Лабораторна робота № 1
Вступ у систему R

1.1. Мета роботи

Вивчення можливостей математичного пакета **R**, придбання навичок використання цього пакета для розв'язання математичних задач та графічного представлення результатів досліджень на комп'ютері.

1.2. Методичні рекомендації щодо організації самостійної роботи

За темою лабораторної роботи студент повинен: *знати* загальні поняття лінійної алгебри та програмування; *вміти* застосовувати засоби пакета R для розв'язання простих математичних задач.

1.2.1. Числові вектори

Вектори в системі R формуються функцією **c()**. Аргументи цієї функції є компонентами вектора. Наприклад, команда

```
> a = c(1, 2, 3, 4, 5)
> a
[1] 1 2 3 4 5
```

формує вектор-рядок **(1, 2, 3, 4, 5)** і привласнює його змінній **a**.

Аргументи функції **c()** можуть бути векторами. В цьому випадку як результат маємо конкатенацію цих векторів. Скалярні значення (тобто числа) сприймаються R як вектори довжини 1. Таким чином, аргументами функції **c()** можуть бути як вектори, так і скаляри.

Наприклад:

```
> c(c(1, 2, 3, 4, 5), 6, c(7, 8))
[1] 1 2 3 4 5 6 7 8
```

Вектор, що складається з послідовних чисел, можна отримати за допомогою команди <початкове_значення>:<кінцеве_значення>. Наприклад, 1:5. На цю команду схожа функція **seq**, яка генерує відрізки арифметичної прогресії. Можна задати початкове значення, кінцеве значення і крок:

```
> seq(0, 1, by=0.1)
 [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
```

В іншому варіанті задається початкове значення, кінцеве значення і кількість точок. У результаті буде згенерований вектор, що складається з заданої кількості компонент, рівномірно розподілених на заданому відрізку:

```
> seq(0, 1, len=11)
 [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
```

Функція **rep(v,k)** створює вектор, що складається з **k** копій вектора **v**. Наприклад:

```
> rep(c(1, 2), 3)
[1] 1 2 1 2 1 2
```

Над векторами можна виконувати арифметичні операції й елементарні функції. Елементарні математичні функції застосовуються до кожної компоненти вектора. Наприклад:

```
> x = c(0, pi/2, pi)
> sin(x)
[1] 0.000000e+00 1.000000e+00 1.224606e-16
```

Доступ до елементів вектора здійснюється оператором **[i]**. Наприклад, **u[5]** – це 5-й елемент вектора **u**. Нумерація елементів починається з 1. Вирази виду **u[i]** можуть зустрічатися і в лівій частині від знака привласнення. При цьому, якщо вектор має довжину не менше **i**, то **u[i]** просто прийме нове значення. В інакшому випадку вектор **u** збільшить свою довжину до **i**, компонента **u[i]** прийме нове значення, а останні нові компоненти приймуть значення **NA**.

```
> u = 1
> u[5] = 5
> u
[1]  1 NA NA NA  5
```

Деякі корисні функції:

crossprod(v, w) – скалярний добуток векторів **v** та **w**;

sum(v) – сума елементів вектора **v**;

prod(v) – добуток елементів вектора **v**;

max(v) – максимальний елемент;

min(v) – мінімальний елемент;

length(v) – довжина вектора;

mean(v) – середнє значення елементів вектора **v** (оцінка математичного сподівання);

var(v) – варіація елементів вектора v (оцінка дисперсії);

sort(v) – повертає вектор тієї ж довжини, що і **v**, з елементами, відсортованими в порядку збільшення.

