботу програми в „аварійному" випадку і зробити комп'ютер знов керованим, використовують комбінацію клавіш

< *Ctrl – C* >.

4. Читач, напевне, помітив, що лише частину коментарів до програм ми подали українською мовою. Ми змушені так робити, бо MATLAB[21] не завжди "розуміє" українську або російську. У MATLAB версії до 6, якщо ви спробуєте вводити у командному рядку навіть коментар

>> % Проблемна Ситуація –

все буде гаразд, аж поки ви не надрукуєте літеру ***c***-маленьке; з великим ***C*** – все гаразд. Використання коментарів з символами кирилиці в скриптах і програмах-функціях призводить до повідомлення про помилки, хоча їх насправді немає. Тому користуйтеся латинськими літерами, доки програмісти остаточно не пристосують MATLAB до потреб нашого ринку.

В меню графічного вікна *Figure*, однак, труднощів з кирилицею майже немає. Пункт меню ***Insert\Title*** дозволить написати такий, наприклад, заголовок над графіками: "*Обєм еліпсоїда*". (Проблема, як бачимо, лише з апострофом).

Взагалі, якщо у заголовках графіків, у надписах осей тощо використовувати грецькі літери („Кут β”), „двоповерхові” формули („$\frac{z}{x}$”, „$e^{\sqrt{x}}$”), більш складні математичні вирази – користуйтеся елементами L_ATeX, пишіть на зразок *title*('*\alpha \beta \gamma \delta*'), *legend*('*e^x*'), *text*(-1,1,'{*\color*{*red*} слово}') тощо, див. *help tex* та [6].

5. Зверніть також увагу на те, що редактор *m*-файлів – дуже „розумна” програма, і багато чого робить, аби Вам допомогти! Як само вона допомагає – залежить від версії програми MATLAB. Вже починаючи з версії **v.6.*** редактор розміщує рядочки Вашої програми у стилі драбинки, аби вона легше читалася. А от у MATLAB **v.7*** редактор *m*-файлів допоможе Вам краще побачити структуру Вашої

[21] Це стосується версії v.6.5. У наступних версіях MATLAB v.7.* і v.8.*5 цей недолік, здається, виправлено.

програми – він може „згортати” або „розгортати” структурні блоки програми, як то пояснює рис. 3.8.

Крім того, редактор *m*-файлів намагається звернути Вашу увагу на проблемні місця у тексті програми. Рис. 3.9 надає подробиці. Якщо редактор „вважає”, що у програмі є помилки[22], то квадратик, що показано стрілкою 1, стає червоним. Якщо редактор помилок не знаходить, та деякі частини коду „вважає” недоліками, то квадратик стає помаранчевим, і недоліки позначаються рискою такого ж кольору праворуч. Наведіть на риску мишу – з’явиться пояснення щодо знайденого недоліку[23] (знову привід вивчати англійську!).

6. Якщо ви працюєте не на власному комп’ютері, а **в комп’ютерному класі** загального користування, у вас можуть виникнути проблеми зі збереженням ваших програм, даних і результатів розрахунків. Справа у тому, що Системний Адміністратор надає вам лише обмежені ***права доступу*** до ресурсів комп’ютера. Часто ви будете мати право зберігати ваші файли лише в директорії *TEMP*, на дискеті *A* або на флешці – куди саме, спитайте Адміністратора. Оскільки інші студенти також зберігають там свої файли, радимо вам суворо дотримуватися певного порядку. Доцільно створити в *TEMP* піддиректорію для всієї групи студентів, скажімо *IAN-203*, всередині якої кожний студент створює піддиректорії вже для власних файлів. Ці піддиректорії краще назвати прізвищами студентів – *Kozenko*, *Mironchenko* тощо. Не знищуйте файлів в чужих директоріях! Проте, на всяк випадок, робіть копії важливих файлів на власну дискету.

7. Може статися, однак, що MATLAB "не захоче" бачити створені вами програми в директорії, скажімо, *TEMP/Kozenko*, або на дискеті. Це означає, що дані директорії треба "прописати" в спеціальному переліку директорій *PATH* (Шлях). Спробуйте самі (у разі потреби –

[22] Він „бачить” лише ***синтаксичні помилки***... Тобто, не всі!

[23] Автори деякі „недоліки”, що їх показує редактор, не вважають за такі.

просіть Системного Адміністратора) зробити наступне. В меню MATLAB оберіть ***File/Set_Path…***. У вікні *Set_Path*, що з'явиться, натисніть клавішу ***Add Folder…***. З'явиться "драбина" з переліком усіх директорій і дисків комп'ютера. Знайдіть директорію з вашими файлами та натисніть *OK* – вашу директорію буде вказано в загальному переліку вікна *Set_Path*. Якщо покинути це вікно за допомогою *Close*, ваше налагодження буде дійсним лише протягом даного сеансу роботи – і часто цього достатньо. А натискання *Save* потрібно в разі довготривалої роботи з MATLAB.

8. Якщо потрібно зберегти ваші результати розрахунків (змінні у вигляді масивів чи структур) для наступного сеансу роботи – в меню оберіть

File/Save Workspace As…

У вікні, яке з'явиться, оберіть директорію, де зберігаєте ваші дані і надайте файлу даних певне ім'я (доцільно використати або ваше прізвище, або дату роботи, скажімо, *Kozenko_Dezember7*). Буде утворено двійковий файл *Kozenko_Dezember7.mat*. Наступного разу для "пригадування" збережених даних просто оберіть з меню ***File/Open…***, або натисніть "іконку"  і оберіть потрібний файл. Як прочитати у MATLAB дані з інших програм, див. у другій частині Посібника.

Іноді доцільно зберегти не лише дані, а і певний конспект ваших дій під час роботи з MATLAB, аби пізніше можна було б пригадати, як саме ви впорались з тією чи іншою задачею. MATLAB буде "нотувати" всі ваші дії в текстовому файлі *diary*, якщо з командного рядка виконати команду *diary*. "Щоденник" вашої роботи під час MATLAB-сеансу також доцільно назвати датою вашої роботи; тоді команду слід виконати у вигляді

>> diary(' *Kozenko_Dezember7'*)

– назва файлу в лапках.

3.5. ПРИКЛАД 1: АНАЛІЗ ЗБІЖНОСТІ РЯДУ ФУР'Є

Використаємо створені в даному розділі програми для аналізу (вже більш досконалого з точки зору програмування)

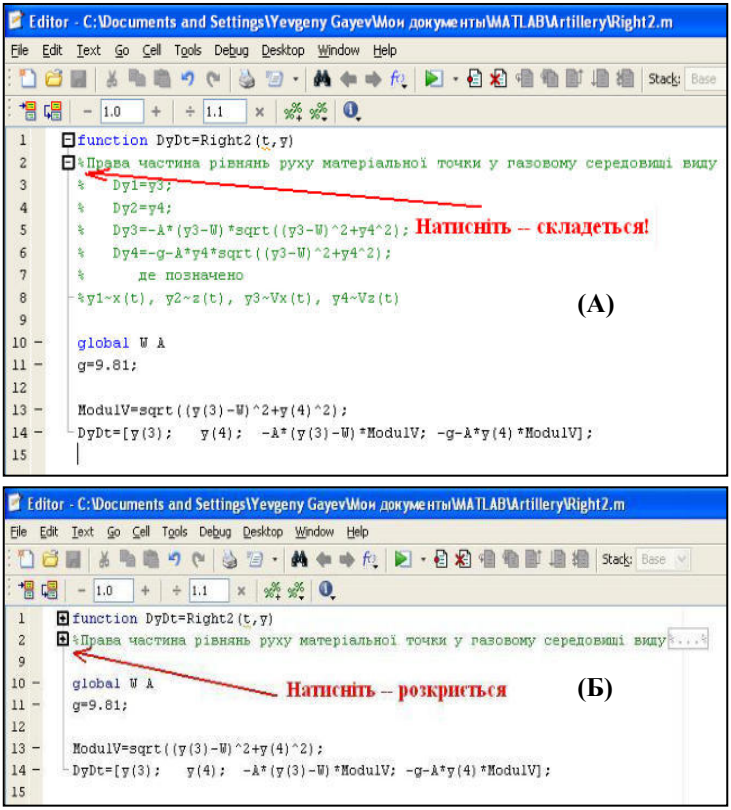


Рис. 3.8. Вигляд лівої сторони вікна редактора *m*-файлів з допоміжними засобами налагодження програм, що розробляються: з засобами „згортання” (А) і „розгортання” тексту програми (Б).

математичної задачі лабораторної роботи 2, запропонованої раніше.

Нагадаємо, що створено скрипт *visualize* і підпрограми-функції *G1*, *G2*, **…** *G10*, а також програму побудови періодичної функції (3.4), і все це дозволяє дослідити, чи дійсно послідовність частинних сум ряд Фур‘є (3.3) наближає розривну функцію (3.4), і як швидко збігаються до неї частинні суми ряду (3.3).

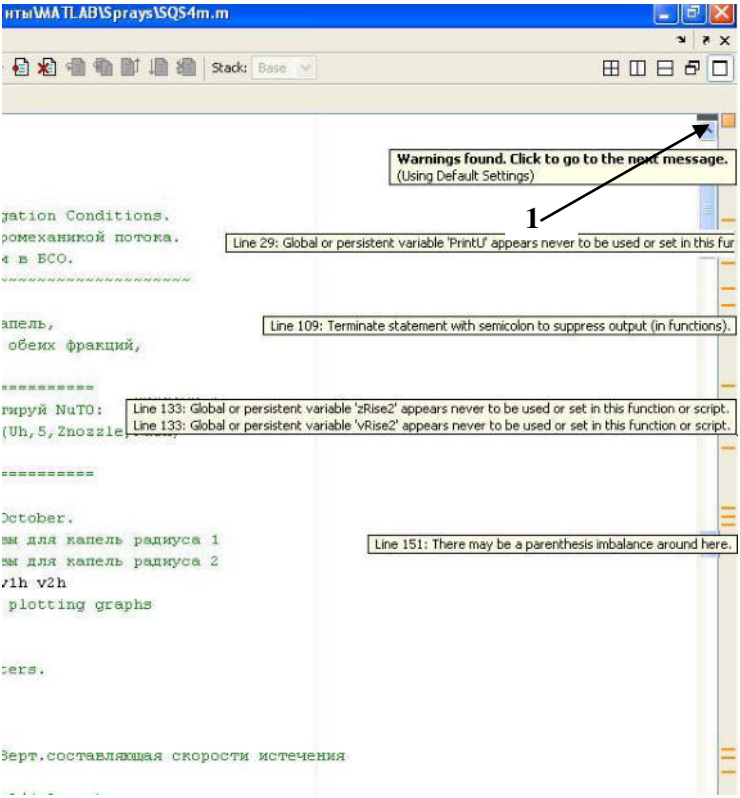


Рис. 3.9. Вигляд правої сторони вікна редактора *m*-файлів з „підказками” про помилки, що виникають після наведення миші на відповідну помаранчеву риску.

Виконуємо в MATLAB послідовність команд аналогічно переліку (3.1), призначення яких пояснюється коментарями:

```
>> Xbegin=-3*pi ;  Xend=2*pi ;  N=1000 ;
>> Step= (Xend - Xbegin)/N ;
>> X= Xbegin : Step : Xend ;                % Аргумент
>> %  Вектори значень 3-х функцій
>> F0=step_pi(X); F1=f3(X); F2=f5(X);
>> %Побачити графіки
```

```
>> visualize
>>%  Вектори значень ще 2-х функцій
>> F1=f8(X); F2=f10(X);
>> %Побачити графіки знов
>> visualize
>> %Побачити графіки знов, і так далі...
```

Результатом роботи будуть два графічних вікна з графіками, які показані на рис. 3.10. Стрілки і пояснюючі надписи на рисунках зроблено вже в самих графічних вікнах, так само збільшено товщину ліній. Лінія у вигляді сходинки, що задається рівнянням (3.4), помічена цифрою 0. Можна зробити такі математичні висновки.

Частинні суми ряду (3.3), хоча і є неперервними функціями, дійсно поступово наближають функцію (3.4), розривну в точках $x = (2k+1)\pi$, де k пробігає значення $k = \pm 1, \pm 2, \pm 3, \ldots$. Старші частинні суми *G8*(*x*) і *G10*(*x*) краще наближають функцію (3.4), аніж частинні суми *G3*(*x*) і *G5*(*x*). Можна очікувати, що з більш старшими частинними сумами наближення ще покращиться, та строго таке можна показати лише математичним доведенням, та уточнити – у якому сенсі "покращиться". Проте, геометричне дослідження на ґатунок наведеного для, скажімо, *f20*(*x*) і *f30*(*x*) і т.д. може бути цікавим!

Зауваження: Критично налаштований читач може помітити, що навіть для старших частинних сум все одно лишаються помітні "сплески" функцій, що наближують, в точках розриву. Так, це явище дійсно має місце і було вперше помічено відомим фізиком Гібсом [41]. Воно так і було названо – ефект Гібса[24].

А ви звернули на це увагу тоді, коли вивчали ряди Фур'є у курсі вищої математики? Радимо детальніше дослідити ряди Фур'є, див. розділ 4.2.

[24] Фяди Фур'є та, точніше, частинні суми Фур'є, "працюють" у ваших мобільних телефонах, а ефект Гібса є причиною перешкод та шуму у них.

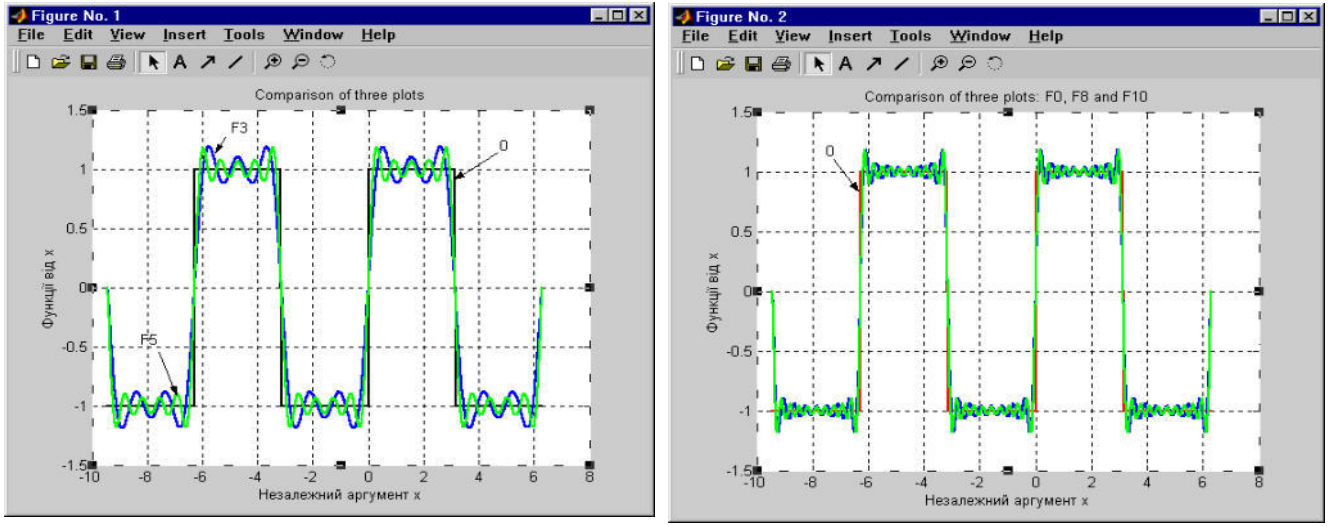


Рис. 3.10. Співставлення третьої і п‘ятої (зверху), восьмої і десятої (унизу) частинних сум ряду (3.3) з граничною функцією (3.4), позначеною номером *0*

3.6. ПРИКЛАД 2: ОБЧИСЛЕННЯ ВИЗНАЧНИКІВ

Дамо ще один корисний приклад складної програми MATLAB. Обчислення визначника – поширена математична операція. Уявіть собі, що ви отримали завдання розробити програму для його обчислення. Ваша робота має складатись з таких етапів.

Етап 1. Для кожної задачі, фізичної чи математичної або економічної тощо, спочатку слід **поставити задачу математично**, пригадати зміст та властивості понять, які вона використовує. Визначник матриці – це число, яке обчислюється за заданою таблицею чисел A:

$$\det(A) = \begin{vmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{vmatrix}. \qquad (3.7)$$

Відомо, що визначник не змінюється, якщо від елементів певного стовпчика відняти відповідні елементи іншого стовпчика, попередньо помножені на довільне число (так само для рядків). Якщо названими діями визначник (квадратну таблицю чисел, масив) перетворити таким чином, аби всі члени певного рядка, за винятком одного, скажімо a_{pq}, дорівнювали 0 (нулю), то визначник дорівнюватиме $(-1)^{p+q} a_{pq}$, помноженому на визначник меншого на одиницю порядку [34,36]. Далі так само перетворюємо цей останній визначник, аж поки отримаємо "визначник першого порядку", тобто число. Пояснимо таку техніку на прикладі „ручних" обчислень [34, с.266]:

$$\Delta_5 = \begin{vmatrix} 3 & 1 & \underline{-1} & 2 & 1 \\ -2 & 3 & \underline{1} & 4 & 3 \\ 1 & 4 & \underline{2} & 3 & 1 \\ 5 & -2 & \underline{-3} & 5 & -1 \\ -1 & 1 & \underline{2} & 3 & 2 \end{vmatrix} = \left\{ \begin{matrix} \text{Третій стовпчик,} \\ \text{помножений на 3,} \\ \text{додаємо до першого;} \\ \text{помножений на 1} \\ \text{до другого; т.д.} \end{matrix} \right\} = \begin{vmatrix} 0 & 0 & \underline{\underline{-1}} & 0 & 0 \\ 1 & 4 & \underline{1} & 6 & 4 \\ 7 & 6 & \underline{2} & 7 & 3 \\ -4 & -5 & \underline{-3} & -1 & -4 \\ 5 & 3 & \underline{2} & 7 & 4 \end{vmatrix}$$

$$= - (-1)^{1+3} \begin{vmatrix} \underline{1} & 4 & 6 & 4 \\ \underline{7} & 6 & 7 & 3 \\ \underline{\underline{-4}} & -5 & -1 & -4 \\ \underline{5} & 3 & 7 & 4 \end{vmatrix} = \left\{ \begin{matrix} \text{З усіх стовпчиків} \\ \text{віднімаємо перший,} \\ \text{помножений на} \\ \text{множники 4, 6 і 4} \end{matrix} \right\} = - \begin{vmatrix} \underline{\underline{1}} & 0 & 0 & 0 \\ 7 & -22 & -35 & -25 \\ -4 & 11 & 23 & 12 \\ 5 & -17 & -23 & -16 \end{vmatrix}$$

$$= -(-1)^{1+1} \begin{vmatrix} \underline{-22} & -35 & -25 \\ \underline{11} & 23 & 12 \\ \underline{-17} & -23 & -16 \end{vmatrix} = \left\{ \begin{matrix} \text{Вихідним приймаємо} \\ \text{перший стовпчик,} \\ \text{помножений на} \\ \text{множники } \frac{35}{22} \text{ і } \frac{25}{22} \end{matrix} \right\} =$$

$$= - \begin{vmatrix} \underline{\underline{-22}} & 0 & 0 \\ 11 & 5{,}5 & -0{,}5 \\ -17 & 4{,}1 & 3{,}3 \end{vmatrix} = 22 \cdot (-1)^{1+1} \begin{vmatrix} 5{,}5 & \underline{-0{,}5} \\ 4{,}1 & \underline{3{,}3} \end{vmatrix} = 22 \begin{vmatrix} 0 & \underline{-0{,}5} \\ 40{,}4 & \underline{\underline{3{,}3}} \end{vmatrix} =$$

$$= 22 \cdot (-0{,}5) \cdot (-1)^{1+2} \cdot \left| (40{,}4) \right| = \left\{ \begin{matrix} \text{Останній} \\ \text{множник-} \\ \text{визначник від} \\ \text{матриці } 1 \times 1 \end{matrix} \right\} = 22 \cdot 0{,}5 \cdot 40{,}4 = 444{,}4$$

(Відносна похибка цього результату 0,4% зумовлена округленням деяких обчислень). Сказане дозволяє "вручну" діяти за наступним алгоритмом.

<u>Етап 2.</u> "Ручне" обчислення заради кращого оволодіння алгоритмом. Єдина відмінність: при ручному обчислені за "основний" (підкреслений) обирався стовпчик, за можливістю, з одиницею на горі; тепер "основним" завжди буде перший стовпчик кожного чергового визначника. Наводимо алгоритм.

0. Робимо копію *Am* заданої матриці *A* оператором *Am= A*, аби не зруйнувати її в результаті перетворень.

1.1. Виконуємо команду

>>p=1; *q=p+1*; *Am*(*p:end,q*)=*Am*(*p*:*end,q*)-**...**

$$Am(p{:}end, p) * Am(p, q) / Am(p, p) \qquad (3.8)$$

Отримуємо увесь другий стовпчик *q=2,* перетворений за потрібним правилом.

1.2. Повторюємо команду (3.8), поклавши, однак, $q = q + 1$. Перетворено 3-й стовпчик матриці *Am.*

1.3. Так робимо доти, аж поки q не стане дорівнювати n, вимірності матриці A. Отримуємо в результаті нову матрицю *Am*, в якій в першому рядку всі нулі, за винятком $a_{11} \neq 0$. Кількість команд, яких було виконано, дорівнює *n-1*.

2. Повторюємо дії 1.1 – 1.3 за формулою (3.8), починаючи з *p=2*. В матриці *Am* маємо вже два "нульових" рядки. Команду (3.8) було повторено *n-2* разів.

3. Так само "отримуємо нулі" у третьому, четвертому і т.д. рядках. В результаті – матриця *Am* має "трикутний" вигляд з нулями вище головної діагоналі.

4. Визначник дорівнює добутку чисел, які стоять на головній діагоналі. (Порівняйте результат з обчисленням за "внутрішньою" програмою MATLAB *D=det*(*A*)!)

Етап 3. Програмування в MATLAB

Створюємо *m*-файл із ім'ям *MyDeterminant.m* зі змістом:

```
function d=MyDeterminant(X)
% Навчальна програма обчислення визначників
% квадратних матриць вимірності n
%-----------------------------------------------
% Приклад роботи: >> d=MyDeterminant(A)
[n,m]=size(X); % size – внутрішня програма MATLAB
if n~=m % Перевірка, чи є матриця квадратною
    disp('Матриця не є квадратною! Перевірте Вашу задачу!');
else % Якщо матриця дійсно квадратна
   Am=X;
      for p=1:n
          for q=p+1:n
              Am(p:end,q)=Am(p:end,q) - ...
                  Am(p:end,p)*Am(p,q)/Am(p,p);
  %pause % Закоментована команда для тестування програми
          end
      end %Отримано матрицю з нулями вище головної діагоналі
v=diag(Am); det=prod(v)
end
```

Якщо з командного вікна наказати, як рекомендовано у Help–овій частині програми

>> d=MyDeterminant(*A*),

де *A* – задана матриця, то програма створює вектор з діагональними елементами перетвореної матриці і видає їх добуток – значення визначника *d*. Якщо в цій програмі "розкоментувати" команду *pause* і прибрати попередню кому з крапкою, то програма буде виводити чергову перетворену матрицю і зупинятись після обчислень кожного із стовпчиків крк за кроком. Натискання на *Ввод* продовжує обчислення. "Розкоментована" команда *pause* – ще одна корисна команда для налагодження програм; у даному випадку вона робить наочним поступове перетворення матриць.

Таким чином, в даному розділі ви навчилися

- ✓програмувати прості скрипти і програми-функції,
- ✓використовувати оператори умовного виконання *if – else – end* і
- ✓оператори управління обчислювальним процесом *for – end*.

На закріплення матеріалу з навичок програмування пропонуємо лабораторну роботу 5. Після цього можна перевірити себе ще раз на контрольних питаннях до даного розділу 3.8, пропустивши, у випадку першого читання, наступний розділ. Але пізніше ознайомитись всеж-таки з блоком команд вибору з кількох випадків ***switch . . . end.***

3.7. ДВІ КЛАСИЧНІ ЗАДАЧІ АЛГОРИТМІЗАЦІЇ

Наука програмування налічує певну кількість стандартних задач, які зустрічаються у підручниках з кожної алгоритмічної мови, яку б не вивчати. З тих задач двома основними є

- ✓ **1. Задача пошуку найменшого елементу вектора**, і
- ✓ **2. Задача пересортування елементів вектора**

Познайомимось із ними. Також познайомимось у цьому розділі з новим блоком команд ***switch . . . end***

3.7.1. НАЙМЕНШИЙ ЕЛЕМЕНТ ВЕКТОРА

Нехай на вхід програми подається якийсь вектор, чи то вектор-рядочок, чи то стовпчик – неважливо. Програма має знайти найменший з елементів вхідного вектора і видати цей елемент разом з номером його позиції. Приклад: на вхід подається вектор *X* =[*2*,*-3*, *4*, *5.1*]; програма має видати *-3* і *2*, бо останнє ціле число є положення найменшого елемента *-3* у векторі *X*.