1.2.2. Рядкові назви елементів вектора

Елементам вектора (числового, логічного, символьного) можна задавати імена. Наприклад:

```
> fruit = c(5, 10, 1, 20)
> names(fruit) = c("orange", "banana","apple", "peach")
> fruit
orange banana  apple  peach
     5     10      1     20
```

Створений вектор довжиною 4, компоненти якого мають вказані назви. Тепер звертатися до елементів вектора можна як по індексу, так і по імені:

```
> fruit[1] = 6
> fruit["peach"] = 60
> fruit
orange banana  apple  peach
     6     10      1     60
```

1.2.3. Матриці

Числову матрицю можна побудувати з числового вектора за допомогою функції **matrix**. Треба вказати кількість рядків **nrow=m** і/або кількість стовпців **ncol=m**. У результаті елементи з вектора будуть записані в матрицю вказаних розмірів. Наприклад:

```
> matrix(1:6, nrow=2, ncol=3)
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
```

Довжина вектора має бути кратною добутку потрібної кількості рядків і потрібної кількості стовпців:

```
> matrix(1, nrow=2, ncol =2)
     [,1] [,2]
[1,]    1    1
[2,]    1    1
```

```
> matrix(1:2, nrow=3, ncol =2)
     [,1] [,2]
[1,]    1    2
[2,]    2    1
[3,]    1    2
```

Елементи записуються в матрицю по стовпцях. Щоб записати їх по рядках, треба задати значення додаткового параметра **byrow=TRUE**:

```
> matrix(1:6, nrow=2, byrow=TRUE)
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
```

Функції **nrow(A)** і **ncol(A)** повертають відповідно кількість рядків і стовпців матриці **A**.

Доступ до елементів матриці відбувається по індексу. Так **A[i,j]** посилається на елемент **i**-го рядка і **j**-го стовпця матриці **A**.

A[i,] еквівалентне **A[i,1:ncol(A)]**, а **A[,j]** еквівалентне **A[1:nrow(A),j]**. Можливий доступ до елементів матриці з допомогою одного індексу. Наприклад, **U[5]** означає 5-й елемент матриці **U**, якщо вважати, що елементи перенумеровані по стовпцях. Якщо **i**-вектор, то **A[i]** – означає вибірку відповідних елементів і т. д.

Можна задати імена стовпцям і рядкам матриці:

```
> A = matrix(c(23, 31, 58, 16), nrow=2)
> rownames(A) = c("petal","sepal")
> colnames(A) = c("length","width")
> A
      length width
petal     23    58
sepal     31    16
```

Після цього доступ до рядків і стовпців може відбуватися за ім'ям. Наприклад:

```
> A["petal","length"]
[1] 23
```

Функція **A = cbind(A, B)** створює матрицю, приписуючи до **A** справа **B** (для цього кількість рядків у **A** і **B** має співпадати).

Функція **A = rbind(A, B)** створює матрицю, приписуючи до **A** знизу **B** (для цього кількість стовпців у **A** і **B** має співпадати). Відмітимо, що в списках аргументів функцій **cbind** и **rbind** можна вказати більше двох матриць.

Арифметичні операції над матрицями здійснюються по компонентно, тому, наприклад, щоб додати дві матриці, вони повинні мати однакові розміри.

Елементарні математичні функції також застосовуються до елементів матриці покомпонентно.

Транспонування матриці здійснює функція **t(A)**, а матричний добуток – операція **%*%**:

```
> A = matrix(1:6,nrow=2)
> A
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> B = t(A)
> B
     [,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]    5    6
> A%*%B
     [,1] [,2]
[1,]   35   44
[2,]   44   56
> B%*%A
     [,1] [,2] [,3]
[1,]    5   11   17
[2,]   11   25   39
[3,]   17   39   61
```

Для розв'язання системи лінійних рівнянь **Ax = b** з квадратною невиродженою матрицею **A** є функція **solve(A,b)**:

```
> A = matrix(1:4,ncol =2)
> A
     [,1] [,2]
[1,]    1    3
[2,]    2    4
> b = c(5,8)
> b
[1] 5 8

> solve(A,b)
[1] 2 1
```

Функція **det(A)** знаходить визначник, а **solve(A)** – обернену матрицю:

```
> det(A)
[1] -2
```

```
> solve(A)
     [,1] [,2]
[1,]   -2  1.5
[2,]    1 -0.5
```