Назвемо майбутню програму *MyMin* (бо в MATLAB є власна команда *Min*). Алгоритм дій з пошуку найменшого елемента дуже простий і майже очевидний. Спочатку „підозрюємо" самий перший елемент вектора *X*(*1*), що він і є найменший X_{min}. Далі перевіряємо це припущення**:** беремо наступний, другий елемент вектора *X*(*2*) і порівнюємо з X_{min}; якщо він дійсно більший – нічого не змінюємо, якщо ж менший – оголошуємо X_{min}= *X*(*2*). Далі так само діємо з третім, четвертим, і до останнього елементу вектора *X*. Одночасно змінюємо, якщо нерівність виявляється порушеною, номер шуканого елемента I_{min}.

Програма, що наводиться, не вимагає довгих пояснень.

Скрипт програми MyMin.m

```
function [Xmin, Imin]=MyMin(X)
%Програма знаходження найменшого елементу Xmin
%вектору X разом з його номером Imin у векторі.
% Автор Є.Гаєв, % Дата . . . .
%Приклад використання: на команду
%>> MyMin([2,-3, 4, 5.1])
% отримаємо -3 і 2.

L=length(X);
Xmin=X(1);   Imin=1;

for i=2:L
      if  X(i) < Xmin
            Xmin=X(i); Imin=i;
      end
end
%End of the Program
```

Завдання для самостійної роботи: розробити аналогічну функцію *MyMax*, яка повертає найбільший елемент вхідного вектора *X* разом з його номером серед елементів *X*.

У наступному підрозділі вважаємо, що обидві команди *MyMin* і *MyMax* Ви створили і ми можемо їх використовувати.

3.7.2. ПЕРЕСОРТУВАННЯ ЕЛЕМЕНТІВ ВЕКТОРА

Нехай на вхід програми подається числовий вектор з певною кількістю елементів, наприклад *X* =[*2*,*-3*, *4*, *-5.1*]. Програма має повернути вектор *Y* з тих самих елементів, розташованих, однак, у порядку збільшення, тобто *Y* =[*-5.1*,*-3*, 2, *4*], чи то у порядку зменшення, тобто *Y* =[*4*, 2, *-3*,*-5.1*]. Нехай обидві операції виконує та сама програма, будемо лише вказувати їй '*Increase*' чи то '*Decrease*'.

Ось приклад, де така задача – зовсім несподівано! – виникає. Припустимо, програміст створив програму пошуку авіаційних сполучень з будь якого замовленого міста планети до будь-якого іншого. Складний алгоритм, складна програма! Та от наостанці треба зробити видачу користувачеві результатів у порядку зростання або зменшення ціни квитка. Ось вам і наша задача! Звернемось до неї.

Спочатку треба запропонувати якийсь алгоритм. Пропонується такий для, припустимо, вимоги '*Increase*'. Шукаємо найменший елемент даного вектору X_{min} за допомогою попередньої програми *MyMin*, засилаємо його першим елементом майбутнього результату *Y*, $Y(1)= X_{min}$. Убираємо знайдений елемент з вектора *X*; для цього створюємо тимчасовий вектор $X1=[X(1{:}\ I_{min}\text{-}1), X(I_{min}+1, end)]$. В останній включено елементи перед найменшим $X(1{:}\ I_{min}\text{-}1)$, і всі елементи поза найменшим $X(I_{min}+1, end)$; це зробити можливо[25], бо програма нам повідомила про положення найменшого I_{min} у вхідному векторі. Сам найменший не

[25] Сподіваємось, ви не забули, що MATLAB є Matrix Laboratoty!

включений, ніби то „виштовханий” з вектору. Тепер шукаємо найменший елемент допоміжного вектору *X1* тією ж командою *MyMin*. Записуємо його черговим елементом результату $Y(2)= X_{min}$, і знов коригуємо допоміжний вектор $X1=[X1(1 : I_{min}-1), X1(I_{min}+1, end)]$. В останньому – усі попередні елементи за виключенням щойно знайденого. Робимо такі ж дії третій раз, „виштовхуючи” черговий найменший елемент, і так далі, аж поки не переберемо усі елементи *X*, і в *X1* не залишиться жодного. В результаті, вектор *Y* поступово виросте до повної кількості елементів, і розташовані вини будуть у порядку збільшення.

Описана процедура „виштовхування” нагадує рух бульбашки у рідини, тому алгоритм так і зветься – „***метод бульбашки***”. Існують і інші методи, про які можна дізнатися в [44-46] або в Інтернеті.

Алгоритм дій лишається таким самим, коли треба розташувати елементи у порядку зменшення (замовлення ‘*Decrease*’), лише треба використовувати програму *MyMax*.

Назвемо програму *MyOrder*. Описане вище – її складові. Вони, разом з діями у залежності від замовлення ‘*Decrease*’ чи то ‘*Increase*’, додатково пояснюються блок-схемою.

Наведемо Лістінг файла *MyOrder.m*

```
function Y=MyOrder(X, Order)
% Return vector Y with the elements of the argument vector X
%                      / increasing elements if  'Order' is 'Increase'
% but ordered accordingly to |
%                      \ decreasing elements if  'Order' is 'Decrease'
%
% Example: Y=MyOrder([1 2 3], 'Decrease') returns Y=[3 2 1].
% Uses MyMax and MyMin already developed

% Copyright Ye.Gayev, Dec. 2005

L=length(X);  X1=X;

switch Order
    case {'Increase', 'increase'}
        % Rearrange X in Increasing Order
        for i=1 : L
```

```
            [Y1,I]= MyMin (X1);
            y(i)=Y1;  X1=[X1(1 : I-1),  X1(I+1 : end)];
        end
        Y=y;
    case {'Decrease', 'decrease'}
        % Rearrange X in Decreasing Order
        for i=1 : L
            [Y1, I]=MyMax(X1);
            y(i)=Y1;  X1=[X1(1 : I-1),  X1(I+1 : end)];
        end
        Y=y;
    otherwise
        disp('  ')
  disp('Другий параметр має бути "Increase" або "Decrease".')
  disp('Перевірте, будь ласка, Ваше завдання і спробуйте ще раз!')
        disp('  ')
end  % End of the Program
```

Програма працює таким чином. У MATLAB-середовищі попередньо створюється якийсь чисельний вектор v, і у командному вікні подається команда на зразок

```
>> v1=MyOrder(v,'Decrease');
```

(завдання *'Decrease'* взято у лапки!). На виході отримуємо вектор v*1* з елементами, що зменшуються. Зауважте, що такий саме результат отримаємо і у випадку команди

```
>> v1=MyOrder(v,'decrease');
```

У випадку ж

```
>> v1=MyOrder(v,'increase');
```

елементи у векторі v*1* будуть розташовані у порядку збільшення.
А от на команду

```
>> v1=MyOrder(v,'INCREASE');
```

отримаємо відповідь

```
Другий параметр має бути "Increase" або "Decrease".
Перевірте, будь ласка, Ваше завдання і спробуйте ще раз!
```

як і у випадку будь-якого іншого, незрозумілого (непередбаченого) замовлення!

Ми створили програму, яка аналогічна програмі *sort*, що вже існує в MATLAB. Цікаво, чия програма краще? Дослідження на цю тему дивіться у другій частині Підручника.

У програмі *MyOrder* ми вперше стикнулися з блоком *switch . . . end*. Інтуїтивно ясно, як він працює, але строгі пояснення дивіться у наступному підрозділі.

Добре, а як же позбавити нашу програму відміченого недоліку? Як зробити, аби вона правильно реагувала на слова завдання *'INcrease'*, *'inCREASE'*, тобто не залежала від помилково включеного регістру великих чи маленьких літер? Тут нам треба навчитись працювати з текстовим типом даних.

Таким чином, в даному підрозділі ви навчилися:

✓ Розв'язувати дві стандартні задачі теорії алгоритмів, і

✓ використовувати структурний блок ***switch . . . end.***

3.7.3. БЛОК СТРУКТУРНОГО ПРОГРАМУВАННЯ *switch . . . end*

Цей блок дозволяє програмі обирати і виконувати одну дію з багатьох можливих, і виглядає він наступним загальним чином:

switch Вираз
    case {*Значення*$_{11}$, *Значення*$_{12}$, . . ., *Значення*$_{1K1}$}
        Група операторів 1;
    case {*Значення*$_{21}$, . . ., *Значення*$_{2K2}$}
        Група операторів 2;
. . . . . .
    case {*Значення*$_{m1}$, . . ., *Значення*$_{2Km}$}
        Група операторів m;
    otherwise
        Група операторів m+1;
end ,

а працює він так. Програма доходить до цього блока з якимось значенням виразу *Вираз*, математичного, логічного

чи то текстового; далі вона порівнює це значення з листом значень вершого випадку *case* („*випадок*” у перекладі з англійської). Якщо воно співпадає хоча б з одним значенням $Значення_{1i}$, то виконується *Група операторів 1*, пропускаються усі наступні випадки *case*, і програма виходить за „операторну закриваючу скобку” *end*.

Якщо ж значення *Вираз* не співпадає ні з жодним з першої групи значень, то така само перевірка провадиться з листом значень другого випадку *case*: якщо співпадає хоча б з одним $Значення_{2i}$, де $i= 1, 2, . . ., K2$, то виконується *Група операторів 2* і „управління передається” за „скобку” *end*. Всі інші *case* – пропускаються.

Так само перевіряються усі *m* випадків *case*. В останньому *case* знов співставляється значення *Вираз* з листом значень у фігурних дужках, і якщо хоча б з одним $Значення_{mi}$ співпадає – виконується *Група операторів m*, а *Група операторів m+1*пропускається і управління передається за *end*.

Однак, якщо значення *Вираз* не співпадає з жодним значенням останнього переліку $\{Значення_{m1}, . . ., Значення_{2Km}\}$, то виконується нарешті *Група операторів m+1*, що стоїть за ключовим словом *otherwise* (воно і означає, власне, „у іншому випадку”). І далі управління передається за *end*.

У програмі *MyOrder* виразом слугувала текстова змінна *Order* (а текстові змінні мають вводитись в апострофах!). Якщо ця змінна мала значення *'Increase'* або *'increase'*, то виконувався перший цикл *for . . . end* (перестановка елементів *x* у порядку зростання), якщо ж її значенням було *'Decrease'* або *'decrease'*, то виконувався другий цикл *for . . . end* (перестановка у порядку зменшення). Якщо ж не співпадало з жодним названим значенням, то ключове слово *otherwise* виводило на групу операторів *disp*.

Радимо ознайомитись і з іншими прикладами використання операторного блоку *switch . . . end*,

>> help switch

Таким чином, в даному розділі ви навчилися програмувати прості скрипти і програми-функції,

використовувати оператори умовного виконання *if – else – end* і оператори управління обчислювальним процесом *for – end*. На закінчення пропонуємо лабораторні роботи 5 і 6 для закріплення навичок програмування.

3.8. КОНТРОЛЬНІ ПИТАННЯ ДО РОЗДІЛУ 3

1. Які властивості методів обчислення, що розглянуті в розділі 2, слід використовувати для програмування обчислень? Що саме у них повторюється, що можна автоматизувати?
2. Які види програм існують в MATLAB? Чим вони відрізняються щодо вхідної і вихідної інформації? Як викликати редактор *m*-файлів для написання програм MATLAB? Як записують *коментарі* в програмах MATLAB, для чого вони призначені? Чого слід очікувати, запрошуючи *help* для щойно створеної програми?
3. Як має виглядати програма-сценарій (*script*)? Якщо у скрипті створено якусь нову змінну, чи зберігається її значення після закінчення роботи скрипту? Чи можна у скрипті використовувати змінну, створену зовні? Чим ще відрізняються програми-сценарії від програм-функцій?
4. Як має виглядати програма-функція (*m*-функція), з чого слід її починати? Якщо у тілі функції створено якусь нову змінну, чи зберігається її значення після закінчення роботи програми? Чи можна у *m*-функції використовувати змінну, створену зовні? Як можна передавати значення змінних із зовні у функцію і, навпаки, з функції у загальне середовище MATLAB? Як у MATLAB слід звертатись до програми-функції? Скільки вхідних значень має функція? Скільки вихідних?
5. Які ви знаєте оператори умовного виконання? Які вони мають формати запису? Поясніть смислове значення

ключових слів цих операторів *if*, *elseif*, *else*, *end*. Яких значень можуть набувати *умови*, що пишуть безпосередньо за ключовими словами *if* і *elseif*? Як пишуть *умови* після ключового слова *else*? Чи можуть умови бути складними?

6. Які логічні операції використовують для запису складних умов? Як значення складної умови залежить від значень операндів?
7. Які ви знаєте оператори управління обчислювальним процесом? Чому ключові слова тих чи інших циклів *for*, *while*, *switch*, *try* мають закінчуватись ключовим словом *end*?
8. Як можна пояснити, чому програму обчислення розривної функції (3.4) ми розбили на версії *step0, step1* і *step_pi*? Чим одна версія відрізняється від іншої?
9. Поясніть, у чому полягає „нестандартність" операцій логіки в MATLAB, розділи 1.4, 1.5?
10. У чому полягає налагодження програми?

Якщо не змогли відповісти на більшість питань – радимо проробити все з початку

3.9. ЗАДАЧІ ДО РОЗДІЛУ 3

1. Побудуйте графіки трьох послідовних функцій (візуалізуйте функції), що задаються загальними формулами

1.1. $f_n(x)=\dfrac{x^{2n}}{(2n)!}$ для $n=0,1,2,...$

1.2. $f_n(x)=\dfrac{x^{2n-1}}{(2n-1)!}$ для $n=1,2,...$

(наприклад $f_3(x)$, $f_5(x)$ і $f_9(x)$, і тому подібні трійки).

2. Нехай якийсь фізичний процес „підкоряється" закону $y=e^t$. Ми „вимірюємо" величину y у певні моменти часу $t=t_1,t_2,....,t_n$ і отримуємо значення $y_1, y_2,....,y_n$. Програма

MyDiff дозволяє наближено визначити швидкість зміни фізичної величини y. Дослідіть, як проміжки часу вимірювання $\Delta t = t_i - t_{i-1}$ впливають на точність результату.

3. Таке ж саме питання щодо точності програми *MyDiff* у відношенні до фізичного процесу за законом $y = \sqrt{x}$.

4. Самостійно розробіть програму чисельного інтегрування аналогічних результатів „вимірювання". Використовуйте визначення інтегралу як площі (визначений інтеграл) з курсу вищої математики. Дослідіть, як проміжок часу вимірювання $\Delta t = t_i - t_{i-1}$ впливає на точність результату.

5. Дослідіть попередні задачі 2, 3 і 4 для того випадку, коли „експериментальні" дані $y_1, y_2, \ldots, y_n$ отримано з похибкою. Остання може бути змодельована командою MATLAB
sigma **randn*(*1,1*) ,
що генерує „випадкові числа", розподілені нормально з ***дисперсією*** *sigma=0.1* [1,51,54]. Як величина дисперсії впливає на точність розрахунків?

6. Зробіть анімаційну програму *helicopter* (Додаток 4, с. 234) більш довершеною: нехай вона „спілкується" з користувачем через діалог з командного рядочка, запитує його про бажаний колір відрізка, що обертається, про напрям обертання (за годинниковою стрілкою чи проти), про швидкість обертання. (Така задача у майбутнєму буде „одягнена" у GUI, графічний інтерфейс користувача).

7. Зробіть програму (MATLAB-команду), що обертає „трипелюсткову розу" $x = sin(3t)cos(t)$, $y = sin(3t)sin(t)$, $t \in [0,2\pi]$ за годинникової стрілкою чи проти.

8. Розробіть програму, що створює на екрані пульсуючий еліпс, або пульсуюче сонечко з „променями".

9. Розробіть діалогову програму, яка питає користувача про ціле число *N*, створює на дисплеї правильний *N* –кутник того чи іншого кольору та обертає його у замовленому напрямку.

У другій частині підручника „одягнемо” цю задачу у графічний інтерфейс користувача (GUI).

10. Створіть програму для обчислення і побудови графіків (усіма командами *ezplot*, *fplot* , *plot* і *comet*) періодичної функції, що нагадує пилу і на інтервалі $x \in [-1,1]$ подається рівнянням

$$f(x) = \begin{cases} x+1, \ if \ \ x \in [-1,0] \\ 1-x, \ if \ \ x \in (0,1] \end{cases}.$$

11. Створіть програму для обчислення і побудови графіків (усіма командами *ezplot*, *fplot* , *plot* і *comet*) „кускової” функції з періодом $T = 2\pi$, складеної з трапецій, одна з яких (на одному періоді) подається рівнянням

$$y = \begin{cases} \frac{4}{\pi} x, & if \ \ x \in [0, \frac{\pi}{2}] \\ 2, & if \ \ x \in [\frac{\pi}{2}, \pi] \\ 2(2 - \frac{1}{\pi} x), & if \ \ x \in [\pi, 2\pi] \end{cases}$$

Вказівка для задач 10 і 11: радимо розробляти програму поступово, як у випадку функцій *step01, step02* та *step_pi*, тобто (*i*) спочатку побудувати кускову функцію для усієї числової осі $-\infty < x < \infty$ не зважаючи на періодичність заданої функції; (*ii*) пересвідчитись, що створена програма „розуміє” векторний аргумент *x*; доробити програму, якщо „не розуміє”; (*iii*) спробуйте варіанти програми з урахуванням „логічного відкриття” MATLAB, та без нього; (*iv*) і на останок остаточно доробіть програму з урахуванням вказаної періодичності функції, як у програмі *step_pi*.

Сподіваємось, що Ви розв’язали більшість задач.
Вітаємо!
Можете рухатись далі.

4. ВЛАСНІ ДОСЛІДЖЕННЯ З MATLAB

Нашим вчителям,
що направляли наш шлях до науки

Хоча ви витратили на MATLAB ще небагато часу, ви вже можете починати ваші власні дослідження. Вони можуть виявитися зовсім новими і дуже важливими для людства, а скоріше за все – корисними для вас особисто, для допомоги вам в оволодінні університетськими науками, особливо таких складних і важливих, як математика і фізика.

Треба визнати, що деякі вчителі викладають свою дисципліну, математику тощо, на жаль, доволі сухо і схоластично. Тоді навіть дивно стає, як можна такою наукою захопитися? Та однак таке дійсно колись сталося з юнаками, які всім зараз відомі як великі математики – ***Леонард Ейлер***, ***Огюст Коші***, ***Карл Фрідріх Гаус*** та інші!

І ось відтепер, коли ви оволоділи основами MATLAB, математика може стати для вас наукою експериментальною, ви тепер здатні легко перевіряти шляхом „*чисельного експерименту*" ті факти, що розповідав лектор, і так довго і незрозуміло доводив...

У цьому розділі надано кілька прикладів, як саме можна проводити такі власні дослідження і експерименти за допомогою MATLAB. Сподіваємось, що вони спонукають вас на власні дослідження і пошуки...

4.1. ЦЯ НЕЗРОЗУМІЛА "ε – δ МОВА". . .

„Вища математика" в університеті починається з вивчення поняття „*границя*" (*limit* англійською), яке позначається як lim. І досвід показує, що вже тут викладачі можуть назавжди налякати студента. Ось дивіться:

Визначення 1: Послідовність чисел $\{X_k\}$, де $k = 1,2,3,...$ ***має границю*** X_*, тобто[26] $X_* \overset{Df}{=} \lim\limits_{k\to\infty} X_k$, якщо для будь-якого маленького ε знайдеться число N таке, що нерівність $|X_* - X_k| \le \varepsilon$ виконується для всіх номерів k, більших за N.

Визначення 2: Функція $f(x)$ ***має своєю границею*** значення f_* **за умов, що** x **прагне до** x_0, тобто $f_* = \lim\limits_{x\to x_0} f(x)$, якщо для будь-якого маленького ε знайдеться число δ таке, що з нерівності $|x - x_0| \le \delta$ витікає нерівність $|f(x) = f_*| \le \varepsilon$.

Аби ця "$\varepsilon - \delta$ мова" стала зрозумілою і звичною, давайте дослідимо, як саме послідовність або функція „***прагне до ...***" за допомогою буквального і самоочевидного розрахунку з MATLAB і наочного представлення результатів.

1. Розглянемо відому „*Другу Визначну Границю*[27]"

$$f_n = \lim_{n\to\infty}\left(1+\frac{1}{n}\right)^n, \qquad (4.1)$$

тимчасово забувши, хто пам'ятає, про її доведення. Давайте „прямо у лоб" обрахуємо члени цієї послідовності $S_n = \left(1+\frac{1}{n}\right)^n$. Візьмемо $n = 1$ і обчислимо $S_1 = \left(1+\frac{1}{1}\right)^1 = 2$, потім $n = 2$ і обчислимо $S_2 = \left(1+\frac{1}{2}\right)^2 = \left(\frac{3}{2}\right)^2 = 2{,}25$, далі $S_3 = \left(1+\frac{1}{3}\right)^3$, S_4, S_5 і так далі із збільшенням номеру n. Можна це робити в MATLAB „вручну",

---

[26] Тут і в інших місцях *DF* над знаком рівності означає *Definition*, *за визначенням*.

[27] Завдання 12 лабораторної роботи №1, див. наприкінці підручника.

>> n=1; S(n)=(1+1/n)^n

у перший раз, та

>> n= n +1; S(n)=(1+1/n)^n

>> n= n +1; S(n)=(1+1/n)^n

у другий і третій раз, і так далі поки не набридне:

>> n= n +1; S(n)=(1+1/n)^n

. . . . . .

і нарешті побудувати графік з „експериментальними точками”:

>> plot(S, ' rh ')

А можна процес побудови точок послідовності запрограмувати, див. гл. 3.

Цю програму, яку можна назвати *limit2.m*, доцільно побудувати наступним чином. Як завжди, можна навести довідку про неї, *help limit2*, із зрозумілим поясненням про її роботу. Після запуску з командного рядочка програма вітається так, я ви запрограмували у *Welcome.m* (Додаток 4). Далі вона проголошує, що саме вона робить:

„*Ця програма досліджує Другу Чудову Границю*”

і будує графік з першими десятьма точками послідовності (4.1). На графіку – підказка, що подальші дії треба робити через командне вікно MATLAB. Якщо ви працюєте з MATLAB версії 7, то доцільно натиснути на кнопку праворуч зверху – тоді графічне вікно буде „пристебнуте” до командного і працювати стане зручніше.

У командному вікні на вас чекає підказка: „*Введи додатне число для продовження, чи від’ємне для закінчення*!”. Вводимо будь-яке додатне число, наприклад *1* – нові 10 точок додано до графіку (Цей параметр можна контролювати, змінюючи *N* у програмі). Так можна продовжувати як завгодно довго. Коли ж ви на черговий запит введете довільне від’ємне число, *-1* наприклад, програма зупиниться і запропонує вам проаналізувати результат у вигляді графіку. Звернемось до такого аналізу.

Графічні результати для лише *N=20.* подані на рис. 4.1,А. Таке число *N* вводимо кілька разів, увесь час подовжуючи послідовність (4.1) і її зображення на рисунку. І

нарешті стане зрозумілим, що усі наступні числа (точки на графіку) продовжують зростати, але все менше і менше відрізняються від попередніх, не перевищуючи деяку границю. На рис. 4.1,Б ми дійшли до *N=10 000.* Виникне нарешті питання, **так до якого числового значення вона прагне?**

Іконкою  („збільшуюче скло") окреслимо зону, куди попадають останні точки – отримаємо щось на зразок рис. 4.1,Б: ну, тут вже очевидно, що **послідовність прагне до числа 2.7181**!

Напевно, у цьому значенні є якась похибка – адже „розрядна сітка" пам'яті комп'ютера обмежена. Більшу кількість знаків після коми можна визначити, запросивши у MATLAB новий формат і виводячи наступні дані у командне вікно:

```
>> format long
>> n= n +1;  S(n)=(1+1/n)^n
```

Вітаємо вас! Ви зробили те, що вперше зробив шотландський вчений барон Джон Непер у 1614 році – знайшли числове значення цього дивного числа *e* !

Зробити таке дослідження вам пропонувалося ще у лабораторній роботі 1. Сподіваємось, вам сподобаються й подальші відкриття з MATLAB!

4.2. ТЕЙЛОР, ФУР'Є. . . . ХТО ЩЕ?


Brook Taylor (1685–1731) Jean Fourier (1789-1830)

Імена цих вчених ви зустрічаєте, мабуть, найчастіше у курсі вищої математики. Теореми, що позначені цими іменами, схожі між собою. Оскільки вони конче важливі і у математиці і у практиці, доцільно їх дослідити додатково до університетського курсу. Графічне дослідження, яке ми тут пропонуємо, має на меті

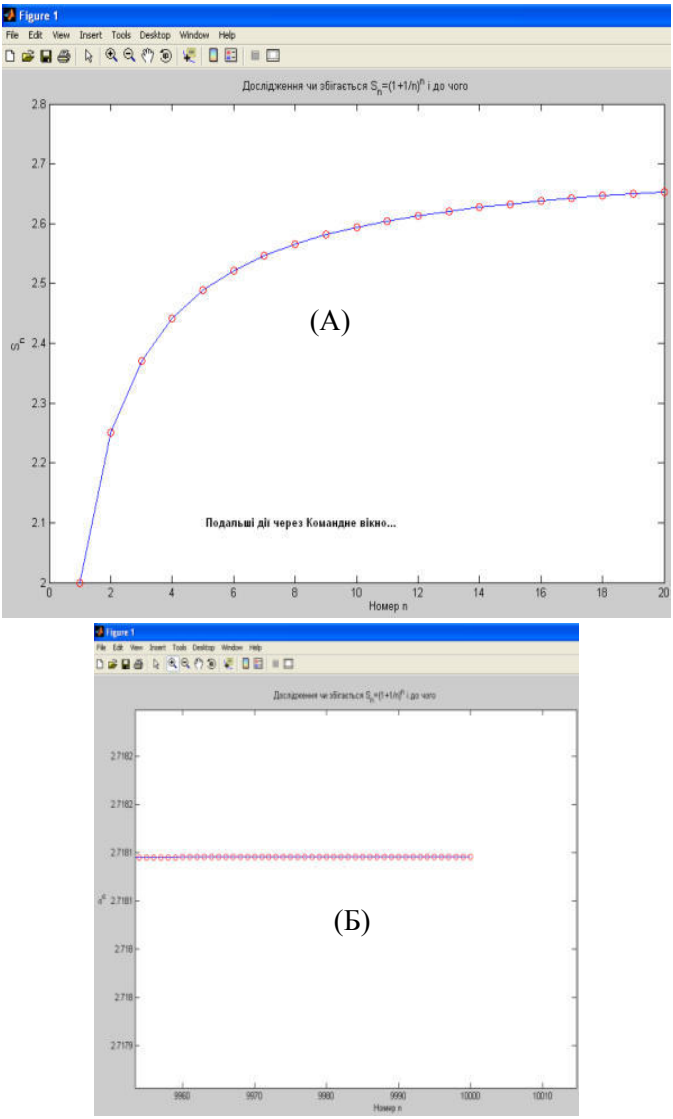


Рис. 4.1. (А) Перші 20 результатів роботи програми *limit2* для дослідження Другої чудової границі. (Б) Побудовано 10 000 членів послідовності S_n ; вони вже незмінні! Кнопка дозволяє знайти граничне значення.

пропонуємо, має на меті покращити розуміння цих проблем студентами.

Теорема 4.1 (про ряд Тейлора): майже[*)] для кожної функції $f(x)$ можна знайти многочлен $T(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n$, який її наближує; це наближення тим краще, чім більше взято доданків n.

Теорема 4.2 (про ряд Фур'є): майже[*)] для кожної періодичної функції $f(x)$ можна знайти „*тригонометричний многочлен*"

$$T(x) = a_0 + (a_1 \sin x + b_1 \cos x) + (a_2 \sin 2x + b_2 \cos 2x) + \ldots$$

$$\ldots + (a_n \sin nx + b_n \cos nx),$$

який її наближує, і це наближення тим „краще", чім більше взято доданків n.

(Коефіцієнти 1, 2, 3, . . ., n і т.д. у тригонометричних функціях задають частоту гармонічних коливань; тому, відповідні доданки називають першою, другою, третьою і т.п. *гармоніками*).

4.2.1. Проведемо графічну перевірку теореми Тейлора. Візьмемо, наприклад, періодичну функцію $y = \sin x$ – вона напевно дуже відрізняється від будь-якого многочлена! Чи діє для неї теорема Тейлора?

Етап 1. Який многочлен відповідає обраній функції? Відповідь можна знайти у підручнику вищої математики, наприклад [36,41]. А можна звернутися до самої MATLAB. Дійсно, команда ***taylor*** дозволяє знайти певну кількість (наприклад, 10) доданків у розкладі функції $y = \sin x$ у так званий „*ряд Тейлора*":

>> syms x; T10=taylor(sin(x) , 'Order', 10), pretty(T10)

дає таку формулу[28]:

[*)] „Майже" позначає, що є насправді деякі обмеження, які узнаєте у спеціальній літературі. Так само слід пояснити зміст „*краще*", див. нижче.

[28] Параметром команди '*Order*' замовлено степінь многочлена.

*T10 = x-1/6*x^3+1/120*x^5-1/5040*x^7+1/362880*x^9*

Тобто

>>pretty(T10)

```
                3          5          7          9
x - 1/6 x  + 1/120 x  - 1/5040 x  + 1/362880 x
```

Легко здогадатися, що за аналогією її можна подовжити до будь-якої кількості доданків:

$$T_{2n+1}(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \ldots\ldots + (-1)^n \frac{x^{2n+1}}{(2n+1)!}, \qquad (4.2)$$

де для $n = 0,1,2,3,\ldots$ сумуються лише непарні степені n. Це відповідає тому, що $y = \sin x$ – непарна функція. Зверніть також увагу на чергування знаків **+** та **-**!

Етап 2. Розглянемо тепер послідовність графіків для многочленів

$$T_1(x) = x, \quad T_3(x) = x - \frac{x^3}{3!}, \quad T_5(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!},$$

$$T_7(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} \qquad \text{і т.д.,}$$

позначивши їх *T1*, *T3*, *T5*, *T7* і так далі відповідно до найстаршого степеня, і порівнюючи їх з графіком вихідної функції $y = \sin x$. Це можна робити „вручну”, як будувалися графіки частинних сум „ряду Тейлора” у лабораторній роботі № 1. Для *T1* маємо[29]:

>> syms x, T1=x;

*>> H0=ezplot(sin(x), [-3*pi, 3*pi]); set(H0, 'Color','r')*

*>>hold on, H1=ezplot(T1, [-3*pi,3*pi]); axis([-3*pi, 3*pi, -4, 4])*

(результат подано на рис. 4.2,А). Можна сказати, що обидві функції співпадають на проміжку лише $-0,5 \le x \le 0,5$, а далі суттєво розрізняються.

---

[29]Формат команди *H0=ezplot(...)* дозволений у MATLAB v. 7 і вище. У MATLAB молодших версій використовуємо *ezplot(...)* без ***хендла*** *H0*. Тоді, однак, немає способу управляти кольором лінії...

Тепер у новому вікні побудуємо *T3*(*x*) і порівняємо з $y = \sin x$ на тому ж проміжку $-3\pi \le x \le 3\pi$:

```
>>figure, H0=ezplot(sin(x), [-3*pi, 3*pi]);  set(H0, 'Color','r')
>> T3=T1-1/6*x^3;
>>hold on, H3=ezplot(T3,[-3*pi, 3*pi]); axis([-3*pi, 3*pi,-4, 4])
```

Новий графічний результат подано на рис. 4.2,Б. Тепер графіки функцій співпадають на ширшому проміжку. Графік многочлену третього степеня навіть робить одне коливання, „пристосовуючись" до тригонометричної функції!

Підемо далі, побудуємо *T5*(*x*) і порівняємо з вихідною функцією на тому ж вихідному проміжку $-3\pi \le x \le 3\pi$:

```
>> T5=T3+1/120*x^5;
>>figure, H0=ezplot(sin(x), [-3*pi, 3*pi]); set(H0, 'Color','r')
>> hold on, H5=ezplot(T5,[-3*pi, 3*pi]); axis([-3*pi, 3*pi,-4, 4])
```

Можна бачити (зробіть це самі), що інтервал співпадіння двох функцій, що досліджуються, ще розширився. Зробимо кілька кроків відразу:

```
>> T7=T5-1/factorial(7)*x^7;
>> T9=T7+1/factorial(9)*x^9;
>> T11=T9-1/factorial(11)*x^11;
>> T13=T11+1/factorial(13)*x^13;
```

Зверніть увагу: ми повторюємо попередню команду і „вручну" редагуємо її; числа змінюємо на 2 більше і знаки – на протилежні. Тепер знов побудуємо графік:

```
>> figure, H0=ezplot(sin(x), [-3*pi, 3*pi]); set(H0, 'Color','r')
>>hold on, H13=ezplot(T13,[-3*pi, 3*pi]); axis([-3*pi, 3*pi,-4, 4])
```

Графічний результат на рис. 4.2,В показує, що співпадіння вихідної функції $y = \sin x$ з многочленом Тейлора *T13* має місце вже на проміжку $-6 \le x \le 6$! Крім того, якщо раніше многочлен невисокого степеня „був здатен" зробити лише одне коливання разом з тригонометричною функцією, то *T13* робить вже два коливання!

Радимо вам продовжити це дослідження до многочленів Тейлора ще більшого степеня. Коли співпадіння буде досягнуто на усьому інтервалі $-3\pi \le x \le 3\pi$ – спробуйте

піти далі, до многочленів ще вищого степеня, але вже на ширшому проміжку, скажімо $-5\pi \le x \le 5\pi$. Радимо вам також провести аналогічні дослідження і для інших функцій, як то запропоновано наприкінці лабораторній роботі № 3.

Ще важливо **звернути увагу:**

1. **У якому сенсі многочлен Тейлора „*наближує*" вихідну функцію?** У буквальному: „*відстань*" між ними або дорівнює нулю, або прагне до нуля із подальшим збільшенням степеня.
2. **Чи можна бути упевненим у цьому нашому відкритті?** Мабуть, це потребує ***доведення***, що відкритий ефект не є випадковим, лише для даної функції, що хоча ми і зупинилися на якомусь степені *N*, та він має місце і для усіх подальших степенів; що теорема Тейлора справедлива не для поодинокій функції, а для широкого класу функцій. Нарешті, за яким загальним правилом знаходити „розклад у ряд Тейлора"? От на такі узагальнення комп'ютер поки що (поки що?) не здатен...

За цих умов можна стверджувати, що Теорема 5.1 має загальне значення. Навіщо вона? Пригадайте, що в момент народження комп'ютер „знав" лише дві операції – додавання (віднімання) та множення. Усе інше – корінь квадратний, синус, косинус, експоненту, логарифм він „навчився" за допомогою рядів Тейлора.

Етап 3. Виникає бажання зробити MATLAB-програму, що здатна проводити дослідження, що описане, і для довільних функцій. Пропонуємо вам самостійно, за описаним алгоритмом, зробити програму *MyTaylor*, яка працює у формі діалогу з користувачем з командного рядочка і дозволяє досліджувати графічно і експериментувати з будь-якою функцією.

4.2.2. Тепер щодо графічної перевірки теореми Фур'є 4.2. Власне, деякі задачі ви вже розглядали у Лабораторній роботі 2 і у розділі 3. За таким саме зразком, як частинні суми ряду Тейлора, поводимось і тут; два результуючі графіки подано на рис. 3.10.

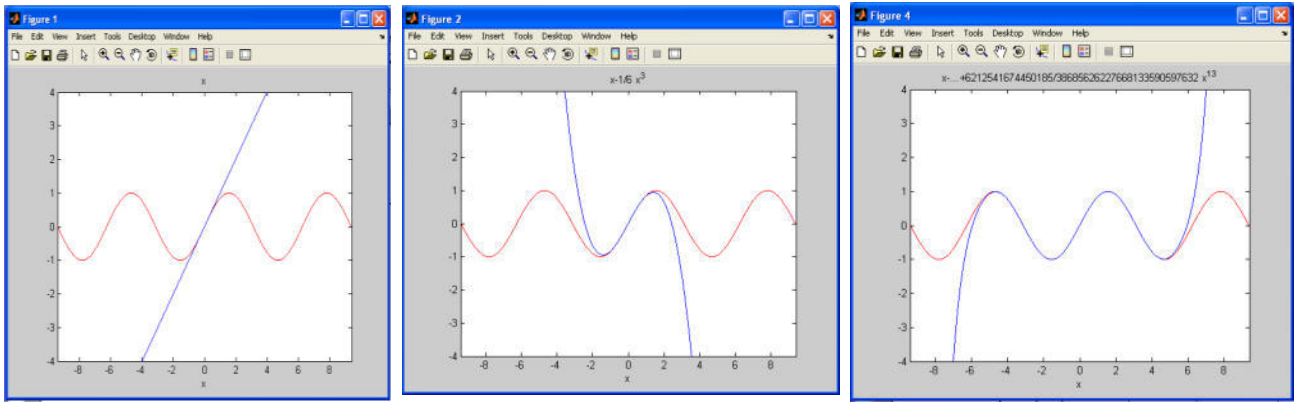


Рис. 4.2. Порівняння частинних сум ряду Тейлора (4.2) з вихідною функцією $y = sin\, x$ на інтервалі $-3\pi \le x \le 3\pi$: (А) взято 1 доданок $T1(x)$; (Б) взято доданок третього степеня $T3(x)$; (В) п'ятого степеня $T5(x)$. Все сильніше, на все більшому інтервалі наближуємось до вихідної функції $y = sin\, x$.

Та дещо до нього треба додати.

Зверніть увагу, що характер наближення частинних сум до вихідної функції – у даному випадку такою є розривна „сходинкова функція" (3.4) – суттєво відрізняється від того, що було з рядом Тейлора. Тепер немає буквального зближення функцій, а є коливання навколо граничної функції. Математики запропонували таке розуміння цього *наближення*: якщо сумувати площі тих фігур, що утворюються біля відрізків прямої, то вона прагне до нуля із збільшенням кількості доданків тригонометричного полінома *n*. Така *умова збіжності* строго виражається через інтеграл і покладена в основу математичних доведень усіх властивостей рядів Фур'є. Саме у цьому сенсі слід розуміти Теорему 4.2.

І ще зверніть увагу: якщо гранична функція (або, можна сказати: функція, яка *розкладається* у ряд Фур'є) є розривною, то коливання тригонометричних поліномів у точках розриву ніколи не зменшуються до нуля! Це відкриття зробив відомий термодінамік Дж. Гібс [41]. Нікуди не подітися від „*ефекту Гібса*"! Сьогодні ряди Фур'є „запряжені працювати" і у вашому радіо, і у телевізорі, і у вашій „мобілці", і у відомому графічному форматі ***.jpg***. Як тільки помітили певне викривлення сигналу або зображення – то знайте, це „*ефект Гібсу*" винний!

4.3. ІТЕРАЦІЇ І РЕКУРСИВНІ ПРОГРАМИ

Слово ***ітерація*** прийшло до нас з латини (так само як слово *iteration* до англійської мови) і означає *багатократне повторення*. Цей термін і поняття, за ним сховані, відіграють, однак, надзвичайну важливу роль у математиці, програмуванні і в усіх математизованих дисциплінах загалом. Тому, розглянемо це поняття більш докладно.

Якщо якесь рівняння відносно x – чи то алгебраїчне, чи диференціальне, чи то інтегральне – вдається записати у вигляді

$$x = f(x) \tag{4.3}$$

(де $f(x)$ будь-яка послідовність дій над x – алгебраїчних, операцій диференціювання або інтегрування), то виникає спокуса взяти якесь початкове значення $x = x_0$, визначити наступне за формулою $x_1 = f(x_0)$, і продовжувати цю процедуру за формулою

$$x_{n+1} = f(x_n), \quad n = 2,3,\ldots \tag{4.4}$$

Оце і є ітерації, повторення. **Чи можна так знайти розв'язок вихідного рівняння** (4.3)**?**

Не кожний науковець, який використовує такий *метод ітерацій* (особливо, у практичних або фізичних задачах), задає собі таке питання. А можливо, він покладається на таку теорему:

Теорема 4.3: Якщо послідовність ітерованих величин $\{x_1, x_2, \ldots, x_n, \ldots\}$ збігається до якогось значення (назвемо його x_*), і функція $f(x)$ є „достатньо якісною" (наприклад – неперервною), то граничне значення x_* дійсно дає розв'язок вихідного рівняння.

Збігається послідовність чи ні – практичний висновок роблять, спостерігаючи за різницею двох послідовних значень[30] $|x_{n+1} - x_n|$, [1]. Якщо вона стає менше за бажану

---

[30]Немає складнощів, коли x_n – числа. Однак, що являє собою "різниця" $|x_{n+1} - x_n|$ у випадках, коли x_n більш складний об'єкт, потребує обговорення. Якщо x_n – вектор (розв'язок СЛАР), то поняття *відстані* є більш складним. А якщо $x_n(t)$ – функція від t, то „*близькість*", „*метрику*" визначають як площу між лініями $x_n(t)$ і $x_{n-1}(t)$, тобто як $\int_a^b |x_n(t) - x_{n-1}(t)|\,dt$. Подробиці дивіться у курсах функціонального аналізу.

точність, $\varepsilon = 0{,}01$ або навіть $\varepsilon = 0{,}0001$, то обчислення припиняють і радіють знайденому „*наближеному розв'язку із заданою точністю* ε".

Доведення є простим**.** Факт збіжності послідовності до граничного значення записують як $\lim\limits_{n\to\infty} f(x_n) = x_*$. Розглянемо $\lim\limits_{n\to\infty} f(x_n)$. У випадку неперервної функції $f(x)$, обчислення границі „можна внести під знак функції", $\lim\limits_{n\to\infty} f(x_n) = f(\lim\limits_{n\to\infty} x_n)$. А на підставі припущення про збіжність остання величина є $f(x_*)$. Таким чином, застосування операції $\lim\limits_{n\to\infty}$ до співвідношення (4.4) дає $x_* = f(x_*)$, тобто x_* дійсно є коренем рівняння (4.3). Довели!

У багатьох випадках послідовність $\{x_1, x_2, \ldots, x_n, \ldots\}$ дійсно збігається, і **ітерування** дає простий і ефективний метод розв'язку багатьох задач. Ось чому умови збіжності ретельно вивчаються у курсах функціонального аналізу!

Однак, випадки розбіжності послідовності $\{x_1, x_2, \ldots, x_n, \ldots\}$ теж цікаві з наукової точки зору. Таке ми розглянемо у наступному розділі 4.4. Зараз же зупинимось на спорідненому і теж важливому поняттю ***рекурсії.***

Визначення. Послідовність об'єктів (елементів) $\{u_0, u_1, u_2, \ldots, u_n, \ldots\}$ називають *рекурентною послідовністю*, якщо кожний її елемент u_n обчислюється через попередній елемент u_{n-1} за допомогою певного правила, формули

$$u_n = f(u_n), \qquad (4.5\text{А})$$

або навіть через два а то й три попередніх елементи за певною формулою

$$u_n = f(u_{n-1}, u_{n-2}, u_{n-3}). \qquad (4.5\text{Б})$$

Зрозуміло, що з якогось елементу u_0 треба починати. Його називають *нульовим*, або *початковим*. У випадку рекурентної формули (4.5Б) одного такого елементу вже недостатньо, а потрібно вже три початкових елементи u_0, u_1 і u_2. Він (вони) мають бути заданими.

Приклад 1 надають відомі зі школи арифметична та геометрична прогресії.

Арифметична: $u_1 = u_0 + q$, $u_2 = u_1 + q$, $u_3 = u_2 + q$, . . .,

$u_n = u_{n-1} + q$, і т.д.

Геометрична: $u_1 = qu_0$, $u_2 = qu_1$, $u_3 = qu_2$, . . .,

$u_n = qu_{n-1}$, і т.д.

Якщо u_0, початковий елемент, та q відомі, то й усю послідовність можна побудувати. Ці послідовності настільки прості, що легко отримати остаточний вигляд їх довільного, n-го члену через u_0 та q: $u_n = u_0 + nq$ для арифметичної і $u_n = q^n u_0$ для геометричної. Однак бувають і більш складні випадки.

Приклад 2. Італійський математик Фібоначі запропонував у 1228 році таке правило утворення послідовності, що отримала його ім'я:

$$u_n = u_{n-1} + u_{n-1}, \text{ причому } u_0 = 0 \text{ і } u_1 = 1. \quad (4.6)$$

Обчислюємо далі, для $n = 2,3,\ldots$ за вказаним правилом:

$u_2 = 1$, $u_3 = 2$, $u_4 = 3$, $u_5 = 5$, $u_6 = 8$, $u_7 = 13$, $u_8 = 21$,

$u_9 = 34$, $u_{10} = 55$, і так далі до нескінченності.

Для чисел Фібоначі вже зовсім не легко отримати формулу їх загального члену. Та для знаходження чисел Фібоначі нехай нам допоможе комп'ютер! Можливо скласти програми за двома алгоритмами.

Програма 1, *My1Fibonachi.m*.

Якщо треба обчислити число Фібоначі номер *N*, тобто, *My1Fibonachi*(*N*), то кожного разу починаємо с першого

$u_0 = 0$, з другого $u_1 = 1$, і так далі за формулою до потрібного *N*, дивіться лістинг програми:

```
function Y=My1Fibonacci(N)
%знаходження чисел Fibonacci за їхнім номером
%починаючи з двох перших
% і без використання РЕКУРСІЇ
%Приклад: >>My1Fibonacci(10)%маємо ans=55
N1=1;
if N==1
    Y=0;
elseif N==2
    Y=1;
else
    Y0=0; Y1=1;
    while N1<N
        Y=Y1+Y0;
        N1=N1+1;
        Y0=Y1;Y1=Y;
    end
end
```

Можливий, однак, і принципово інший алгоритм обчислень. Розпочинаємо відразу від потрібного номера *N*. Це число знаходиться за двома попередніми $u_n = f(u_{n-1}, u_{n-1})$, де $f(u,v)$ визначається формулою (4.6). В свою чергу, оскільки аргументи u і v знаходимо за попередніми номерами, то

$$u_n = f(f(u_{n-2}, u_{n-3}), f(u_{n-3}, u_{n-4})).$$

Ті числа Фібоначі з меншими номерами залежать від попередніх у свою чергу, тобто

$$u_n = f(f(f(u_{n-3}, u_{n-4}), f(u_{n-4}, u_{n-5})), f(f(u_{n-4}, u_{n-5}), f(u_{n-5}, u_{n-6})))$$

і так далі, аж поки дійдемо до u_0 і u_1, що задані. Така процедура „f **через** f **, а та знову через** f ” нагадує змію, яка поїдає сама себе за свій хвіст, або відому гравюру голландця М. Ешера „Рука, що малює сама себе”. Тим не

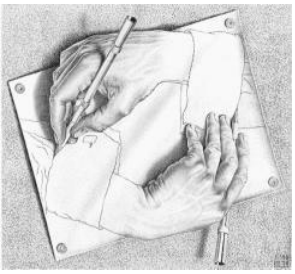


Рис. 4.3. М.Ешер „Рука, що малює сама себе”. Аналогія рекурсії.

менше, таке можливо в програмуванні!... Побудуємо таку програму.

Програма 2, *My2Fibonacci.m*.

Аби обчислити число Фібоначі номер *N*, тобто *My2Fibonacci*(*N*), ця програма звертається до *My2Fibonacci*(*N-1*) і *My2Fibonacci*(*N-2*), тобто сама до себе.

Лістінг програми *My2Fibonacci.m*

```
function Y=My2Fibonacci(N)
%знаходження чисел Fibonacci за їхнім номером
% з використанням РЕКУРСІЇ.
%Приклади використання:
% >>My2Fibonacci(10)%маємо ans=55
% >>My2Fibonacci(1)%маємо ans=1
  if N==1
       Y=0;
  elseif N==2
       Y=1;
  else
       Y=My2Fibonacci(N-1)+My2Fibonacci(N-2);
  end
```

Програми, які звертаються самі до себе, називаються **рекурсивними**. Зазвичай вони виглядають набагато простіше, аніж інші. Механізм рекурсії був заборонений у багатьох ранніх алгоритмічних мовах (Algol, Fortran), та дозволений у модернових мовах, таких як C^{++}, Java і MATLAB.

Яка з двох створених програм краще? Таке питання розглянемо далі, у розділі 4.7.

Увага: існує обмеження на кількість рекурсії (звертаній програми до себе самої). Комп’ютер може видати

??? Out of memory.

Дякуємо Windows, що захищає комп'ютер від випадкової помилки!

4.4. ПОСТРІЛЯЄМО?

Чого це ми все про математику? В університетській фізиці теж вистачає проблем, де MATLAB буде вам добрим помічником. Дійсно, давайте розглянемо, наприклад, задачу механіки про траєкторію і політ тіла (точніше – матеріальної точки), що кинуто з певною початковою швидкістю під кутом до горизонту. Із повсякденного досвіду ми знаємо, що полетить воно у далину і у верх одночасно, досягне певної граничної висоти, почне падати (продовжуючи, однак, горизонтальній рух) і десь там упаде на землю.

Виникають питання: якої висоти тіло досягне? Як далеко долетить? Як це залежить від кута „пострілу" α, від швидкості попутного або протилежного вітру, від розмірів і маси тіла?

Задачу цю не „з неба" взято. Ще у недалекому минулому, коли держави воювали одна з одною, це була найактуальніша задача – великі колективи людей обраховували траєкторії снарядів на (ще тоді!) арифмометрах, саме для неї створювалися перші – дуже засекречені – електролампові комп'ютери, засекречувалися найкращі (найшвидші та з найменшою кількістю дій) математичні алгоритми! Потім настала космічна ера – і ті самі машини й алгоритми почали прокладати траєкторії космічним човнам і супутниками...

Та й сьогодні сучасні задачі можуть у якоїсь частини привести до такої, вже стандартної, задачі механіки. Одному з авторів цієї книжки, наприклад, довелося розробляти метод теплового розрахунку бризкальної системи охолодження (БСО) Запорізької атомної електростанції. Ця БСО ЗаАЕС призначена для охолодження величезного потоку дніпрової води, біля *70 м*^{3}/*сек*, після її проходження конденсаторів

АЕС. Бризкальна система являє собою велику кількість фонтанів висотою *6 м*. Завдяки розвиненій поверхні контакту крапель з атмосферним повітрям (у тисячу разів більше, аніж була б, якби вода текла як річка чи через ставок), краплі ефективно охолоджуються, збираються разом у каналі та знов подаються у конденсатори для його подальшого охолодження. Зрозуміло, що перш ніж обчислювати охолодження, треба розраховувати рух крапель...

Так от, давайте постріляємо нашим снарядом – без усякої зброї, лише за допомогою друга-MATLABa. Аби це здійснити на екрані комп'ютера, треба спочатку алгоритмізувати задачу, що вже й раніше робилося. Та задачі, пов'язані з фізичною реальністю, вимагають урахування її певних законів. Потрібно зробити наступні кроки:

4.4.1. Виробити математичну модель процесу на підставі фізичних законів, якщо треба – розумне спрощення моделі;

4.4.2. Реалізувати цю математичну модель у середовищі MATLAB;

4.4.3. Провести чисельні експерименти на комп'ютері та аналіз результатів.

А далі виникне питання: ну, працює модель; а чи можна їй вірити, чи ***достовірна*** вона? Таке запитання виникає у кожній задачі фізичного або практичного змісту. Тому, і такий етап треба розглянути – достовірність розв'язку.

Цей розділ трохи довший за інші розділи даної книжки. Це тому, що він включає усі названі питання від постановки проблеми до її розв'язку і критичного аналізу, і тим дає приклад реального закінченого наукового дослідження. Адже дехто з наших юних читачів стане науковцем!

Підрозділи, що помічено знаком , вимагають більших знань, аніж вони можуть бути у першокурсника, а саме – розуміння теорії звичайних диференціальних рівнянь. У такому випадку студент може пропустити цей підрозділ, а потім до нього повернутися.

Отже починаємо з кроку 4.4.1.!

4.4.1. Математична модель процесу

Про фізичні закони, які „керують" рухом тіла (матеріальної точки масою m) ви вчили у школі і у курсі механіки університету. У найбільш загальній формі, рух тіла підкоряється третьому закону Ньютона

$$m\vec{a} = \vec{F}, \tag{4.7}$$

де $\vec{a}$ – вектор прискорення матеріальної точки і $\vec{F}$ – сила, що на неї діє. Обговоримо ці поняття.

Вектор прискорення $\vec{a}$ є „швидкість зміни у часі t швидкості тіла $\vec{V}$", тобто похідна $\vec{a} = \frac{d\vec{V}}{dt}$. У свою чергу, швидкість $\vec{V}$ складається з трьох компонентів $\{V_x, V_y, V_z\}$, тобто швидкостей V_x у напрямку" Ox, V_y у напрямку Oy і V_z у напрямку Oz, причому кожна з останніх є похідною від відповідної координати як функції часу

$$V_x = \frac{dx}{dt}, \qquad V_y = \frac{dy}{dt}, \qquad V_z = \frac{dz}{dt}. \tag{4.8}$$

Будемо вважати, що тіло є сферичною кулькою. Тоді його маса m є добутком об'єму $\Omega = \frac{4}{3}\pi r^3$ на щільність матеріалу ρ_2, тобто $m = \rho_2\Omega$.

Сила $\vec{F}$ також є тривимірним вектором з трьох компонентів $\{F_x, F_y, F_z\}$. У нашому випадку вона складається з сили тяжіння $m\vec{g}$ і сили опору оточуючого середовища $\vec{F}_{опору}$, рис. 4.4. Проаналізуємо їх.

Природно, що ми обираємо вісь Oz спрямованою вертикально уверх. Тоді з трьох компонентів вектору тяжіння лише один відрізняється від нуля, $m\vec{g} = \{-mg, 0, 0\}$. Тут літерою g позначено прискорення земного тяжіння. Ще з часів ***Галілео Галілея*** відомо, що ця величина в умовах Землі

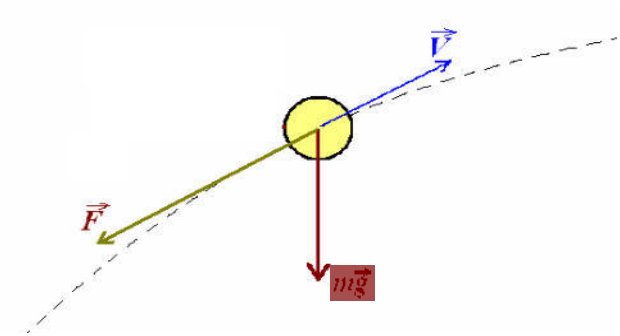


Рис. 4.5. Схема руху кульки уздовж траєкторії (пунктир) і сил, що на неї діють.

постійна і дорівнює $g = 9{,}81 \ \ м/с^2$ – це буде перша „цеглинка" у фундаменті нашої моделі.

Сила опору газового середовища $\vec{F}_{опору}$ має усі три складові, F_x, F_y і F_z. Вона досліджувалася в останні 150 років у багатьох аеро- і гідродинамічних лабораторіях світу, особливо під кутом зору створюваних кораблів і літаків. Навіть з особистого досвіду ви щось знаєте, помічали, що опір повітряного середовища спрямований протилежно вектору швидкості $\vec{V}$, як то показано на рис. 4.4. А точні аеродинамічні досліди дозволяють визначити і величину цього опору $\vec{F}_{опору}$. Знайдено, що для незначної швидкості обтікання та у випадку маленьких сфер, як то буває для крапель рідкого дощу, ця сила пропорційна швидкості крапель (першому степеню швидкості крапель)

$$\vec{F}_1 = -c_1 \rho_1 \vec{V}, \qquad (4.9А)$$

де ρ_1 – щільність газу, що обтікає, і у випадку повітря є $\rho_1 \approx 1{,}3 \ \ кг/м^3$. А c_1 є емпіричним коефіцієнтом, тобто таким, що визначається не з загальнотеоретичних міркувань, а вимірюється експериментально для кожних конкретних умов. Цю формулу для опору називають *лінійним законом*

опору, і тому силу і емпіричний коефіцієнт позначено індексом 1.

Зауваження. У фізичних задачах, так само як і у задачах програмування, важливо надавати змінним зрозумілі, „зручні" імена та індекси.

Але ж у більшості випадків сила опору пропорційна квадрату швидкості,

$$\vec{F}_2 = -\tfrac{1}{2} c_2 \rho_1 \vec{V} \,|\vec{V}|\, S\,, \qquad (4.9Б)$$

де c_2 емпіричний коефіцієнт, що відповідає такій *квадратичній формулі* опору. Встановлено, що для сфери він приблизно постійний і дорівнює $c_2 \approx 0{,}40$ – це буде другою „цеглинкою" математичної моделі (множник $\frac{1}{2}$ у формулі – це наслідок домовленості науковців; можна й не множити на $\frac{1}{2}$, але тоді брали б c_2 двічі більшим). А S у цій формулі – площа проекції тіла, що обтікається; вона дорівнює $S = \pi r^2$, де r – радіус сферичної кульки. Квадратичний закон (4.9Б) встановили відомий автор „Ейфелевої вежі" у Парижі та німецький засновник аеродинаміки Людвіг Прандтля [57].

Коли середовище нерухоме (немає вітру, $\vec{W} = 0$), то вектори $\vec{F}_{опору} = \vec{F}_2$ і $\vec{V}$ мають однаковий напрямок, позначення вектору можна опустити, і виходить, що величина сили опору пропорційна квадрату швидкості, $F_{опору} \sim -V^2$ – ось на якій підставі закон (4.9Б) називають *квадратичним*, на відміну від *лінійного* закону (4.9А). Нас має цікавити, однак, той практичний випадок, коли вітер присутній, і тому $\vec{W} \neq 0$. Очевидно, що в наслідок вибору системи координат цей вектор має ненульовими лише горизонтальні складові, $\vec{W} = \{W_x, W_y, 0\}$, бо вертикальна компонента $W_z = 0$. Це призводить до того ускладнення, що у формулі опору (4.9А) або (4.9Б) ми маємо враховувати не

абсолютну швидкість кульки відносно поверхні землі $\vec{V}$, а її **швидкість відносно оточуючого газового середовища** $\vec{V}_{відносна} = \vec{V} + \vec{W}$. А це означає, що абсолютне значення відносної швидкості є

$$|\vec{V}_{відносна}| = \sqrt{(V_x - W_x)^2 + (V_y - W_y)^2 + V_z^2} ,$$

і тому „по-компонентний" розклад квадратичної сили (4.9Б) дає доволі складні вирази

$$F_x = -\tfrac{1}{2} c_2 \rho_1 (V_x - W_x)\sqrt{(V_x - W_x)^2 + (V_y - W_y)^2 + V_z^2}\, S ,$$

$$F_y = -\tfrac{1}{2} c_2 \rho_1 (V_y - W_y)\sqrt{(V_x - W_x)^2 + (V_y - W_y)^2 + V_z^2}\, S ,$$

$$F_z = -\tfrac{1}{2} c_2 \rho_1 V_z \sqrt{(V_x - W_x)^2 + (V_y - W_y)^2 + V_z^2}\, S .$$

На цій підставі можемо нарешті надати остаточний вигляд диференціальному рівнянню третього закону Ньютона (4.7), „розписавши" його по компонентах на три рівняння:

$$m\frac{dV_x}{dt} = -\tfrac{1}{2} c_2 \rho_1 (V_x - W_x)\sqrt{(V_x - W_x)^2 + (V_y - W_y)^2 + V_z^2}\, S ,$$

$$m\frac{dV_y}{dt} = -\tfrac{1}{2} c_2 \rho_1 (V_y - W_y)\sqrt{(V_x - W_x)^2 + (V_y - W_y)^2 + V_z^2}\, S , \qquad (4.10)$$

$$m\frac{dV_z}{dt} = -mg - \tfrac{1}{2} c_2 \rho_1 V_z \sqrt{(V_x - W_x)^2 + (V_y - W_y)^2 + V_z^2}\, S .$$

Доволі складні рівняння вийшли! Усі три треба розв'язувати у загальному випадку тривимірного руху. Але доречно спростити задачу і розглядати рух лише у вертикальній площині Oxz , що співпадає з напрямком вітру $\vec{W}$. Тоді $V_y = 0$, $\vec{W} = \{W,0,0\}$ і перші два рівняння руху можна спростити. Друге рівняння зникає зовсім, а перше і третє приймають вигляд:

$$m\frac{dV_x}{dt} = -\tfrac{1}{2} c_2 \rho_1 (V_x - W)\sqrt{(V_x - W)^2 + V_z^2}\, S , \qquad (4.11)$$

$$m\frac{dV_z}{dt} = -mg - \tfrac{1}{2}c_2\rho_1 V_z\sqrt{(V_x - W)^2 + V_z^2}\,S\,.$$

Повсякденна інтуїція підказує, що для прогнозування руху снаряду ще треба враховувати надані йому у початковий момент $t = 0$ компоненти швидкості V_{x0} і V_{z0} (або значення швидкості V_0 і кута α, під яким її надано – у цьому випадку $V_{x0} = V_0\cos\alpha$ і $V_{z0} = V_0\sin\alpha$). Теорія диференціальних рівнянь також підказує, що без таких додаткових умов визначити розв'язок однозначно не можна. Таке співпадіння свідчить, що нашою інтуїцією треба користуватися!

Ще раз подивіться на рівняння: у них невідомими є дві компоненти швидкості як функції часу $V_x = V_x(t)$, $V_z = V_z(t)$. А де ж траєкторія нашого снаряду? Значить, до розгляду треба ще додати *кінематичні рівняння* (4.8). І крім цього – завдати положення тіла у початковий момент часу $x(0) = 0$, $z(0) = 0$. Тепер задача сформульована остаточно! Ці остаточні рівняння виглядають так:

$$m\frac{dV_x}{dt} = -\tfrac{1}{2}c_2\rho_1(V_x - W)\sqrt{(V_x - W)^2 + V_z^2}\,S\,,$$

$$m\frac{dV_z}{dt} = -mg - \tfrac{1}{2}c_2\rho_1 V_z\sqrt{(V_x - W)^2 + V_z^2}\,S\,, \qquad (4.12)$$

$$V_x = \frac{dx}{dt}, \qquad V_z = \frac{dz}{dt}.$$

Початкові умови

$$V_x(0) = V_0\cos\alpha\,,\ V_z(0) = V_0\sin\alpha\,, \qquad (4.13)$$
$$x(0) = 0\,,\ z(0) = 0\,.$$

Можна йти далі.

4.4.2. Реалізація математичної моделі ∾ у середовищі MATLAB

Цей розділ помічено знаком ∾, бо він вимагає знання теорії звичайних диференціальних рівнянь, [16-19,34-41]. Студент може пропустити цей підрозділ, працювати відразу з

4.5.3, а повернутися до нього після вивчення відповідного матеріалу.

Таким чином, створена математична модель руху снаряду задається чотирма рівняннями (4.12) від чотирьох невідомих $x(t)$, $z(t)$, $V_x(t)$ і $V_z(t)$ однієї незалежної змінної t, часу, і з початковими умовами (4.13). Це – система звичайних диференціальних рівнянь першого порядку (ЗДР). У ній m, c_2, ρ_1, V_0 і α є незмінними параметрами задачі.

Тепер можна тимчасово забути про фізичний зміст задачі і зайнятися самою математикою. Елементи теорії ЗДР і команди MATLAB для розв'язування початкових задач для ЗДР розглянуто в книжках [2,5,15,34-41].

Відповідно до методики MATLAB, систему ЗДР слід спочатку переписати у так званій стандартній формі

$$\frac{d}{dt}\begin{pmatrix} y_1 \\ y_2 \\ \dots \\ y_n \end{pmatrix} = f\left(\begin{pmatrix} y_1 \\ y_2 \\ \dots \\ y_n \end{pmatrix}, t\right).$$

Якщо позначити $x = y_1(t)$, $z = y_2(t)$, $V_x = y_3(t)$ і $V_z = y_4(t)$, то маємо таку систему ЗДР:

$$\begin{aligned} \frac{dy_1}{dt} &= y_3, \\ \frac{dy_2}{dt} &= y_4 \\ \frac{dy_3}{dt} &= -A(y_3 - W)\sqrt{(y_3 - W)^2 + y_4^2} \\ \frac{dy_4}{dt} &= -g - Ay_4\sqrt{(y_3 - W)^2 + y_4^2} \end{aligned} \quad (4.14)$$

Заодно з'ясувалося – приємна несподіванка – що кількість фізичних параметрів задачі з чотирьох зменшилася до трьох, до g, W і $A = \frac{c_2\rho_1 S}{2m}$. Ще два параметри, однак,

присутні у початкових умовах (4.13), які у нових змінних слід сформулювати як

$$y_1(0) = 0\,, \quad y_2(0) = 0\,, \quad y_3(0) = V_0 \cos\alpha\,,$$
$$y_4(0) = V_0 \sin\alpha\,. \qquad (4.15)$$

Далі, у відповідності до методики MATLAB[31] створюємо *m*-функцію, що відображає праві частини рівнянь руху (4.14). Наводимо лістинг цієї *m*-функції *Right2.m* у Додатку 7. (Номер *2* у назві нагадує, що ми маємо справу з квадратичним законом руху (4.9Б)). Можна також написати цю програму у складі головної програми, що „керує" обчисленнями і виводить результати в анімаційній графічній формі, яку ми назвали *Ball.m*. Тепер можемо виконувати „***чисельні експерименти***" з командного рядочка MATLAB.

4.4.3. Чисельні експерименти на комп'ютері, аналіз результатів[32]

У програмі *Ball* покладено, що маса кульки дорівнює *m=200* грамів, але ми можемо обирати довільними і експериментувати з її радіусом *r* (задаємо у сантиметрах), наданої початкової швидкості *V0* (у метрах за секунду), кутом „стрільби" до горизонту *Alfa* (у градусах) і швидкості вітру *Wind* (задаємо теж у метрах за секунду, але вважаємо додатнім, якщо вітер у напрямку „стрільби", і від'ємним, якщо вітер протилежний напрямку „стрільби"). Ось приклад команди:

>> r=6; *V0=10*; *Alfa=60*; *Wind=0*; *Ball*(*r*, *V0*, *Alfa*, *Wind*)

За такою командою побачимо картину руху кульки. Її певні моменти зображені на рис. 4.6 і 4.7. У заголовку анімаційного графіку відображені умови вітру, за яких рух відбувається. Біля точки викиду (*0,0*) – умови стрільби (початкова швидкість і кут). Синхронно із рухом кульки

---

[31] Запросіть в МАТЛАБ допомогу, *>>help ode45*, або*>>help ode23*, де *ode45* – один з чисельних методів розв'язку звичайних диференціальних рівнянь.

[32] Програму *Ball* можна або самому набрати за текстом Додатку 7, с. 236, або позичити з сайту *http://www.gayev-shubina.narod.ru*.

змінюються компоненти поточної швидкості V_x і V_z. Експерименти можуть полягати у вивченні, наприклад, траєкторії кульки у залежності від вітру:

```
>> Wind=5;      Ball(r, V0, Alfa, Wind)
>> Wind=- 10;  Ball(r, V0, Alfa, Wind)
```

У цих командах параметри *r*, *V0* і *Alfa* не задавалися, бо після першої команди вони не змінювалися. З таких експериментів випливає, природно, суттєва залежність траєкторії кульки від вітру. Вітер вздовж її руху сприяє йому, а вітер назустріч (*Wind* < *0*) – пригальмовує. При сильному зустрічному вітрі кулька навіть сама змінює напрям руху, а траєкторія може нагадувати петлю, як на рис. 4.6.

Аналогічним чином цікаво експериментувати з кутом „стрільби" *Alfa*. Зробіть самостійно таке дослідження:

Завдання 4.1: за умов відсутності вітру *Wind = 0* дослідіть вплив кута „стрільби" *Alfa* на дальність і висоту „стрільби". Яким „емпіричним" рівнянням (пригадайте розділи 2.3) можна описати залежність висоти від кута, $H = H(\alpha)$? А залежність дальності від кута $L = L(\alpha)$? Цікаво, а який кут „стрільби" дозволяє досягти найбільшу дальність? А як це залежить від вітру?

Підказка: у графічному вікні MATLAB вказані параметри траєкторії зручно знаходити за допомогою „*числового курсору*" (*Data Cursor*) 

, зображеного на рисунку.

Сподіваємось, що вам подобається „стріляти кулькою" на екрані вашого комп'ютера. Але виникне питання: ну добре, працює модель. А чи можна їй вірити, чи відповідає вона тому, що трапилось би у реальному випадку, чи ***достовірна*** вона?

А є й такі студенти, які усе піддають сумніву... Вітаємо таких! Давайте трохи зупинимось і перевіримо себе. **Наскільки достовірною є щойно створена модель?** Слід відрізняти фізичну достовірність і математичну правильність (адже й за умов правильної фізичної моделі можна робити

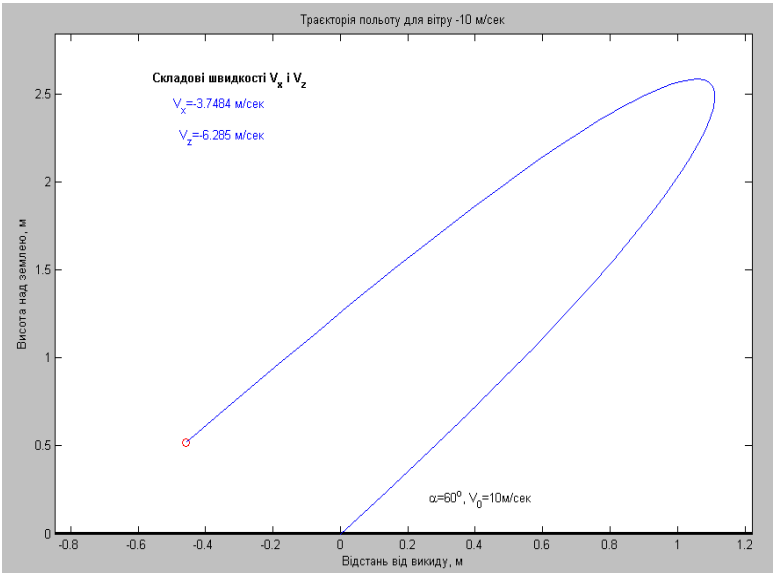


Рис. 4.6. Траєкторія кульки при зустрічному вітрі *Wind*= –10 м/сек нагадує петлю.

математичні, обчислювальні помилки) на кожному з наших кроків. Таким чином треба обговорити питання:

4.4.4. Математична достовірність розв'язку

4.4.5. Фізична достовірність моделі.

4.4.4. МАТЕМАТИЧНА ДОСТОВІРНІСТЬ РЕЗУЛЬТАТІВ

Вам зрозуміло, чому слід відрізняти **математичну достовірність** від **фізичної достовірності**?

Фізична достовірність моделі – це її відповідність законам природи. Якщо фізична модель явища помилкова, а рівняння такої моделі ми розв'язуємо також з помилками – то ми можемо все-таки отримати результати, що співпадають із вимірами. „*Погану теорію завжди можна узгодити з експериментом*", шуткував Альберт Ейнштейн.

А якщо рівняння правильної теорії розв'язувати помилковим математичним методом – результати можуть бути як „у городі бузина, а у Києві дядько". Таким чином,

якою б не була наша математична модель руху кульки у газовому середовищі, правильною чи помилковою, треба спочатку упевнитись, що запропоновані нею рівняння (4.12) ми розв'язуємо правильно. Тоді принаймні за **математичну достовірність** будемо спокійні!

Як же перевірити математичну правильність? Було б добре, якби наші рівняння (4.12) або (4.13) можна було б розв'язати аналітично[33] – тоді чисельний розв'язок за алгоритмом, що використано у програмі *Ball*, ми порівняли б з отриманими формулами. Однак розв'язати (4.13) аналітично, у вигляді формул, неможливо!

Лишається таке: рівняння (4.11) спростимо до таких, які вже можуть бути розв'язані у вигляді формули, і порівняємо два розв'язки – за алгоритмом і аналітичний. Такі спрощення можуть мати не лише математичні, а й фізичні значення, бо можуть відкритися певні зв'язки між фізичними змінними, певні фізичні закономірності.

Спрощення 1. Розглянемо випадок, коли рух тіла здійснюється у пустоті, у середовищі без опору. Рівняння такого руху можна отримати безпосередньо з (4.11), поклавши коефіцієнт опору рівним нулю, $c_2 = 0$. Тоді математична модель руху перетворюється у наступні рівняння (на масу m скорочено):

$$\frac{dx}{dt} = V_x, \qquad \frac{dz}{dt} = V_z,$$

$$\frac{dV_x}{dt} = 0, \qquad \frac{dV_z}{dt} = -g, \tag{4.16}$$

$$t = 0 \qquad x = 0, \qquad z = 0,$$

$$V_{x0} = V_0 \cos\alpha, \quad V_{z0} = V_0 \sin\alpha.$$

Ось і перше відкриття: рівняння і, таким чином, сам **рух тіла у пустоті не залежить від маси тіла та від його**

[33] Чи не пригадати розділ 1.7?

розміру! Не засмучуйтесь, що це вже було відкрито ***Галілео Галілеєм***, все одно чудово! Правильність такого висновку багатократно була перевірена у космічному просторі...

Ці рівняння (4.16) дуже просто розв'язуються аналітично. З них витікає, що горизонтальний і вертикальний рухи незалежні один від одного. Отримуємо, що горизонтальна швидкість увесь час стала, а горизонтальна координата – лінійно залежить від часу t,

$$V_x = V_0 \cos\alpha = const\ , \quad x = V_0 \cos\alpha \cdot t\ . \qquad (4.17)$$

Щодо вертикальної складової руху, то її швидкість виявляється лінійною, а координата – квадратичною функціями часу t,

$$V_z = V_0 \sin\alpha - g\,t\ , \quad z = V_0 \sin\alpha \cdot t - \tfrac{1}{2} g\,t^2\ . \qquad (4.18)$$

Формули, що отримано, запрограмовані як підпрограма *EmptyMotion* і вставлена у програму *Ball*. Аби її задіяти – уберіть значки коментарів %% (два значки % поспіль!) у рядочках 14 та 50 – 55. Вони програмують побудову „зірочок" – прогноз траєкторії кульки у випадку, що розглядається, коли навколишня атмосфера не створює їй опору. Як можна бачити на рис. 4.6, кульки летять точно за розрахунком програмою *Ball*. Для більш поглибленого співставлення, на графік також виводяться складові швидкості кульки за аналітичними формулами (4.17), (4.18). Можна упевнитись, що вони також співпадають з розрахунками за попередньою програмою у випадку $c_2 = 0$. Співпадіння двох розрахунків значно збільшує довіру до розробленої програми *Ball.m*.

Наша нова програмна доробка *EmptyMotion* насправді має значення не лише для тестування розрахунків. Вона є корисною також і для отримання певних фізичних оцінок. Знову „закоментуйте" 14-й рядочок програми *Ball.m*, тобто там знову стане

```
%%c2=0;
```

Оскільки рядочки 50 – 55 лишаються незакоментованими, то на графік буде виводитись знову результати двох

розрахунків: зеленим кольором для випадку руху у пустому просторі, і синім кольором – рух у реальному просторі, що створює опір кулькові. Розходження двох ліній, як на рис. 4.6.*б*, – це кількісна оцінка впливу атмосфери. **Хіба не цікаво?**

Спрощення 2. Розглянемо тепер більш складний випадок, коли атмосфера впливає на рух кульки, але – за лінійним законом (4.9А). Рівняння такої **лінійної моделі руху** аналогічні (4.10),

$$\frac{dx}{dt} = V_x, \qquad \frac{dz}{dt} = V_z,$$

$$m\frac{dV_x}{dt} = -c_1\rho_1(V_x - W), \tag{4.19}$$

$$m\frac{dV_z}{dt} = -mg - c_1\rho_1 V_z,$$

$$t = 0 \qquad x = 0, \qquad z = 0,$$

$$t = 0 \quad V_{x0} = V_0\cos\alpha,\ V_{z0} = V_0\sin\alpha. \tag{4.20}$$

але, як і попереднє спрощення, **дозволяють аналітичний розв'язок**. Розв'яжемо їх двома способами – чисельним, із застосуванням команди MATLAB *ode45*, і аналітичним способом з наступним програмуванням отриманих формул.

Чисельний розв'язок. Ці рівняння не можна отримати з (4.10) шляхом надання певного значення коефіцієнтові, і тому потрібно програмувати в окремій підпрограмі *m*-функцію їх правих частин. Назвемо її *Right1*, де номер *1* підкреслює відношення до лінійного закону опору (4.9А). Ця програма аналогічна підпрограмі *Right2*, її можна побачити передостанньою у головної програми *Ball.m*. Якщо „розкоментувати" рядочки 59, 66, 67 та 78, позначені трьома знаками проценту %%%, то відповідні програми починають працювати і на екрані бачимо одночасно синю траєкторію за

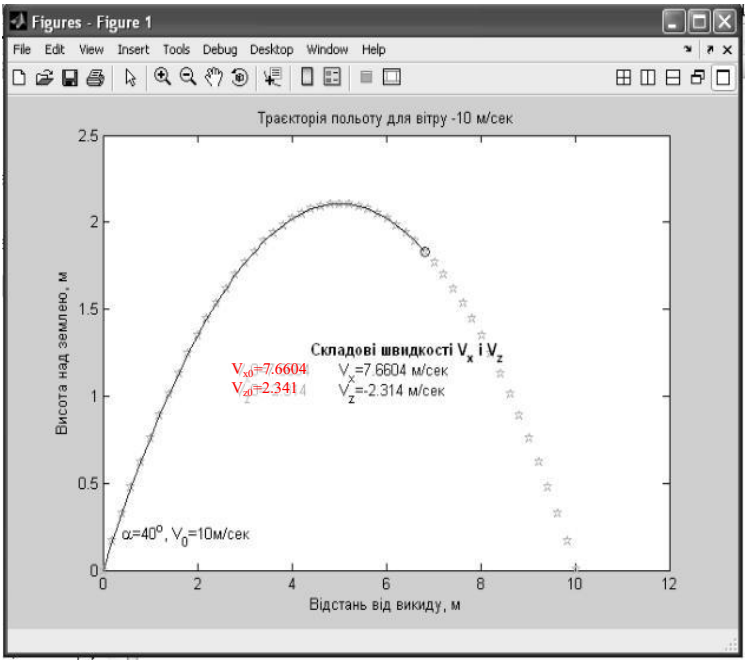


Рис. 4.7. Співставлення траєкторії кульки за аналітичною формулою („зірочки”) з розрахунком програмою *Ball* у випадку $c_2 = 0$ (лінія, атмосфера опору не створює).

квадратичною моделлю і малинову траєкторію за лінійною моделлю.

Увага: аби не „заблукати” між підпрограмами, радимо спочатку знову „закоментувати” рядочки 14 та 50 – 55, що відносяться до випадку середовища без опору.

Аналітичний розв’язок. У рівняннях (4.19) перше із третім і друге із четвертим утворюють дві непов’язані системи лінійних звичайних диференціальних рівнянь другого порядку. Це означає, що вертикальний і горизонтальний рухи не залежать один від одного. Обидві системи залежать лише від параметру $A_1 = \dfrac{c_1 \rho_1}{m}$ (індекс 1 нагадує про належність його до лінійної теорії). Кожна з систем легко розв’язується, даючи, з урахуванням

початкових умов (4.20), по парі наступних параметричних формул:

$$V_x = W + (V_0 \cos\alpha - W)e^{-A_1 t},$$

$$x = Wt + \tfrac{1}{A_1}(V_0 \cos\alpha - W)\left(1 - e^{-A_1 t}\right), \qquad (4.21)$$

$$V_z = -\tfrac{g}{A_1} + (\tfrac{g}{A_1} + V_0 \sin\alpha)e^{-A_1 t},$$

$$z = -\tfrac{g}{A_1}t + \tfrac{1}{A_1}(\tfrac{g}{A_1} + V_0 \sin\alpha)\left(1 - e^{-A_1 t}\right).$$

Ці формули запрограмовані в останній підпрограмі *LinearMotion* файлу *Ball.m*. Блоки у головній програмі *Ball*, що з нею пов'язані, також відокремлені трьома знаками коментарю %%%. Після „розкоментування" цих рядочків починають діяти елементи програми, відповідальні за розрахунок і графічну побудову результатів лінійної моделі за аналітичним і за чисельним алгоритмами. Після запуску з командного рядочку на зразок

>> r=6; *V0=10*; *Alfa=40*; *Wind=- 5*; *Ball*(*r*,*V0*,*Alfa*,*Wind*)

програма „прокладає" траєкторію лінійного руху кульки, розрахованої за аналітичними формулами (4.21), помічаючи їх значками ▽. Далі починають рухатись дві кульки, розраховані чисельно, – одна за лінійним законом (малинова траєкторія із значком ✸ на кінці), друга за квадратичним законом (синя лінія з ○ на кінці). Бачимо, що малинова траєкторія проходить точно за значками ▽, рис. 4.6.*в*. Таким чином, обидва розрахунки за лінійним законом підтверджують один одного. Співпадіння двох траєкторій – ще одне підтвердження **математичної** надійності нашого алгоритму і програми *Ball*.

Зробимо ще й висновки фізичного змісту:

1. Рух за лінійним законом „більш чутливий" до вітру W (це, однак, залежить від обраного значення сталої c_1).

2. На початку траєкторії обидва рухи співпадають, або близькі. Можна навіть підібрати значення c_1 так, аби траєкторії були близькими на більшій частині шляху кульки. Це дозволить сформулювати на підставі формул (4.21) загальні висновки про якісну поведінку кульки.

3. Щодо горизонтального руху кульки (перший рядочок формул (4.21)), то можна стверджувати, оскільки із збільшенням t член $e^{-A_1 t}$ швидко прагне до нуля, що $V_x \to W$ і $x \to Wt$. Тобто уся „залишкова" енергія кульки буде погашена тертям і вона буде рухатись разом із вітром, куди б він не ніс. За тієї ж причини другий рядочок формул (4.21) дає, що швидкість падіння кульки прагне до сталої величини, $V_z \to -\frac{g}{A_1}$. Останню називають „швидкістю парашутування", $\frac{dx}{dt} = V_x$; вона залежить від ваги кульки. (До речі, таку ж $V_{парашюта}$ можна знайти і у випадку квадратичного закону руху. Знайдіть її самі!).

Ну, хіба такі висновки не варті часу, витраченого на додаткові математичні розмірковування?

4.4.5. ФІЗИЧНА ДОСТОВІРНІСТЬ РЕЗУЛЬТАТІВ

Таким чином, математичних, розрахункових помилок ми не зробили. На чому ж базується **фізична достовірність** нашої математичної моделі і наших розрахунків? Чи можна очікувати, що розрахунки співпадуть з експериментальними вимірами, якщо буде зроблений експеримент?

Слід очікувати! Бо у її основу покладено фундаментальний закон Ньютона (4.7), багатократно і у різноманітних ситуаціях перевірений. Одна з складових моделі – закон тяжіння з постійним прискоренням *g*– теж перевірений ***Галілеєм*** і ***Ньютоном***, та у новий час – у космічних умовах. Друга складова – сила опору газового середовища (4.9Б) – має найбільшу частку „емпіризму". Вона

перевірена, однак, багатьма точними вимірами коефіцієнту g у лабораторних і натурних (на кораблях, літаках тощо) експериментах.

У названому коефіцієнтові c_2 ховаються, однак, можливі похибки наших розрахунків. Встановлено, що він залежить від форми кульки. Якщо вона сферична, то $c_2 \approx 0{,}40$, якщо циліндрична, то $c_2 \approx 1{,}0$. Крім того, ми не даремно написали знак $\approx$ (дорівнює наближено) в останніх формулах. Бо встановлено, що c_2 все-таки змінюється із швидкістю. Спеціальні експерименти із візуалізацією потоку навколо тіла показали, що поза тілом виникають ті чи інші вихорі, вони увесь час змінюють картину обтікання[*)]. Залежність $c_2 = (\mathrm{Re})$, де „***число Рейнольдса***" $\mathrm{Re} = \dfrac{2rV}{\nu}$ і ν – в'язкість газу, встановлено у вигляді експериментального графіку.

Однак, є ще обставини, які можуть суттєво вплинути на експериментальні результати. Це – плинність реального вітру W, який увесь час змінюється за силою і напрямком. Обговорення ще й цього питання може завести нас дуже далеко – у теорію турбулентності (найскладнішої теорії у сучасній фізиці!) і у теорію ймовірностей. У цій книжці ми маємо змогу торкнутися лише останньої.

4.5. ЯК НА КОМП'ЮТЕРІ У КОСТІ ГРАТИ, або ТЕОРІЯ ЙМОВІРНОСТЕЙ З MATLAB

„Бросая камни в воду,
наблюдай за кругами,
от них исходящими".
Козьма Прутков

Навіть складні і абстрактні математичні теорії виникли з (або ведуть до) практичних проблем, інтуїтивно очевидних ситуацій. Теорія ймовірностей не є виключенням:

[*)]Детальніше про це можна прочитати в Идельчик И.Е. Справочник по гидравлическим сопротивлениям. - М.: Машиностроение, 1975.

вона виникла з питань азартних ігор у кості та у карти. Це вже потім з'ясувалося, що саме такі ситуації виникають й у „більш поважних" дисциплінах. Однак кості (рис. 4.7) чи карти так і лишилися *моделями* теорії ймовірностей [54]...

Як нам подати застосування MATLAB до теорії ймовірностей, дуже складної науки, якщо ми орієнтуємось, в основному, на студентів-новачків?

Ми обрали наступний шлях. Пропонуємо читачеві декілька програм, *OrelReska* і *OrelReskaM* (Додаток 8), які моделюють ті чи інші випробування з ймовірнісним ісходом. Усі ці програми так чи інакше використовують генератор випадкових чисел *rand* (можете подивитись у Help), а самі генерують вже такі числа, нібито ви граєте у певну гру. Збережіть ці програми у робочій директорії, поки що не розбирайтеся з ними, та – гайда разом з нами „кидати каміння в океан випадковостей"! (А відповідь, як зроблені наші програми – прочитаєте пізніше)...

Рис. 4.7. Звичайна гральна кость – важлива модель випадкових подій....

Гра перша широко відома і називається Орел та Решка. Кидаємо монету та дивимось, якою стороною вона впала на стіл – Орлом (будемо вважати це за *0*) чи Решкою (вважаємо за *1*). Так от, запустіть у командному рядочку команду

>> X=OrelReska

– у відповідь обов'язково отримаєте *1* чи *0*. „Кидайте" *OrelReska* багато разів, і підрахуйте, яке з цих двох чисел випадає частіше. Можете навіть удвох позмагатися – один справжню монету кидає, інший за MATLAB. Хто раніше зробить всі випробування та підрахує результат? ☺...

З MATLAB можна „одним рухом” провести відразу кілька (навіть дуже багато!) „кидків”, випробувань. Для цього слід виконати попередню команду у такому форматі:

>> N=5; *X=OrelReska*(*N*)

Оскільки не поставили крапку з комою „**;**” наприкінці, то випадковий результат з нулів та одиничок (усього *N=5*) бачимо у командному рядочку. Можемо отримати таку відповідь у командному рядку

X = 1 1 0 0 1 (Іспит № 1)

а можемо і таку

X = 0 1 0 0 0 (Іспит № 2)

це – ***випадково***, на відміну від ***детермінованих*** попередніх задач!

Інтуіція підказує, що результати „*X =0”* або „*X =1”* мають *однакові шанси*, ще кажуть *рівні ймовірності*. Природно ввести таку міру ймовірності:

$$\text{Ймовірність випадку} = \frac{\text{Кількість випадків}}{\text{Загальна кількість випробувань}} \qquad (4.21)$$

Тобто ймовірність випадку „*X =0”* має бути ½=0**.**5, так само як ймовірність випадку „*X =1”*. А сума обох ймовірностей має завжди дорівнювати 1 – хоч ми поки що не знаємо Теорії Ймовірностей, та добре це розуміємо!

А тепер перевіримо випробування № 1 і № 2: у першому *1* зустрічається *з частотою* 3/5 і число *0* з *частотою* 2/5, а у другому – з частотами 1/5 і 4/5 відповідно (сума 1 завжди). Чому ж не ½? Відповідь відноситься до філософії цієї науки: теоретична ймовірність ½ має місце лише за умов великої кількості випробувань, навіть може бути – дуже великої. Тому й відрізняють терміни *ймовірність* і *частота* події [41,49]: другу величину можна виміряти, перша – ідеальна характеристика, що досягається лише за умов нескінченності. Перевіримо такий висновок.

Отримали *N* іспитів – знайдемо кількість *1* у масиві *X* і кількість *0* (частоти *p1* і *p0*) за такими формулами:

>> p1=sum(*X*)/*N*, *p0*= (*N-sum*(*X*))/*N* (4.22)

На комп'ютері ми можемо брати дуже великі *N*, і тому робимо такі обчислення (ваші результати не обов'язково саме такі):

```
>> N=500;  X=OrelReska(N); p1=sum(X)/N,  p0=(N-sum(X))/N
    p1 =  0.4780  %Не забудьте поставити  ; після OrelReska!
    p0 =    0.5220
>> N=500;  X=OrelReska(N); p1=sum(X)/N,  p0=(N-sum(X))/N
    p1 =    0.5040
    p0 =    0.4960
```

Бачите – **доволі далеко від теоретичної ½! Чому?**
Проводимо саме такі обчислення для ще більшого *N*:

```
>>N=500000; X=OrelReska(N); p1=sum(X)/N,p0=(N-sum(X))/N
%Не забудьте  ; після OrelReska!
```

З причини такої кількості обчислень *N=500000* (п'ятьсот тисяч!)[34[†]] ваш комп'ютер трохи затримається (а як той товариш, що кидає монети вручну?), і видасть на зразок

```
    p1 =    0.5007
    p0 =    0.4993
```

Ось і бачимо, що тим ближче до теоретичної ½, чим більше кількість випробувань! Таблицю реального фізичного експерименту з киданням монети читачі знайдуть у підручнику [49], стор. 31...

Додамо на закінчення, що наша програма *OrelReska* сама може обчислювати частоти появи *0* або *1* у виборці. Задля цього треба звернутися до неї із значенням 1 у другому аргументі:

```
>> N=5000000;  X=OrelReska(N,1);
```

Відповідь буде на зразок:

```
Отримано виборку X з кількістю випадкових 1 та 0
                          N =  .......
                          Частота 1 у виборці є  .......
                          Частота 0 у виборці є  .......
```

А спробуєте виконати програму із трьома аргументами? Ось яка розумна *OrelReska*! Про те, як вона створена, чому вона

розуміє різну кількість аргументів – у другій частині Підручника.

Гра друга. Нехай ваш партнер – MATLAB у даному випадку – „махлює" під час гри. Він „спотворив монетку" таким чином, що *1* стала випадати рідше або частіше; висловимося науково: з ймовірністю p, де $0<p<1$. Як його „викрити" під час гри?

Надаємо вам програму *OrelReskaM*, яка є аналогічною попередній, однак „махлює". Граємо з нею,

```
>> N=5000000; X=OrelReskaM(N,1);
   Отримано виборку X з кількістю випадкових 1 та 0
                        N =     5000000
   Частота 1 у виборці є   pN1 =    0.2998
   Частота 0 у виборці є   pN0 =    0.7002
```

І бачимо, ну немає тут „теоретичної" ½, а є 1/3! І дійсно, у рядочку № 15 програми є команда `p=0.3`, яка саме і „відповідає за махлювання"! Замініть її на `p=0.7`, і тоді *1* буде з'являтися з частотою 0.7, а *0* – з частотою 0.3. Продовжуйте гру! Подобається?

Тут ми запитаємо: А чи бачите ви, які нові цікаві „ігри" (цікаві перспективи, цікаві задачі!) тут виникають? Пропонуємо у них розібратися самостійно, граючись:

Задача 1. А як часто, з якою ймовірністю, у виборці з N елементів (нулів *0* або одиничок *1*) присутні лише одиниці? Візьміть спочатку N=2, у MATLAB зробіть кілька „експериментів" командою *OrelReska*(*2*) та підрахуйте частоту пари {*1*, *1*} серед результатів за формулою (4.22). Таке ж питання щодо частоти появи пари {*0*, *0*}.

Задача 2. А з якою частотою у такій виборці з N нулів або одиничок присутня *1* на першій позиції (і немає значення, що стоїть далі)?

Задача 3. Розгляньте ті самі питання, але ж щодо другої програми *OrelReskaM*, де 1 з'являється з ймовірністю $p \neq \frac{1}{2}$.

Гра третя. Візьмемо тепер складніший „генератор випадкових чисел" – *N*-гранник, де *N*=2 (та сама монета), *N*=3

(правильна піраміда), або *N*=4 (кубик), або ще складніші тіла з числа таких, що бачимо на рис. 4.8. Кожна грань (число на ній) може випасти з однаковою ймовірністю 1/*N*. Назвемо цю програму *KubiK* – пограйтеся з нею, Додаток 9!

Так само, як попередні програми, звертатися до неї можна за різним форматом, із різною кількістю аргументів. Коли аргументу немає зовсім, програма видасть результати одного „кидання" правильної піраміди (N=3 за умовчанням) – ціле число від 1 до N=3. Хочемо одноразово „кидати" кубик (шість граней) – звертаємось до команди з одним аргументом *X*=*KubiK*(*6*) і для *X* отримуємо одне з можливих чисел 1, 2, 3, 4, 5 або N=6. Хочемо отримати вибірку з *M* чисел {1, 2, 3, **. . .**, N}, де *M* і N будь-які цілі, виконуємо ту ж саму команду з двома вхідними аргументами:

>> N=5; *M=10*; *T=KubiK*(*N*,*M*)

У такому разі вибірка *T* буде складатись з десяти чисел, серед яких присутні лише 1, 2, 3, 4 і 5, як, наприклад

T = *3 2 4 4 1 3 3 4 4 4* (4.23)

У такій виборці одиничка випала 1 раз (і тому, її частота дорівнює 1/10=0,10), двійка – теж один раз (частота, так само, 0,10), трійка 3 рази (і тому, її частота дорівнює 3/10=0,30), четвірка п′ять разів (її частота випадіння сталася 5/10=0,50), а п'ятірка не з'явилася жодного разу (її частота 0) – обчислення проведено за формулою (4.21). Результати такого „чисельного експерименту" можна представити такою таблицею

Випадкове число	**Частота в експерименті**	**Кумулятивна частота**
1	0,1	**0,1**
2	0,1	**0,2**
3	0,3	**0,5**
4	0,5	**1,0**

Зрозуміло, що сума усіх частот – обов'язково дорівнює 1 (*одиниці*)!

Результати „експериментів", природно, випадкові, і так само випадково виглядає графічне їх представлення. Так,

Рис. 4.8. Кубики з трьома, вісьма, дванадцятьма і т.д. гранями, які можуть „випасти” з однаковою ймовірністю – зручні „генератори” випадкових чисел.

Рисунки з Вікіпедії — вільної енциклопедії.

результати одного з них, що також складався з десяти чисел, представляє ламана 1 на рис. 4.9.

А тепер – будемо „набирати статистику”, розглядати вибірки з більшою кількістю результатів експериментів, які для зразку представлено на рис. 4.9 іншими ламаними. Зрозуміло, вручну рахувати кількість усіх можливих чисел у виборці – справа нудна, доручимо це програмі. За умов, що третій параметр команди дорівнює одиниці *KubiK*(*N,M,1*), вона видасть таблицю, як показано. А покладемо третій вхідний параметр двійці *KubiK*(*N,M,2*), програма ще й графік додасть!

Збільшуємо кількість експериментів (другий параметр команди *M*), і бачимо, що частота появи чисел наближується до теоретичної величини 1/*N*! У даному випадку теоретичне значення буде 1/*N*=1/5=0,20. Саме про це свідчить рис. 4.9. Залежність частоти як функції усіх можливих значень випадкової величини *X*, а також її графічне представлення, як на рис. 4.9, називається **частотною** (або **диференціальною**) **функцією**. Її позначають маленькою літерою *p*.

Можна і іншим чином характеризувати результати комп’ютерного експерименту (4.23). Цю **міру ймовірності** позначаємо великою літерою *P* Яка у нашому експерименті ймовірність появи чисел, що **менше за *1***? Відповідь очевидна: такі числа не можуть з’явитися, ймовірність *P*(*0*) =0. А яка у числовому експерименті (4.23) частота появи чисел, що менше або дорівнюють *1*? Відповідь: одно з десяти, тобто *P*(*1*) =0,1. Яка ж в експерименті частота появи чисел, що менше або дорівнюють *2*? Відповідь: враховуємо усі одинички (1 раз) і двійки (1 раз), тобто два з десяти, *P*(*2*) =0,1+0,1=0,2. Яка була частота появи чисел, що менше або

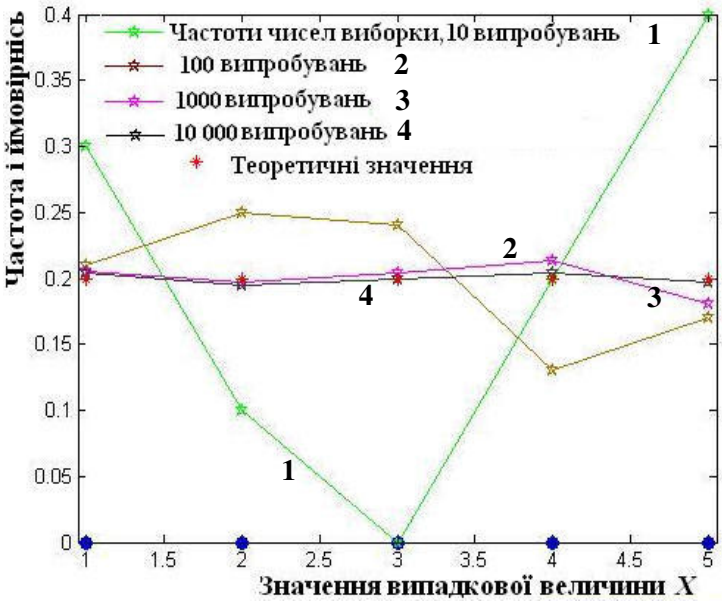


Рис. 4.9. Експериментальні частоти чисел, що випадають у разі „кидання кубика" *KubiK(N,M)* з N=5 гранями з різною кількістю випробувань *M*.

дорівнюють *3*? Відповідь: усі випадки попереднього питання, плюс три рази 3, і тому п'ять з десяти, $P(3)$ =0,2+0,3=0,5.

Аналогічно: Частота появи чисел, що менше або дорівнюють *4*? Частота тих, що менше 3 плюс четвірка з'явилася п'ять разів, тобто $P(4)$ =0,5+0,5=1. П'ятірка у тому експерименті (4.23) не випала жодного разу, тобто відповідь $P(5)$ =1. А якою може бути частота появи чисел, що менше або дорівнюють *6*? Зрозуміло, *M* разів з *M* можливих, бо усі числа в експериментах з N=5 менше за шість, тобто $P(n)$ =1 для усіх n >5.

Функція цілого аргументу $P(n)$ є не спадною, що змінюється від 0 до 1. Вона „накопичує" всі частоти, менші за $p(n)$, і тому називається **інтегральною функцією** випадкової величини. Графік інтегральної функції за експериментом (4.23) подано синьою ламаною лінією на рис. 4.10, а червоні зірочки – теоретичні значення інтегральної

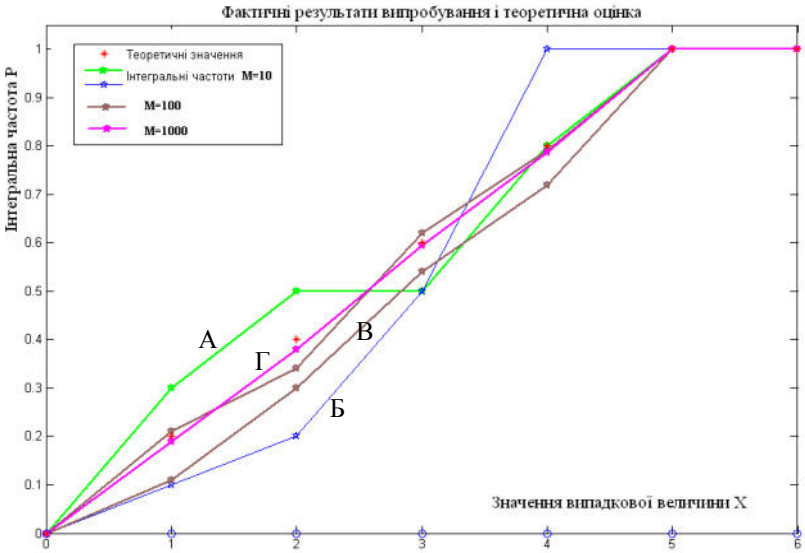


Рис. 4.10. Інтегральні частоти випадкових чисел у разі „кидання кубіка” *KubiK*(*N*,*M*) з *N*=5 гранями з різною кількістю випробувань *M*: ламані А і Б – *M=10* випробувань; обидві В – *M=100* і ламана Г – *M=1000*.

функції, що виходять з припущення про однакову ймовірність кожного з чисел від *1* до = *N*=5. Зелена лінія – ще один „експеримент” з кількістю випадкових чисел *M=10* – синя і зелена лінії демонструють значне „хитання” відносно теоретичних точок.

А тепер проведемо експеримент з відразу *M=100* випробуваннями:

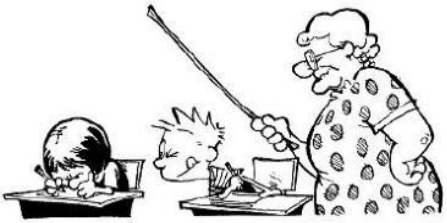
>> N=5; *M=100*; *KubiK*(*N*, *M*, *3*);

При значенні третього параметра „*3*” програма *KubiK*(*N*,*M*,*3*) видає додатково графік інтегральної функції. Результати демонструють дві коричневі криві. Зверніть увагу, що вони вже менше „коливаються” відносно теоретичних точок! А рожева ламана відображає вже „статистику” з *M=1000* випробувань, і вона ще ближче до теоретичних точок!

Таким чином випадкові явища все ж-таки демонструють у своїй поведінці „статистичні” закономірності. І ці закономірності можна характеризувати або диференціальними розподілами ймовірності, або інтегральними.

Як зроблено ці програми – обговоримо у наступній частині Підручника...

ЛАБОРАТОРНІ РОБОТИ ДО МОДУЛІВ 1 та 2



Мета цього розділу полягає у тому, аби студенти на практиці засвоїли і закріпили все те, що вивчають у теоретичній частині посібника. Кожна з робіт є лише приблизний план – що саме зробити, на що звернути увагу. Студентам пропонується провести певне дослідження та експерименти з MATLAB.

Викладач очікує від вас певний Звіт у довільному форматі: що саме ви зробили, що цікавого знайшли та – чому ні? – чи є якісь відкриття ☺? Бажано не лише виконувати та описувати для викладача, та ще робити висновки: заради чого це? Що саме ви узнали? Та обов'язково Висновок наприкінці: чи досягнута мета роботи?

Модуль 1
„Ручна" алгоритмізація типових математичних задач

Лабораторна робота № 1

Дослідження інтерфейсу MATLAB, робота з числами та масивами, графіки 1*d*-функцій

Метою є накопичити перші навички роботи з MATLAB, оволодіти побудовою графіків функцій та розв'язати перші задачі, які у подальшому будемо програмувати.

1. Запустіть MATLAB: як виглядає його інтерфейс[35]? Чи можна якось змінювати його конфігурацію? Які "вікна" (windows) він може мати? Для чого використовують *Command Window* та *Command History Window*? Які ще є "вікна", хоча й наразі вам не потрібні? Як зконфігурувати найбільш зручне середовище?

[35] До речі, що так „***інтерфейс***", ваша думка?

2. Що таке Help: як знайти допомогу з того чи іншого питання? Для прикладу, отримайте допомогу щодо команд *fplot* та for *ezplot.*
3. Поясніть оператори MATLAB +, -, *, /, ^, *sqrt* (спробуйте команду *help arith*). Перевірте роботу „внутрішніх" функцій MATLAB *sin*, *cos, tan*, *exp, log, log10*. Наприклад, спробуйте обчислити $\frac{\sqrt{x+\sqrt{y}}}{\sqrt{x-\sqrt{y}}}+\frac{\sqrt{x-\sqrt{y}}}{\sqrt{x+\sqrt{y}}}$ для $x = 2\,\boldsymbol{i}$ та and $y = 3$ (зверніть увагу: MATLAB розуміє комплексні числа з ***i***). А щодо інших значень x та y? Яким чином MATLAB „скаржиться" на помилки з вашого боку? Спробуйте й інші обчислення з дійсними та комплексними числами, наприклад (*a*) *sin*(47°), *cos*(80°)? (*b*) *sin*(***i***), *cos*(***i***), *tg*(***i***), *exp*(***i***), *log*(***-1***)? Вам казали у школі, що $\sqrt{-1}$ або $ln(-1)$ не існують – спробуйте у MATLAB! Зокрема, чому дорівнює ***i*i*** (тобто „уявна одиниця" у квадраті, $\boldsymbol{i}^2$)? Почитайте у Вікіпедії та поясніть: що таке "*комплексне число*"?
4. Обчисліть функції $F_1 = \frac{1}{2}(e^x - e^{-x})$, $F_2 = ln(x+\sqrt{x^2+1})$ для кількох значень x, побудуйте їх графіки та порівняйте з графіками функцій відповідно *sinh*( ) та *asinh*( ). Ваш висновок? Як щодо функцій *cosh*( ) та оберненої до неї? (Радимо ознайомитися у Вікіпедії, що є „Число e" та „гіперболічні функції")
5. Які *імена змінних* (*ідентифікатори*) дозволені в MATLAB? Які не дозволені? Поясніть роль ідентифікаторів *ans*; *pi*, *i*, *j*, *inf*, *NaN* ? Поясніть призначення операторів **=** та **;** (крапка з комою)**.**
6. Як ввести у пам'ять MATLAB матрицю вимірності *m* x *n*? Приклади: 1) $A = \begin{pmatrix} 1 & 2 & 3 \\ 2{,}1 & 3{,}3 & 4{,}25 \\ 5{,}7 & 0{,}03 & -1{,}7 \end{pmatrix}$; 2) матриця-рядочок

$row = (2{,}1 \quad 3{,}3 \quad 4{,}25)$; 3) матриця-стовпчик $col = \begin{pmatrix} 2 \\ 3{,}3 \\ 3e-2 \end{pmatrix}$.

Поясніть оператори для матриць (+, -, *, .* , /, ./ , ^, .^). Чому MATLAB може „скаржитись”?

7. Уявіть собі масив (матрицю) *M* з великою кількістю стовпчиків. Як отримати масив M_1, що включає лише кожний 3-ій стовпчик з *M*? Наведіть власний приклад: як з даної матриці *A* отримати якусь під-матрицю?. Як транспонувати задану матрицю? Як отримати довжину (кількість елементів) вектора, вимірність матриці, яку приготував хтось інший?
8. Як отримати „впорядкований вектор на зразок $v = \{3,4,\ldots,101\}$ або $x = \{-0.5,\ 0,\ 0.5,\ 1.0,\ 1.5,\ 2.0,\ldots,\ 4.5, 5.0\}$? Які висновки можна зробити щодо оператора **:** (дві крапки, colon)?
9. Дослідіть „точність” обчислень у MATLAB в залежності від команд *format short* та *format long* у командному вікні (Command Window), спробуйте інші формати через меню ***File\Preferences…***
10. Ознайомтеся з Help щодо команд побудови графіків в MATLAB; які саме команди існують для цього? Побудуйте найпростіші графіки функцій на зразок $y = a\sin(k_1 x) + b\cos(k_2 x)$, $y = ax^{\alpha} + b$ тощо; значення коефіцієнтів a, b, k_i, α оберіть самі.
11. Побудуйте кілька графіків <u>в одному вікні</u>; побудуйте ще кілька графіків <u>в іншому вікні</u>. Поясніть дію і призначення команд *figure*(1), *figure*(3), *hold on* та *hold off*. Дослідіть можливості команд ***plot***, ***fplot***, ***ezplot*** та ***comet***. Побудуйте графіки *параметрично заданих* функцій

(***1***) *ezplot*('t*cos(t)','t*sin(t)', [0, 10*pi], **2**);

(***2***) *ezplot*('sin(3*t)*cos(t)','sin(3*t)*sin(t)', [0,pi], 4) [36]

[36] Останній аргумент у цих командах не працює у MATLAB версій нижче за 7; опустіть його у такому випадку.

Те саме прохання зробити командою *plot*. У чому різниця в їх використанні? В цих задачах „поекспериментуйте” з параметрами, візьміть (*1*) 10**pi*, (*2*) 5**t* тощо.

12. *a*) Побудуйте графіки функцій $y = \frac{sin(kx)}{x}$ для різних k в одному вікні. Що ви знаєте про їх поведінку в точці $x = 0$ з курсу вищої математики (так звана ***перша чудова границя***)?
    b) Підтвердить за допомогою графіку, що числова послідовність $f_1 = 2$, $f_2 = \left(\frac{3}{2}\right)^2$ і т.д. за формулою $f_n = \left(1 + \frac{1}{n}\right)^n$ за умови $n \to \infty$ наближується до певної величини („збігається”) та знайдіть цю величину якнайточніше (так звана ***друга чудова границя***). Зверніть увагу, що у цьому випадку n дискретний та додатній аргумент!
13. Як можуть команди MATLAB *grid, xlabel*, *ylabel, title, axis, legend* "покрасити" ваші графіки?. Як застосувати різні стилі кривих, кольори, позначки коли будуєте графіки? Як можна ваші побудовані в MATLAB графіки та обчислення розмістити у Word документі?
14. Дослідіть графічне вікно "Figure": які воно має меню та іконки, як можна викликати "Property Editor" (Редактор властивостей) та змінити атрибути ваших графіків?

15.1.[37] Шляхом побудови графіків підтвердити або спростувати, що частині суми ряду Тейлору[*)]:

$$S_n(x) = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots + \frac{x^n}{n!}$$

наближується до функції $y = e^x$.

[37]Номер вашого варіанту *VarN*, чи то 15.1, або 15.2, 15.3 або 15.4 можна визначити MATLAB-обчисленням *VarN=mod*(*N_list*,4)+1, де *N_list* є вашим номером у списку групи.

(Ознайомітеся із довідкою http://en.wikipedia.org/wiki/Taylor_series, http://uk.wikipedia.org/wiki/ стаття „Ряд_Тейлора”).

15.2. Підтвердити або спростувати, що ряд Тейлора *):

$$S_n(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots + (-1)^n \frac{x^{2n+1}}{(2n+1)!} = \sum_{k=0}^{n} (-1)^k \frac{x^{2k+1}}{(2k+1)!}$$

наближується саме до функції $y = \sin x$.

15.3. Підтвердити або спростувати, що ряд Тейлора *):

$$S_n(x) = x + \frac{x^3}{3!} + \frac{x^5}{5!} + \ldots + \frac{x^{2n+1}}{(2n+1)!} = \sum_{k=0}^{n} \frac{x^{2k+1}}{(2k+1)!}$$

збігається саме до $y = \mathrm{sh}\, x$ (див *No*. 4.).

15.4. Підтвердити або спростувати шляхом побудови графіків, що частинні суми ряду Фур’є *)

$$\Phi(x) = \frac{4}{\pi}\left(\frac{\sin(x)}{1} + \frac{\sin(3x)}{3} + \frac{\sin(5x)}{5} + \frac{\sin(7x)}{7} + \ldots + \frac{\sin((2n-1)x)}{2n-1}\right),$$

$n \to \infty$ збігається до функції $f(x) = \begin{cases} -1, & x \in (-\pi, 0) \\ +1, & x \in [0, \pi] \end{cases}$.

А графік останньої функції $f(x)$ побудувати можете? (Побудуйте „руками”; комп’ютерну програму створимо у роботі 5). А на усій число осі $(-\infty, +\infty)$ з урахуванням того, що $f(x)$ має період $T = 2\pi$?

(Пояснення можна знайти в http://uk.wikipedia.org/wiki/ стаття „Ряд_Фурє”)

16. Дослідити текстові масиви на зразок $\begin{pmatrix} Nina & Doctor \\ Sasha & Student \end{pmatrix}$.

Строго виконуйте правило, аби в кожному стовпчику кількість знаків була однакова! Запропонуйте власні приклади. Чи потрібні такі масиви у програмуванні?

Лабораторна робота № 2

Основні математичні задачі з MATLAB.

Мета: продемонструвати можливості MATLAB *у розв'язуванні основних математичних задач вашого курсу математики, зокрема* (1) *лінійної алгебри та систем лінійних алгебраїчних рівнянь* (*СЛАЕ*), (2) *розв'язування поліноміальних та трансцендентних рівнянь однієї невідомої.*

Порада: *Використати всі методи, які вчили, тобто метод Крамера, оберненої матриці та, для просунутих студентів, метод елімінації невідомих Гауса тощо. Враховуючи, що всі операції є однотипними, намагайтеся використовувати* Matrix Laboratory *якнайширше. У наступних лабораторних роботах вам буде запропоновано* ***алгоритмізувати*** *ці дії.*

Друга мета*роботи полягає у тому, аби ви почали використовувати* MATLAB *в інших дисциплінах, що вивчаються, і перш за все у математиці.*

1. Які методи розв'язування систем лінійних алгебраїчних рівнянь (СЛАР) ви знаєте?
2. Як можна обчислити в MATLAB визначник матриці, обернену матрицю? Умова на аргумент?
3. Розв'яжіть наступну СЛАР[38]:

$$\begin{aligned} 4x_1 + 2{,}4x_2 - 0{,}8x_3 + x_4 &= 9, \\ x_1 + 3x_2 - 1{,}5x_3 - x_4 &= 8, \\ 0{,}4x_1 - 0{,}8x_2 + 4x_3 + 2x_4 &= 20, \\ 2x_1 - x_2 - 2x_3 - 3x_4 &= 7. \end{aligned}$$

Порада: підготуйте спочатку розширену матрицю СЛАР та вже з неї отримуйте всі інші матриці, що потребує хід розв'язку.

Як можна перевірити розв'язок?

4. Оберіть СЛАР з Додатку 1 у відповідності до вашого номеру у списку групи[39] та розв'яжіть методом, який вам подобається більше[40].

---

[38] Студенти з парними номерами у списку групи використовують метод Крамера, з непарними – матричний метод.

[39] Ось MATLAB-алгоритм отримання номеру варіанту за номером студента:

5. Розв'яжіть трансцендентне рівняння $\cos x = ax$. Яка кількість в нього коренів? (Використати значення коефіцієнту a ваші власті, більше та менше 1, наприклад $a = 3$ та $a = 1/3$). Спробуйте MATLAB-команди ***fzero***, ***solve*** та сформулюйте ваші думки або запитання з їхнього приводу. Використайте також графічний метод розв'язку, особливо якщо є щось незрозуміле з тими командами.
6. Оберіть рівняння з одним невідомим з Додатку 2 у відповідності до вашого номеру у списку групи, розв'яжіть його кількома методами та дайте їх порівняння.
7. Чи можна також назвати рівняння $P_5(x) = x^5 - 2x^4 + x^3 - 2x^2 + x + 2$ трансцендентним? Чи якось інакше? Побудуйте графік функції $y = P_5(x)$. Чи можете надати формулу розв'язку цього степеневого рівняння через його коефіцієнти? Кубічного рівняння $x^3 + ax^2 + bx + c = 0$? Квадратного рівняння $x^2 + ax + b = 0$?

Деякі поради можна знайти в [1-3].

VariantNumber= *mod*(*YourNumber*, *N*)+*1*,
де *N=12* загальна кількість варіантів у Додатку 1.

[40] Прохання використати метод, відмінний від методу у попередньому завданні.

Лабораторна робота № 3

Головні математичні задачі – з MATLAB.
Символьні обчислення.

Мета: *продемонструвати можливості MATLAB у розв'язуванні головних математичних задач університетського курсу математики; підкреслити зокрема фундаментальну різницю між аналітичними та чисельними обчисленнями в MATLAB.*

1. Прохання зобразити на екрані правильний *N*-кутник, де *N* є будь-яку ціле число більше трьох, наприклад номер дня тижня чи місяцю. Спробуйте також зробити його повернеким на певний кут, наприклад $\varphi_0 = 10^o$ або $\varphi_0 = 40^o$. Зробіть його контур того чи іншого кольору.
2. Для просунутих студентів! Зобразіть на екрані *N*-конечну зірку.
3. Символічні обчислення вищої математики:

   3.1. Знайти похідну функції $y = \left(\frac{x}{1+x} \right)^x$, обчислити її у точках $x = 1$, $x = 2$, $x = 3$; побудувати графіки фнкції та її похідної та порівняти їх. Запропонуйте та дослідіть власний приклад! Використовуйте команду ***pretty*** тут та в інших завданнях. Важливо навчитися використовувати як ***ezplot***, так і ***plot***, та приймати до уваги область визначення функцій.

   3.2. Знайдіть інтеграл функції $y = \left(\frac{1}{1+\sqrt{x+1}} \right)$, побудуйте графік нової функції та порівняйте його з графіком вихідної. Запропонуйте та дослідіть власний приклад! Знову використовуйте ***ezplot*** та ***plot*** та враховуйте область визначення.

   3.3. Знайдіть границю $\lim\limits_{n\to\infty} n^{\frac{3}{2}} \left(\sqrt{n^3+3} - \sqrt{n^3-2} \right)$ двома способами: методами символьних обчислень MATLAB та побудовою графіку, що розкриє цю вимогу „ $n \to \infty$ ”.

4. Сума частинної суми числового ряду $S_n = 1 + \frac{1}{2^k} + \frac{1}{3^k} + \ldots + \frac{1}{n^k}$ залежить як від кількості взятих у суму доданків n, так і від показника степеню k. Взявши k яке хочете ($k = 2$ або вашому номеру у списку групи), спробуйте кілька доданків у сумі. Використовуйте MATLAB-команду *symsum*. Що відбувається яз сумою S_n якщо n, кількість доданків, прагне до нескінченності? **Увага:** чи завжди відповідь MATLAB зрозуміла?

5. У роботі №1 досліджували дивні властивості частинних сум ряду Тейлора. Тепер використайте MATLAB - команду *taylor*, аби отримати ряди Тейлора для функцій за вашим бажанням, таких як e^x, $sin\,x$, $cos\,x$, ln(x+1) тощо. Перевірте, чи справді поліноми, що отримуєте, наближуються до відповідної функції якщо кількість членів у сумі нескінченно зростає?.

Коментар: в MATALAB є графічна програма *taylortool*; цей факт ще раз підтверджує, яким є важливим цей математичний матеріал!

6. Чи вміє MATLAB думати? Перевірте на його вмінні доводити тотожності:

Чи вміє MATLAB „довести", що для будь-яких x та y

(i) $sin^2 x + cos^2 x = 1$? (ii) $ln\,x + 5\,ln\,y = ln(xy^5)$ (x та y мають бути також додатніми)?

Дослідіть для цього команди *simple* та *simplify*! Запросіть help, якщо буде потрібно. Пригадайте ще кілька шкільних тотожностей з математики та перевірте їх у MATLAB? Так який висновок: чи можна доводити теореми з використанням комп'ютеру?

7. Спробуйте охарактеризувати різницю між чисельними та символьними об'єктами у MATLAB, та чи може останні бути для вас корисними.

Лабораторна робота № 4

Символьні об’єкти MATLAB. Символьні та чисельні обчислення.

Мета: *поглибити розуміння символьних об’єктів* MATLAB, різницю між ***numeric*** *та* ***symbolic*** *обчисленнями.*

1. Яким ***типом даних*** є **3.14926** та ***sin***(***pi***/**2**)? Які ще типи даних ви знаєте в інформаційних технологіях? Що є числовими даними (***int***, ***double***), що є ***string data type*** (або текстовий тип даних)? Що є **symbolic data type**? Чи можна конвертувати один тип даних в інший? Чи можете своїми словами пояснити, що саме пов’язане з поняттям „тип даних”? Чи можете пояснити різницю між **символьними обчисленнями** та **чисельними**? Які операції пов’язані з тими чи іншими?
2. Наведіть приклади операцій над даними **текстового** (або ***string***) типу даних. Чим він відрізняється від **symbolic data**? У чому саме різниця між командами ***ezplot***, ***comet***, ***fplot*** та ***plot***?
3. Візьміть якусь функцію $f(x)$ (Наприклад, $sin(x)$) та отримайте її числові значення для аргументу x на якомусь числовому інтервалі. Для двох числових векторів, що отримано
   3.1. уявіть собі, що f_i є дистанція, що об’єкт проходить за час x_i. Яку приблизно швидкість має об'єкт у моменти часу x_1, x_2, x_3, … x_N? Побудуйте графік зміни швидкості з часом.
   3.2. функція $f(x)$ відома, таким чином, таблицею своїх значень f_i при аргументі x_i. Ця крива на відрізках $[x_1, x_2]$, $[x_2, x_3]$, $[x_3, x_4]$ обмежує якусь площу S_{12}, S_{23}, S_{34} , S_{45} і т..д., тобто маємо функцію S від x. Побудувати її графік.

   Порада: замість площі під „криволінійною трапецією” $f(x)$ на кожному з інтервалів $[x_i, x_{i+1}]$ брати її наближене значення як площі прямокутника. Без сумніву, ви відчули якийсь зв’язок з шкільними визначеннями похідної $f'(x)$ та інтегралом $\int f(x)dx$. Мета завдання як раз у тому, аби цей зв’язок прояснити. Важливо з’ясувати, як результати будуть залежати від „*кроку*”

$\Delta x = x_i - x_{i-1}$. У наступній роботі такі обчислення запрограмуємо!

4. У MATLAB многочлени $P_n(x) = a_1 x^n + a_2 x^{n-1} + a_3 x^{n-2} + ... + a_n x + a_{n+1}$ можуть бути представлені як числові об'єкти, так і як символьні. Тут їх розглянемо як ***numeric objects***:
   Як такий многочлен ввести у пам'ять MATLAB як об'єкт чисельного типу? Чи це можливо для символьних коефіцієнтів $\{a_1, a_2, a_3, .. a_{n-1}, a_n\}$, або для числових їх значень? Запропонуйте приклади многочленів (поліномів), обчисліть їх у якійсь точці $x = x_0$, тобто $P_n(x_0)$ за умови, що x_0 число. Представивши поліном з числовими коефіцієнтами як числовий об'єкт, знайдіть його корені, перевірте їх! Скільки коренів у многочлена? Скільки серед них комплексних та дійсних? Чи можуть MATLAB-команди ***polyval*** та ***roots*** бути корисними для поставлених задач? Чи можна використати команди ***ezplot*** та ***plot*** для побудови графіків поліномів як функції від x?
5. Уявіть собі тепер обернену задачу: задано вектор чисел $\{x_1, x_2, ..., x_k\}$, знайти многочлен з такими кореням. Команда ***poly*** вирішує таку задачу. Як? Нехай дано два многочлена, ще й $Q_r(x) = b_1 x^r + b_2 x^{r-1} .... + b_r$. Знайдіть добуток двох многочленів $R(x) = P_n(x) Q_r(x)$, який, зрозуміло, також є многочленом. Спробуйте команду ***conv***(P,Q). Що можете сказати про корені многочлена $R(x)$?

6. Поліноми як ***symbolic objects*** MATLAB:
   Конвертуйте числовий многочлен, що використовували вище, у такий саме поліном, але у символьному вигляді за допомогою команди ***poly2sym***. Погляньте на нього командою ***pretty***. Можете побудувати його графік за допомогою ***ezplot*** та ***plot***? Покажіть!

Як представити поліном $P_n(x) = a_1 x^n + a_2 x^{n-1} + a_3 x^{n-2} + \ldots + a_{n-1} x + a_n$ у пам'яті MATLAB у якості символьного об'єкту? Як використати команди ***sym*** та ***syms***? Як помножити один символьний поліном на інший, „розкрити дужки", „привести подібні"? Команди ***expand*** та ***collect***. Як помножити два полінома у числовій формі?

Знайти корені поліному, наприклад, у випадку $n = 3$, $a_3 = 5$, $a_2 = 0$, $a_1 = 3$, $a_0 = 1$. Наведіть власний приклад.

7. Розберіться з командою ***subs***. Як її використати для підстановки числових значень в символьний вираз? Як використати команду ***sym2poly*** у випадку поліномів? Пригадайте приклад из попередньої роботи.

8. Ще раз зробіть висновок, яка різниця між ***numeric*** і ***symbolic*** objects в MATLAB, коли використовувати ті чи інші. Та ще раз підсумуйте, які знаєте Data Types в програмуванні?

9. Нарешті, чи досягнута мета цієї лабораторної роботи? Як широко ви вже застосовуєте MATLAB в інших дисциплінах, що вивчаєте?

Чи готові до модульної контрольної?

Модуль 2

Основи програмування в MATLAB

Лабораторна робота № 5

Починаємо програмувати з MATLAB.

Мета: *розглянути оператори* ***if … end***, ***for … end*** *та застосувати їх у простих, але цікавих програмах.*

1. Поясніть використання операторів ***if***, ***else***, ***elseif***; як „поводить себе" редактор m-файлів, коли „зустрічає" ці слова? Застосуйте їх у розробці таких от функцій (програм) MATLAB, обраних відповідно до вашого номеру у списку групи[41]. Функції мають бути 2π-періодичними на усій числовій осі $-\infty < x < \infty$.:

   1.1. $f(t) = x^2, \quad x \in (-\pi, \pi)$;

   1.2. $f(t) = \begin{cases} -x - \dfrac{\pi}{2}, & x \in (-\pi, 0) \\ +x - \dfrac{\pi}{2}, & x \in (0, \pi) \end{cases}$;

   1.3. $f(x) = \begin{cases} -1, & -\pi < x < 0; \\ \dfrac{x}{2}, & 0 < x < \pi. \end{cases}$

(Порада: йти до потрібної програми крок за кроком, почавши з найпростішої задачі та поступово ускладнюючи (*1*) спочатку зробити програму для одиночного аргументу не звертаючи увагу на вимогу періодичності, (*2*) модифіковану програму, що вже „розуміє" векторний аргумент (два методи запропоновані у книжці), та ще не враховує періодичності, і нарешті (*3*) остаточну програму, що враховує періодичність. На кожному етапі перевіряйте можливість використання команд *ezplot*, *fplot*, *plot*, як це є для функції, наприклад, $y = \sin x$ Остаточна програма має бути повністю сумісна з вимогами MATLAB!).

---

[41] За алгоритмом *VarN*=mod(*N_list*, 3)+1, де *N_list* є вашим номером у списку.

2. Поясніть роботу операторів ***for … end***, ***while... switch… end***. Застосуйте їх для програм, обраних у відповідності до вашого номеру у списку групи (студенти, які відчувають себе сильними у науці програмування, можуть зробити додатково програму за варіантом 2.А). Спробуйте зробити й варіант програми у діалоговій формі:

2.1. Правильний трикутник того чи іншого кольору, що обертається на екрані РС за годинниковою стрілкою або проти неї.

2.2. Правильний квадрат того чи іншого кольору, що обертається на екрані РС за годинниковою стрілкою або проти неї.

2.3. Зробіть круг на екрані РС, що пульсує: росте-росте, потім зменшується за розміром, зменшується, і т.д. періодично. Непогано, якщо він ще й колір змінює…

2.А. Зробіть правильний *N*-кутник або 3-пелюсткову розу (завдання 11 лабораторної роботи №1) того чи іншого кольору, що обертається на екрані у тому чи іншому напрямку. Чи можна керувати ще й швидкістю обертання?

На відміну від попередніх робіт, викладачеві треба здавати не лише Звіт, де вміщено ваші програми та результати їх випробування, та ще носій (CD-R, **що перезаписується** або **флешка**) з програмами (*m*-файлами).
Також важливо доповнювати програми Help-овою частиною та коментарями, що пояснюють найважливіші дії програми.

Лабораторна робота № 6

<u>Станемо програмістами 1-го рівня!</u>

Мета: *оволодіти подальшими інструментами і можливостями програмування, використати знання та досвід для створення вже доволі складних програм у MATLAB-середовищі.*

1. Пригадайте завдання 3 у лабораторній роботі №4, та зробіть програму, відповідно названую ***MyNumDiff.m*** або ***MyNumInt.m***, що автоматично розв'язують ті завдання, а саме:

1.1. ***MyNumDiff*** отримує на вході будь-яку функцію *f*(*x*), запропоновану користувачем, інтервал [*a,b*] та ціле число *N*, число дискретизації інтервалу, та повертає графіки з її похідними, отриманими як чисельним методом на інтервалі $a \le x \le b$, так і аналітичним диференціюванням $f'(x)$. Дослідіть, чи зближується результати двох методів, якщо *N* збільшується ($N \to \infty$), correspondingly, відповідно, $\frac{b-a}{N} \to 0$?

Чи так це – перевірити принаймні на двох функціях *f*.

1.2. ***MyNumInt*** отримує на вході аналітично задану функцію *f*(*x*), запропоновану користувачем, та повертає у графічній формі, як змінюється площа під цією кривою, якщо аргумент x „рухається" від початку інтервалу $x = a$ до його кінця $x = b$. Зрозуміло, на вхід подаються також інтервал $a \le x \le b$ та ціле *N*, число дискретизації інтервалу для чисельного обчислення площ. Чисельно отриману функцію на тому ж графіку порівняйте з аналітично знайденою функцією $I(x) = \int_a^x f(x)\,dx$ та дослідіть, як дві знайдені криві співвідносяться одна до іншої за умови $\frac{b-a}{N} \to 0$ (тобто $N \to \infty$). Перевірити це принаймні на двох функціях *f*.

<u>Додаткові вказівки</u> <u>що враховують типові питання студентів.</u> (А) Чому саме порівнюємо 2 методи? Уявіть собі, ви розв'язали кілька однотипних задач одним якимось методом, та виникла така ж задача, що ним не розв'язується!

Та ось ви винайшли для неї новий метод – як упевнитись, що він правильний? Очевидно, перевірити його у попередніх задачах, перш ніж застосувати до нової. Тому ми й порівнюємо *чисельний* і *аналітичний* методи. (Б) Ваші навички інтегрування функцій не завжди працюють: спробуйте знайти інтеграл від $y = \sin x / x$. Лишається один з чисельних методів. Їх можна застосовувати, якщо описаним методологічним прийомом ви встановили умови його прийнятності.

3. Створіть програму розв'язування системи лінійних алгебраїчних рівнянь $Ax = b$ одним з відомих вам методів (назва програми запропонована):

3.1. методом Гаусса елімінації невідомих (*MyGayss.m*), якщо ваш номер у списку групи є парним[42];

3.2. методом Крамера (*MyCramer.m*), якщо ваш номер у списку групи є непарним[43].

Порада: використовуйте команду ***disp*** для видачі діагностики "*Рішень немає*" та "*Нескінченна кількість рішень*" якщо матриця A та вектор b не гарантують єдиності розв'язку.

P.S. Ці програми вже будуть доволі складні. Не забувайте, тому, пояснювати ваші алгоритми коментарями!

> Цю роботу та всі подальші лабораторні роботи з програмування треба здавати викладачеві не лише із Звітом, та ще й тим самим носієм, CD-R наприклад, де перезаписані зроблені тут програми (m-файли).

[42] Просунутим програмістам пропонується, незважаючи на це, зробити і програму 3.2, бо ця тема дуже важлива!

[43] Просунутим програмістам знов пропонується, незважаючи на це, зробити і програму 3.1, бо ця тема дуже важлива!

Додаток 1

Розв'язати в MATLAB обрані СЛАР:

1. $\begin{cases} 1{,}7x_1 - 2x_2 + x_3 + 0{,}9x_4 = 1 \\ x_1 - 3{,}1x_2 + x_3 - 1{,}2x_4 = -1{,}3 \\ x_1 - 2x_2 + x_3 + 5x_4 = 4{,}9 \\ 3x_2 + 2x_3 - x_4 = 4{,}2 \end{cases}$

2. $\begin{cases} 2{,}1x_1 - 0{,}9x_2 + x_3 + x_4 = 1{,}1 \\ 1{,}5x_1 + 2x_2 - 1{,}1x_3 + 4x_4 = 2 \\ x_1 + 7x_2 - 4x_3 + 11x_4 = 3 \\ 2x_1 + x_3 - x_4 = 4{,}1 \end{cases}$

3. $\begin{cases} 1{,}9x_1 + 1{,}3x_2 - x_3 + x_4 = 1{,}4 \\ 2{,}3x_1 - 2{,}4x_2 + 2x_3 - 3x_4 = 2 \\ 5x_1 + x_2 - x_3 + 2x_4 = -1{,}1 \\ 2x_1 - x_2 + x_3 - 3x_4 = 4 \end{cases}$

4. $\begin{cases} 2{,}3x_1 - 0{,}9x_2 + 1{,}1x_3 - 1{,}7x_4 = 1 \\ 2{,}8x_1 - x_2 - 3{,}2x_4 = 2{,}7 \\ 3x_1 - 1{,}7x_3 + x_4 = -3{,}1 \\ 1{,}9x_1 + 2x_2 - 2{,}3x_3 + 5{,}4x_4 = -6{,}5 \end{cases}$

5. $\begin{cases} x_1 - 2{,}6x_2 + 3{,}1x_3 - 4{,}2x_4 = 4 \\ x_2 - 1{,}3x_3 + x_4 = -3{,}5 \\ 0{,}9x_1 + 3x_2 - 3{,}7x_4 = 1 \\ x_1 - 7x_2 + 3x_3 + 1{,}9x_4 = -2{,}9 \end{cases}$

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

7. $\begin{cases} 1{,}9x_1 + 2{,}1x_2 - 1{,}1x_3 + 5{,}2x_4 = -1{,}1 \\ 3{,}1x_1 - 1{,}3x_2 + 2{,}6x_3 - 6{,}7x_4 = 1{,}3 \\ 4{,}5x_1 + 1{,}1x_2 - 3{,}3x_3 + 6{,}6x_4 = 3{,}1 \\ 0{,}7x_1 - 2{,}4x_2 + 4{,}3x_3 - 6{,}9x_4 = 4 \end{cases}$

8. $\begin{cases} 2x_1 + 4x_2 - 5x_3 + 7x_4 = 1 \\ 2x_1 - 5x_2 + 3{,}3x_3 - 2x_4 = 0 \\ 3{,}09x_1 + 11x_2 - 13x_3 + 15x_4 = -1 \\ 4x_1 - x_2 + x_3 - 3{,}1x_4 = 2 \end{cases}$

9. $\begin{cases} -1{,}61x_2 + x_3 - 3{,}2x_4 = 1{,}1 \\ 2{,}1x_1 + x_2 - 1{,}4x_3 - x_4 = 2{,}3 \\ 6{,}8x_1 + 4{,}3x_2 - 2{,}1x_3 + 3x_4 = 3{,}2 \\ -2x_1 + 2{,}2x_2 + 4x_3 + 4{,}4x_4 = 0{,}8 \end{cases}$

10. $\begin{cases} 0{,}9x_1 - x_2 + 3{,}3x_4 = 8{,}1 \\ x_1 + x_2 + 2{,}1x_3 - x_4 = 2{,}3 \\ 3{,}8x_1 - 2{,}6x_2 + 6{,}3x_3 + 3{,}9x_4 = 1{,}6 \\ 2{,}4x_1 + 4x_2 - 2{,}6x_3 - 7{,}3x_4 = 0{,}5 \end{cases}$

11. $\begin{cases} 1{,}5x_1 + x_2 + 1{,}7x_3 + x_4 = 7{,}4 \\ 3x_1 + 2{,}3x_2 + x_3 + x_4 = -2 \\ x_2 + 2x_3 + 2{,}6x_4 = 23 \\ 5x_1 + 4x_2 + 3x_3 + 3{,}7x_4 = 12 \end{cases}$

12. $\begin{cases} 1{,}3x_1 - 2x_2 + x_3 - 1{,}8x_4 = -1 \\ 2{,}4x_1 + x_2 - 1{,}9x_3 + 2x_4 = 3 \\ 3x_1 - 2x_2 - 1{,}11x_3 + x_4 = 2 \\ 2x_1 - 5{,}4x_2 + x_3 - 2x_4 = -2{,}7 \end{cases}$

Додаток 2

Розв'язати в MATLAB наступні трансцендентні рівняння. Зверніть особливу увагу а те, скільки рівняння має розв'язків (коренів) та чи всі вони видаються тим чи іншим методом.

№ 1. $0{,}5^{x}+1=(x-2)^{2}$

№ 2. $(x-1)^{2}\,ln(x+11)=1$

№ 3. $7\sin x=x+0{,}5$

№ 4. $e^{-2x}-2x+1=0$

№ 5. $x^{2}\cos 2x=-1$

№ 6. $\operatorname{arctg}(x-1)2x=1$

№ 7. $x^{2}-20\sin x=0$

№ 8. $(x-2)^{2}\,lg(x+10)-1=0$

№ 9. $x\log_{5}(x+1)=1$

№ 10. $\cos(x+0{,}3)=x^{2}$

№ 11. $\operatorname{tg}^{3}x=x-1,\ x\in[-\frac{\pi}{2},\frac{\pi}{2}]$

№ 12. $3x^{4}-4x^{3}-12x^{4}+1=0$

Додаток 3 (Розділ 3)

Лістінг програми *MyNumDiff_1.m*

```
function dFdX=MyNumDiff_1(F,a,b,N, Symbol)
%Програма для аналітично заданої функції F знаходить наближено
%масив значень в N узлах її похідної dF,
% також будує на інтервалі [a,b] графіки точної похідної F',
%та і цієї наближеної.
%%Програма дозволяє зробити дослідження того,
%як N (тобто крок диференціювання) впливає на точність
%чисельного диференціювання. Приклад:
%  >> MyNumDiff_1(sin(x),0,2*pi,10,'og');
%-------------------------------------------------------------
%Copyright Ye.Gayev, грудень 2007.

x=a : (b-a)/N : b;   % Крок між точками взяли рівномірний
x2=x(2 : end);%Допоміжн. вектор з N-1 наступних значень аргументу
x1=x(1 : end-1);%Допоміжний вектор з N-1 „попередніх” значень
y=subs(F,x);
%Вектори з N-1 „попередніх” і ”наступних” значень функції:
y2=y(2 : end);  y1=y(1 : end-1);
Dy=y2-y1;  Dx=x2-x1;  % чисельник і знаменник
формули (1.11)
length(Dy);length(Dx);
dFdX=Dy ./ Dx;  % Обчислили похідну за наближ. формулою (1.11)
figure(1)
plot(x1, dFdX, Symbol)  % Побудова графіку чисельної похідної
dF=diff(F);
hold on, ezplot(dF,[a, b]) %Похідна аналітичним шляхом
title('Порівняння двох методів обчислення похідної')
xlabel('x \in [a,b]'); ylabel('y')
legend('Похідна чисельним шляхом', 'Похідна аналітичним шляхом')
```

Додаток 4 (Розділ 3)

Лістінг скрипту Welcome.*m*

```
%Script WELCOME
% greets the User depended on
% the part of Day or Night
% devided into 6 hours, 13, 18, 24 and [0, 6] hours
%Simply run >> Welcome
%% Copyright of Ye. Gayev, Dec. 2005

T=clock; %Returns the array [Year, Month, Day, Hours, Minutes, Seconds]
if (T(4)>5) & (T(4)<=12)
   disp('   ') % to get an empty string on the screen
   disp('Good morning!')
   load chirp; sound(y,Fs)
elseif (T(4)>12) & (T(4)<= 17)
   disp('   ')
   disp('Good afternoon!')
   load gong; sound(y,Fs)
else
   if (T(4)<= 23)
      disp('   ')
      disp('Good evening!')
      load gong; sound(y,Fs)
   else
      disp('   ')
      disp('Good night. But I wish you slept this time!')
   end
end

%Якщо команди load не працюють - її треба закоментувати!
%load handel; sound(y,Fs)
%You may also try
%load chirp; sound(y,Fs)
%load gong; sound(y,Fs)
```

Лістінг скрипту Helicopter

```
%Script HELICOPTER:
% produces a stick that revolves N/2 times in counter-clockwise
direction
% Copyright Ye.Gayev, May 22, 2005
% Simply run   >> Helicopter
        Dt=.01*pi; % Increment of Plotting
        N=10; % Doubled number of Rotations
for t=0:Dt:N*pi
   x1=cos(t); y1=sin(t);
   x2=-x1; y2=-y1;
   Pl=plot([x1, x2], [y1, y2], 'r');
   axis([-1.1 1.1 -1.1 1.1])
   set(gca,'XTick', []); set(gca,'yTick', [])
   hold on;
   plot([0],[0],'o'); hold off; % Symbol and Color of Center Point
   pause(.05) % Speed of Rotations
end
```

Додаток 5 (Розділ 3)

Лістінг скрипту *MyKramer.m*

```
function X=MyKramer(A,b)
% solves SLAE A*X=b by Kramer method
%Matrix A and column of Right Hand Side b
%are the input; returns the SLAE Solution.
%Example:
% >>M=[1 2 3; 1 4 9; 1 8 27]; r=[3; 2; 1];
%>> T=MyKramer(M,r)
%Copyright Ye. Gayev: November 2007.

%Перевіремо, чи правильні вхідні дані:
SizeA=size(A); SizeB=size(b);
N=SizeB(1);
if SizeA(1)~=SizeA(2)
   disp('Матриця має бути квадратною.'),
   disp('Перевірте дані та спробуйте ще раз!')
elseif (SizeB(2)~=1) | (SizeB(1)~=SizeA(1))
   disp('Вектор правої частини "b" має бути стовпчиком.')
   disp(' Перевірте дані та спробуйте ще раз!')
else
   D0=det(A);
   for I=1:N %Дані у порядку, метод Крамера:
       A1=A;
      A1(:,I)=b;
      D(I)=det(A1);
     if (D(I)==0) & (D==0) %Чи є розвязок єдиним?
         disp('Розв"язків СЛАЕ безліч.')
     elseif (D(I)~=0) & (D==0)
         disp('СЛАЕ не має жодного розв"язку.')
     else
         X(I)=D(I)/D0; %Новий Xi на кожному циклі
      end
   end
end
X=X';
```

Додаток 6 (Розділ 3)

Лістінг скрипту *MyGauss.m*

```
function root=MyGauss(M)
% Gauss Matrix Algorithm for Linear Algebraic Equations
% Uses M as extended matrix N*(N+1), where
%  N as its dimension.
%How to run:
%  >> M=[1 2 3 3; 1 4 9 2; 1 8 27 1];  X=MyGauss(M)
%% Ye.Gayev for student labour works, Nov. 17, 2004
% N.B.: No accounting for the case A(Row, Col)=0!

A=M; Dim=size(A); N=Dim(1)
% 1. Direct action of the Gauss Procedure
%Uncomment the commented strings to watch the Calculation Process
%disp('Direct Action of the Gauss is starting. Press a key to begin...'); %pause
for Col=1:N
   for Row=Col:N
      A(Row, Col:end)=A(Row, Col:end)/A(Row, Col);
   end
  %    Col, A
  %    disp('Press a key...')
  %    pause

   for Row=Col+1:N     %       Row
      A(Row, Col:end)=A(Row, Col:end)-A(Col, Col:end);
   end
  %           disp('Press a key...')
  %       pause
end
 %disp('Direct Action completed. Press a key to start Back Action...'); %pause

% 2. Back action of the Gauss Procedure
%disp('Back Path is starting. Press a key to begin...'); pause
for Col=N-1:-1:1
   for Row=Col:-1:1
%    Col, Row, a=A(Row, Col+1)
   A(Row, Col:end)=A(Row, Col:end)-A(Col+1,
Col:end)*A(Row, Col+1);
%    pause
```

```
    end
end
  % disp('Back Action has been completed,'),
  % disp(' Press a key to get the Root...'),
  % pause
   root=A(1:end,N+1);
```

Лістінг програми *Bal.ml* (Розділ 4)

```
function Ball(r,V0,Alfa,Wind)
%Розрахунок траєкторії і швидкості тіла радіусом r
%(маса m=200 гр задана нижче, радіус r береться у см),
%що кинуто із швидкістю V0 під кутом Alfa (у градусах) до
%горизонту
%у газовому середовищі, що горизонтально рухається із
%швидкістю вітру W
%(у тому ж напрямку, якщо W>0, і назустріч тілу,  W<0).
%Приклад запуску програми:
% >> r=6; V0=10; Alfa=60; Wind=5;  Ball(r,V0,Alfa,Wind)
%Copyright Ye. Gayev, November 2007
global W A2 A1;
W=Wind;
m=0.2; Rho1=1.3;
%Маса m=200 грамів, щільність газ. середовища Rho1=1.3
Kg/m3
c2=0.40; c1=.2;
% Коефіцієнти опору сфери у квадратичн. і лінійн. випадках

%%c2=0;
% Коеф опору сфери коли оточуюче середовище опору НЕ створює

r=r*1e-2;  % Радіус у см ==> у метри
S=4*pi*r^2; A2=c2*Rho1*S/(2*m);
% Міделева площа сфери радіусу r та параметр A2
A1=c1*Rho1/m;
% Міделева площа сфери радіусу r та параметр A2
Alfa=Alfa*pi/180; % Градуси у радіани

Delay=.1;
%Штучна затримка у секундах матеріальної точки на графіку

InitCond=[0,0,V0*cos(Alfa),V0*sin(Alfa)];
%Задання початкових умов
opts=odeset('events',@BorderEvent);
[t,y,tFinal]=ode45(@Right2,[0 Inf],InitCond,opts);
```

```
%Пробний розрахунок за квадратичним законом 'Right2'
Ymax=1.1*max(y(:,2));
%Визначення частини траєкторії z(t) ПОНАД землею
Xmax=1.1*max(y(:,1));
%Визначення частини траєкторії z(t) ПОНАД землею
Xmin=1.1*min(y(:,1));
%Визначення частини траєкторії z(t) ПОНАД землею
Tmax=tFinal(2); %Час польоту у секундах

Interval=0:Tmax/50:Tmax;
[t,y]=ode45(@Right2,Interval,InitCond);
%Розрахунок руху, що підкоряється квадратичному закону 'Right2'

[t1,y1]=ode45(@Right1,Interval,InitCond);
%Розрахунок руху тіла, що підкоряється лінійному закону 'Right1'
[xL,zL,VxL,VzL]=LinearMotion(Interval,V0,Alfa);
% лінійний рух тіла за аналітичною формулою
[Zlin,Nlin]=min(zL>=0);
xL(Nlin:end)=xL(Nlin); zL(Nlin:end)=0;%zL(Nlin);

Str1='Траєкторія польоту для вітру ';
Str2=num2str(Wind);
Str3=' м/сек';
Str=[Str1, Str2, Str3];
N=length(t);
% Починається блок графічного виводу результатів
for I=1:N
   hold off
   plot([Xmin, Xmax],[0, 0],'k','LineWidth',3); xlim([Xmin, Xmax]);
   ylim([0, Ymax]); %"Простір" для графіку

   %На випадок руху у середовищі без опору:
%%    [x0,z0,Vx0,Vz0]=EmptyMotion(Interval,V0,Alfa);
%%    plot(x0,z0,'gp')
%%    Htxt01=text(y(round(N/2),1)-(Xmax-Xmin)/5,y(round(N/5),2)
%%   -Ymax/20,['V_x0=', num2str(Vx0)]);
%%    set(Htxt01,'Color','g')
%%    Htxt02=text(y(round(N/2),1)-(Xmax-
%%    Xmin)/5,y(round(N/5),2)-Ymax/10,['V_z0=',
%%    num2str(Vz0(I))]);
```

```
%%    set(Htxt02,'Color','g')
   %Кінець випадку руху у середовищі без опору:

   %На випадок руху у середовищі з лінійним опором
%%%   plot(xL,zL,'mv') %Траєкторія за формулою лінійного руху
   %Кінець випадку руху у середовищі з лінійним опором

   hold on
   plot(y(I,1),y(I,2),'ro') %Положення матеріальної точки,
%що рухається за квадратичним законом сили

   %%%На випадок руху у середовищі з лінійним опором:
%%%   plot(y1(I,1),y1(I,2),'m^')
%Чисельний прогнох руху за лінійним законом сили
%%%   ylim([0, Ymax])

   Htxt0=text(y(round(N/2.2),1),y(round(N/5),2),['Складові
швидкості V_x і V_z']);
   set(Htxt0,'FontWeight','bold')
   Htxt1=text(y(round(N/2),1),y(round(N/5),2)-Ymax/20,['V_x=',
num2str(y(I,3)), ' м/сек']);
   set(Htxt1,'Color','b')
   Htxt2=text(y(round(N/2),1),y(round(N/5),2)-Ymax/10,['V_z=',
num2str(y(I,4)), ' м/сек']);
   set(Htxt2,'Color','b')
   plot(y(1:I,1),y(1:I,2),'b')%Траєкторія точки за квадратичним законом

%%%   plot(y1(1:I,1),y1(1:I,2),'m')%Траєкторія за лінійним опором

   xlabel('Відстань від викиду, м')
   ylabel('Висота над землею, м')
   title(Str)
   text(y(3,1),.2,['\alpha=', num2str(Alfa*180/pi), '^o, V_0=',
num2str(V0), 'м/сек'])
   set(gcf,'WindowStyle','docked')
   pause(Delay)
end
```

```
function DyDt=Right2(t,y)
%Права частина рівнянь руху матеріальної точки у газовому середовищі виду
%   Dy1=y3; %Відповідність позначень
%   Dy2=y4;
%   Dy3=-A*(y3-W)*sqrt((y3-W)^2+y4^2);
%   Dy4=-g-A*y4*sqrt((y3-W)^2+y4^2);
% які відповідають квадратичному закону сили і де позначено
%y1~x(t), y2~z(t), y3~Vx(t), y4~Vz(t)
global W A2
g=9.81;
ModulV=sqrt((y(3)-W)^2+y(4)^2);
DyDt=[y(3);   y(4);  -A2*(y(3)-W)*ModulV; -g-
A2*y(4)*ModulV];


function [value,isterminal,direction] = BorderEvent(t,y)
%Допоміжна підфункція,
%о "слідкує", коли z=y(2) перетне рівень землі z=0
value=y(2);
isterminal=1;
direction=[ ];


function [x,z,Vx,Vz]=EmptyMotion(Interval,V0,Alfa)
%Аналітичні формули руху кульки у пустому просторі
g=9.81;
Vx=V0*cos(Alfa);
Vz=V0*sin(Alfa)-g*Interval;
x=Vx*Interval;
z=V0*sin(Alfa)*Interval-.5*g*Interval.^2;
```

```
function DyDt=Right1(t,y)
%Права частина рівнянь руху матеріальної точки
% газовому середовищі виду
%   Dy1=y3; %Відповідність позначень
%   Dy2=y4;
%   Dy3=-A1*(y3-W);
%   Dy4=-g-A1*y4;
% які відповідають лінійному закону сили і де позначено
%y1~x(t), y2~z(t), y3~Vx(t), y4~Vz(t),   A1=c1*Rho1/m.

global W A1
g=9.81;
DyDt=[y(3);   y(4);  -A1*(y(3)-W); -g-A1*y(4)];


function [x,z,Vx,Vz]=LinearMotion(Interval,V0,Alfa)
%Аналітичні формули руху кульки у середовищі з лінійним опором
global W A1
g=9.81;
Vx0=V0*cos(Alfa);
Vz0=V0*sin(Alfa);
Vx=W+(Vx0-W)*exp(-A1*Interval);
x=W*Interval+(Vx0-W)*(1-exp(-A1*Interval))/A1;
Vz=-g/A1+(g/A1+Vz0)*exp(-A1*Interval);
z=-g/A1*Interval+(g/A1+Vz0)*(1-exp(-A1*Interval))/A1;

% КІНЕЦЬ програми Ball(r,V0,Alfa,Wind).
```

Додаток 8 (Розділ 4)

Лістінг *OrelReska.m*

```
function X=OrelReska(varargin)
%Моделювання кидання монети -- Орел=0 або Решка=1,
%тобто генерація N випадкових чисел (кількість випробувань),
%кожне з яких або 0, або 1 з однаковою ймовірністю.
%(1) Буде проведено ОДНЕ випробування, якщо виконано
%    >> OrelReska   % (тобто без аргументів).
%(2) Буде проведено N випробувань у такому випадку:
%    >> N=5000; X=OrelReska(N); % (тобто з одним аргументом).
%(3) Якщо ж користувач бажає отримати одночасно частоти,
%з якими у виборці зустрічаються випадкові 1 та 0,
%вертайтеся з двома аргументами:
%    >> N=5000; X=OrelReska(N,1);
%%Copyright Ye.Gayev, January 2009.

 Oznaka=0;
%Перевірка кількості вхідних аргументів:
nArgIn=length(varargin);
if nArgIn==0
   N=1;          %Якщо вхідних аргументів немає
elseif nArgIn >2   %Якщо кількість аргументів нерозумна
   disp(' ')
   disp('     ПОЯСНЕННЯ:')
        disp('1. Жодного аргумента не треба, якщо проводите 1
     випробування. Відповіддю буде випадкове число 0 або 1.')
        disp('2. Якщо відразу хочете N випробувань, достатньо
        указати ціле N як ЄДИНИЙ аргумент. Відповіддю буде
                                  виборка N випадкових 0 або 1.')
     disp('3. Якщо ж бажаєте одночасно отримати частоту у цій
     виборці одиничок pN1=p(1), пишіть 1 у другий аргумент.')
           disp('   Також отримаєте частоту у цій виборці нулів
                                                  pN0=p(0).')
  disp('=========================================')
        error('Таким чином, кількість аргументів має бути ноль,
             один або два. Перевірте завдання та спробуйте ще!')
else
   N=varargin{1};
   if nArgIn==2
```

```
      if varargin{2} ~=1
         error('Другим аргументом може бути лише 1, якщо
замовляєте обчислення частот у виборці!')
      else
         Oznaka=1;
      end
   end
end

X=round(rand(1,N));

if Oznaka==1
   disp('Отримано виборку X з кількістю випадків 1 та 0 '); N
   %Тоді, Частота Решки=1 є
   N1=nnz(X);
   disp('Частота 1 у виборці є ');pN1=N1/N
   %і Частота Орла=0 є
   disp('Частота 0 у виборці є ');pN0=(N-N1)/N
end
```

Додаток 9 (Розділ 4)

Лістінг *KubiK.m*

```
function Y=KubiK(varargin)
%Генерація Kidkiv випадкових чисел з набору (1,2,...,N),
%рівномірна ймовірність кожного.
%1. Звертаємось без жодного аргументу –
% видає 1 число з набору (1,2,3),тобто N=3,Kidkiv=1.
%Перший аргумент, якщо є, -- кількість граней "N-гранника",
%і тоді KubiK(N) видає 1 число (Kidkiv=1) з набору (1,2,...,N).
%2. Звертання з двома аргументами, Y=KubiK(N,M),
%генерує M чисел з цього набору.
%3. Якщо ж є третій аргумент і від дорівнює 1,
%тобто Y=KubiK(N,M,1),
%тоді програма видає додатково частоту кожного з чисел,
%що увійшли до виборки Y;
%4. якщо ж третій аргумент відмінний від 1,
%програма скаржиться на це.
%Copyright Ye.Gayev, January 2008.
Oznaka=0;
nArgIn=length(varargin);%Перевірка кількості аргументів
if nArgIn==0
   N=3; Kidkiv=1;%Вхідних арг. немає, N=3-гранник, Kidkiv=1.
elseif nArgIn >3   %Якщо кількість аргументів нерозумна
   disp('  ')
   disp('     ПОЯСНЕННЯ:')
        disp('1. Жодного аргумента не треба, якщо проводите 1
  випробування. Відповіддю буде випадкове число 1,2 або 3.')
           disp('2. Якщо хочете випадкове ціле число від 1 до N
             випробувань, вкажіть це N як ЄДИНИЙ аргумент.')
          disp('3. Якщо ж бажаєте отримати виборку з M таких
                                                    чисел,
                                  пишіть M у другий аргумент.')
         disp('4. Аби одночасно отримати таблицю частот у цій
                                                    виборці
           присутніх у ній чисел pN1=p(Y(i)), пишіть 1 у третій
                                                    аргумент.')
disp('====================================')
```

```
        error('Таким чином, кількість аргументів має бути ноль,
                                                один, два
                    або три. Перевірте завдання та спробуйте ще!')
disp('=========================================
==')
else
   if nArgIn==1
      N=varargin{1}; Kidkiv=1;
   elseif nArgIn==2
      N=varargin{1};  Kidkiv=varargin{2};
   else
      if (varargin{3}~=1)&(varargin{3}~=2)&(varargin{3}~=3)
          warning('Третій аргумент може бути присутнім лише,
                                                якщо
                              замовляєте обчислення частот у виборці!')
          disp('Якщо він дорівнює 1, дамо Ваи таблицю частот.')
          disp('Якщо він дорівнює 2, дамо також графік частот у
                                                порівнянні з теорією.')
              disp('Якщо він дорівнює 3, додатково дамо графік
                                    інтегральних частот у порівнянні з теорією.')
             error('З Вашого завдання незрозуміло, що робити!
                                    Дайте, будь ласка, коректне завдання!')
      else
        N=varargin{1};  Kidkiv=varargin{2}; Oznaka=1;
      end
   end
end

M=Kidkiv;
Y=ceil(N*rand(1,M));

if Oznaka==1
   IntFrequency(1)=0;
         disp('Отримано виборку з кількістю випадкових чисел
                                                1,...,N, що дорівнює');M
   %Обчислюємо частоти у цій виборці кожного з елементів
    disp('Отримано Таблицю частот (праворуч) усіх можливих
                                    випадкових чисел (ліворуч):');
   for i=1:N
```

```
        Number(i)=i;%disp('Current random'),i
         Mi=M-nnz(Y-i);
        Frequency(i)=Mi/M;   %i Частота
        IntFrequency(i+1)=IntFrequency(i)+Frequency(i);
    end
    Result=[Number',Frequency']
    IntFrequency(N+2)=1;
end
if (nArgIn==3)&(varargin{3}==2)
    Ymax1=max(Frequency); Ymax=max([Ymax1,1/N]);
    disp('  ');
    disp('Отриману Таблицю частот дивіться на графіку!');
    Numbers=1:N;
    figure(1), set(1,'Position',[4 245 560 420]);hold off;
plot(Numbers,Frequency,'-gp', Numbers,1/N,'r*'), axis([1-.05,1.01*N,0, Ymax])
title('Фактичні результати випробування і теоретична оцінка')
    xlabel('Значення випадкової величини X')
    ylabel('Частота і ймовірність')
    legend('Частоти чисел виборки','Теоретичні значення')
    hold on,plot(Numbers,0,'bo')
end

if (nArgIn==3)&(varargin{3}==3)
    Ymax1=max(Frequency); Ymax=max([Ymax1,1/N]);
    disp('  ');
          disp('Отриману Таблиці частот і кумулятивних частот
                                                дивіться на графіку!');
    Numbers=[1:N];
    figure(1), set(1,'Position',[4 245 560 420]);hold off;
plot(Numbers,Frequency,'-gp', Numbers,1/N,'r*'), axis([1-
.05,1.01*N,0, Ymax])
    title('Фактичні результати випробування і теоретична
оцінка')
    xlabel('Значення випадкової величини X')
    ylabel('Частота і ймовірність p')
    legend('Частоти чисел виборки','Теоретичні значення')
    hold on,plot(Numbers,0,'bo')
```

```
    Numbers1=0:N+1;
    IntTeory=Numbers1/N; IntTeory(N+2)=1;
    figure(2), set(2,'Position',[512 253 560 420]);hold off;
plot(Numbers1,IntFrequency,'-gp', Numbers1,IntTeory,'r*'),
axis([0,N+1,0, 1.05])
    title('Фактичні результати випробування і теоретична
оцінка')
    xlabel('Значення випадкової величини X')
    ylabel('Інтегральна частота P')
legend('Інтегральні частоти','Теоретичні
                                        значення','Location','NorthWest')
    hold on,plot(Numbers1,0,'bo')
end
disp('Аналіз Вашої "гри" закінчено. Грайте ще!');
disp('~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~'), disp(' ')
```

С П И С О К Л І Т Е Р А Т У Р И

1. **Gayev Ye.A., Nesterenko B.N.** MATLAB for Math and Programming: Textbook. – Zaporozhye: Polygraph, 2006 – 102 p.
2. **Гаєв Є.О., Нестеренко Б.М.** Універсальний математичний пакет MATLAB і типові задачі обчислювальної математики: Навчальний посібник. – К.: НАУ, 2004. – 176 с.
3. **Гаєв Є.О., Нестеренко Б.М.** Збірник лабораторних робіт. К.: НАУ, 2004 р. – 39 с.
4. **Андриевский Б.Р., Фрадлов А.Л.** Элементы математического моделирования в программных средах MATLAB и SciLab. – СПб: Наука, 2001. – 286 с.
5. **Ануфриев И.** Самоучитель MATLAB 5.3/6.x. – СПб.: БХВ-Петербург, 2002. – 736 с.
6. **Говорухин В., Цибулин В.** Компьютер в математическом исследовании. Учебный курс: Maple, MATLAB, LaTex. – СПб: Питер, 2001. – 624 с.
7. **Гультяев А.** MATLAB 5.2. Имитационное моделирование в среде Windows. Визуализация, программирование, анализ данных. - СПб: "Корона", 1999. – 288 с.
8. **Дьяконов В.** MATLAB. Учебный курс. – "Питер", 2001. – 560 с.
9. **Дьяконов В.П., Абраменков И.В.** MATLAB 5.0/5.3. Система символьной математики.– М.: Нолидж, 1999.– 640с.
10. **Ильин С.П.** Вариационное исчисление с применением MATLAB. – Харьков: ХПИ, 2001. – 107 с.
11. Щодо закупівлі ліцензії та використання системи MATLAB для комп'ютеризації навчального процесу у вищих освітніх установах України. Постанова Міністерство освіти і науки України від 2.01.2004. (http://psnk.kpi.ua/index.php?option=com_content&task=view&id=86&Itemid=2).
12. **Лазарєв Ю.Ф.** Початки програмування в середовищі MatLAB. Навч. посібник. – К.: "Політехніка", 2000. – 396 с.
13. **Лазарев Ю. MATLAB 5.x.** – К.: BVH, 2000. – 384 с.
14. **Мартынов Н.Н.** Введение в MATLAB 6.x.– М.:Кудиц-образ,2002.– 352 с.
15. Методи обчислень на персональному комп'ютері. Ч. 1: Системи лінійних алгебраїчних рівнянь. (Уклад. Сулима

І.М., Мейш В.Ф., Гаєв Є.О.). – К.: Нац. аграрн. ун-т, 2002. – 45 с.
16. **Мєтьюз Д.Г., Финк К.Д.** Численные методы. Использование MATLAB. – М.: «Вильямс», 2001. – 720 с.
17. **Потемкин В.Г.** Система инженерных и научных расчетов MATLAB 5.x, в 2х томах. – М.: МИФИ, 1999.
18. **Потемкин В.Г., Рудаков П.И.** MATLAB 5 для студентов. – М.: Диалог–МИФИ. 1999. – 448с.
19. Проведение математических расчетов с использованием системы MATLAB. Метод. пособие. – Харьков: НТУ "ХПИ", 2001. – 56 с.
20. **Чен К., Джиблин П., Ирвинг А.** MATLAB в математическом исследовании. – М.: Мир, 2001. – 346 с.
21. Акастьолова Н.О., Джур О.Є. Розв'язання інженерних та економічних задач в Excel. – Дн-ск: РВВ ДНУ, 2001. – 96 с.
22. **Артамонова Л.А., Халепа Н.В., Клочков В.Е.** Графические возможности среды MathCAD Plus 6.0. – Новомосковск, 1999. – 610 с.
23. **Дьяконов В.** Mathcad 2000. Учебный курс. – СПб: Питер, 2000. – 592 с.
24. **Дьяконов В.** Mathematica 4. Учебный курс. – СПб:Питер, 2001. – 656 с.
25. **Жилкин В.А.** Применение системы MATHCAD при решении задач прикладной математики. – Челябинск, 2001. – 73 с.
26. **Карабутов Н.Н.** Математическое моделирование и анализ систем в среде MATHCAD.Учебн. пособие. – М., 2001. – 79 с.
27. MATHCAD 6.0 PLUS. Финансовые, инженерные и научные расчеты в среде Windows95. – М.: Изд. дом «Филинъ», 1997. – 712 с.
28. **Петров Ю.А.** Современные математические пакеты. Учебн. пособие. – Комсомольск-на-Амуре, 2001. – 148 с.
29. **Плис А.И., Сливина Н.А.** MathCAD 2000: Математический практикум для экономистов и инженеров.– М.:Финансы и статистика,2002. – 656 с.
30. **Пономарев О.П.** и др. Использование математического пакета Maple Y при интегрировании дифференциальных уравнений. Учеб. пособие. – М.: РГОТ УПС, 2000. – 76 с.

31. **Скворцов В.** Explore and Derive. Исследуйте и получите. – Казань: КГТУ, 2001. – 60 с.
32. **Берман Т.Н.** Сборник задач по курсу математического анализа. – М.: Наука, 1985.
33. **Вірченко Н.О., Ляшко І.І.** Графіки елементарних та спеціальних функцій. – К.: Наук. думка, 1996. – 582 с.
34. **Демидович В.П., Марон И.А.** Основы вычислительной математики. – М.: Наука, 1966. – 664 с.
35. Демидович Б.П., Марон И.А., Шувалова Э.З. Численные методы анализа. – М.: Наука, 1967. – 368 с.
36. **Дубовик В.П., Юрик І.І.** Вища математика. Навч. посібник. – К.: А.С.К., 2001. – 648 с.
37. **Зеленский К.Х.** и др. Компъютерные методы прикладной математики. – К.: Дизайн, 1999. – 352 с.
38. **Лященко М.Я., Голованъ М.С.** Чисельні методи. – К.: Либідь, 1996. – 288 с.
39. Методи обчислень: Практикум на ЕОМ: Навч. посібник / Бурківська В.Л., Войцехівський С.О. та ін. – К.: Вища шк., 1995. – 303 с.
40. **Сулима И.М., Гавриленко С.И., Радчик И.А., Юдицкий Я.А.** Основные численные методы и их реализация на микрокалькуляторах. – К.: Вища шк., 1987. – 310 с.
41. **Анго А.** Математика для электро- и радиоинженеров. - М.: Наука, 1964. – 772 с.
42. **Tuck M.** The Real History of the GUI. http://www.sitepoint.com/article/real-history-gui.
43. Wikipedia, Graphical User Interface. http://en.wikipedia.org/wiki/Graphical_user_interface.
44. Сортировка. http://alglib.sources.ru/sorting .
42. **Вирт Н.** Алгоритмы и структуры данных. СпБ..: Изд-во "Невский диалект", 2001. – 352 с.
43. **Ахо А.В., Хопкрофт Д.Э., Ульман Д.Д.** Структуры данных и алгоритмы. – Москва-Киев: Изд. дом. "Вильямс", 2003. – 384 с.
44. Рекомендации по преподаванию программной инженерии и информатики в университетах (Software Engineering 2004). – М.: ИНТУИТ.РУ., 2007. – 462 с.

44a. **Лавріщева К.М.** Програмна інженерія. Підручник. R.: Наук. думка, 2008.

45. **Уоткинс Д.** Основы матричных вычислений. – М.: Бином, 2006.– 664 с.
46. **Хвищун І.О.** Програмування і математичне моделювання: Підручник. – К.: Видавничій дім „Ін Юре”, 2007.– 544 с.
47. **Обруцький А.М.** Фізика на Паскалі: Практикум. – Дніпропетровськ, 2006.– 224 с.
48. **Криводуб Ю.Г.** "Ряди. Навч. посібник", КМУЦА, 1987
49. **Вентцель Е.С., Овчаров Л.А.** Теория вероятностей и ее инженерные приложения. – М.: Наука, 1988. – 490 с.
50. **Мещеряков В.В.** Задачи по математике с MATLAB&SIMULINK.– М.:Диалог-МИФИ, 2007. –528 с.
51. **Мещеряков В.В.** Задачи по статистике и регрессионному анализу с MATLAB. – М.: Диалог-МИФИ, 2009. – 448 с.
52. **Подкур М.Л., Подкур П.Н., Смоленцев Н.К.** Программирование в среде Borland C++ Builder с математическими библиотеками MATLAB C/C++. М. ДМК Пресс. – 2006. 496 с.
53. **Смоленцев Н.К.** МАТЛАБ: Программирование на Visual C#, Borland JBuilder, VBA. 2009.
54. **Сенчевський О.О.** Перші кроки в теорію ймовірностей. – Х.: "Основа", 2008. -- с.
55. **Данциг Т.** Числа – язык науки. М.: "Техносфера", 2008.
56. **Патий Е.** 19 ступеней вверх, или История графических пользовательских интерфейсов. – "IT News", #18/2005 (http://smoking-room.ru/data/pnp/gui_history/).
57. **Гаев Е.А.** Людвиг Прандтль в гидромеханикеп рошлого и будущего. Прикладна гідромеханіка. К.: ІГМ НАН України. Т. 2014, 16 (88), № 2. С. 3 – 16.

Багато додаткової інформації на сайті фірми-розробника The MathWorks, Inc.

Покажчик термінів і команд

Навчальне видання

АЗАРСКОВ Валерій Миколайович
ГАЄВ Євген Олександрович

СУЧАСНЕ ПРОГРАМУВАННЯ.
Частина 1: "Програмування та математика із другом MATLABом"

Навчальний посібник

В авторській редакції

Підписано до друку 15.12.2014 р.
Формат 60Х84/16. Папір офсетний.
Ум.-друк. арк. 14,76. Наклад 300 прим.
Зам № 1512/14

Видавець: ТОВ "НВП "Інтерсервіс",
Київ, вул. Бориспільська, 9,
Свідоцтво: серія ДК №3534 від 24.07.2009 р.

Виготовлювач: СПД Андрієвська Л.В.
м. Київ, вул. Бориспільська, 9.
Свідоцтво: серія В03 №919546 від 19.09.2004 р.