1.2.4. Графіки функцій

Для зображення графіків функцій у R є функція **plot(x,y)**. Тут **x** – вектор значень абсцис і **y** – вектор значень ординат. Якщо вказаний тільки один аргумент – **plot(y)**, то передбачається, що **x=1:n**, де **n** – довжина вектора **y**. Наприклад, команди

```
> x = seq(-pi, pi, len =30)
> y = sin(x)
> plot(x,y)
```

рисують 30 точок, що лежать на синусоїді **y=sin(x)** (рис. 1.1). Графіки з'являються в окремому вікні (або в кількох окремих вікнах).

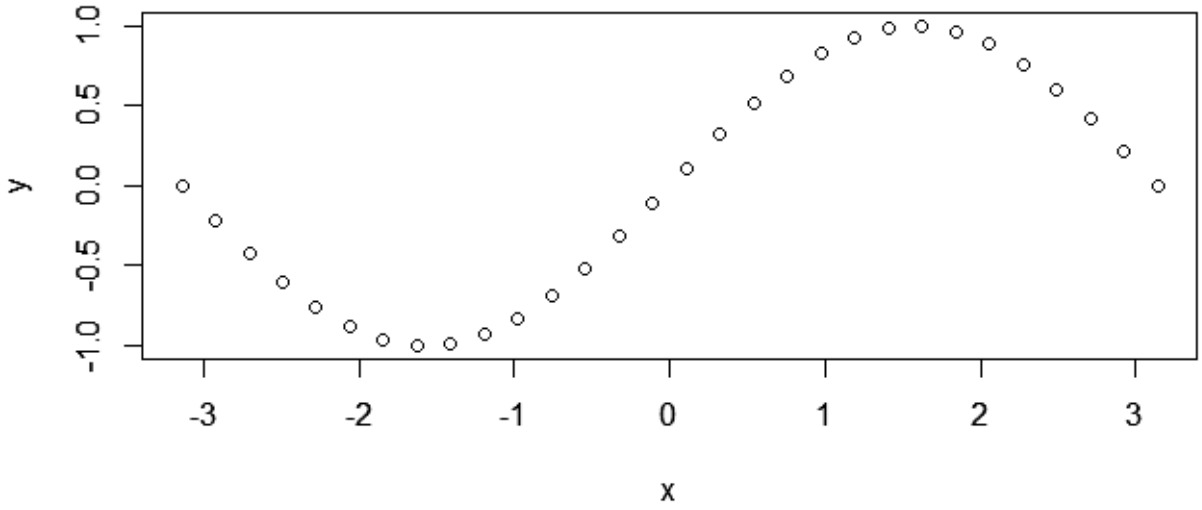

Рис. 1.1. **"Точковий" графік синуса**

Функція **plot** допускає багато додаткових параметрів. Деякі з них:

type=тип-графіка дозволяє вказати тип графіка, що виводиться; деякі значення параметра – "p" (точки, за замовчуванням), "l" (лінії), "b" (лінії і точки), "o" (лінії і точки перекриваються), "n" (нічого не рисується);

xlab=назва-осі-абсцис і **ylab=назва-осі-ординат** дозволяє вказати підписи до осі абсцис і ординат відповідно;

main=основний-надпис і **sub=додатковий-надпис** створюють надписи зверху і знизу графіка;

col=колір задає колір графіка; можливі значення "blue", "red", "green", "cyan", "magenta", "yellow", "black" та багато інших. Кольори можна задати rgb-вектор-функцією **rgb(r, g, b)**. Параметри цієї функції – значення від 0 до 1.

lty=стиль-лінії визначає стиль лінії; можливі значення "blank" (немає лінії), "solid", "dashed", "dotted", "dotdash", "longdash", "twodash".

Розглянемо наступний приклад:

```
> plot(x, y, type="l", col="blue", main="Sine of x")
```

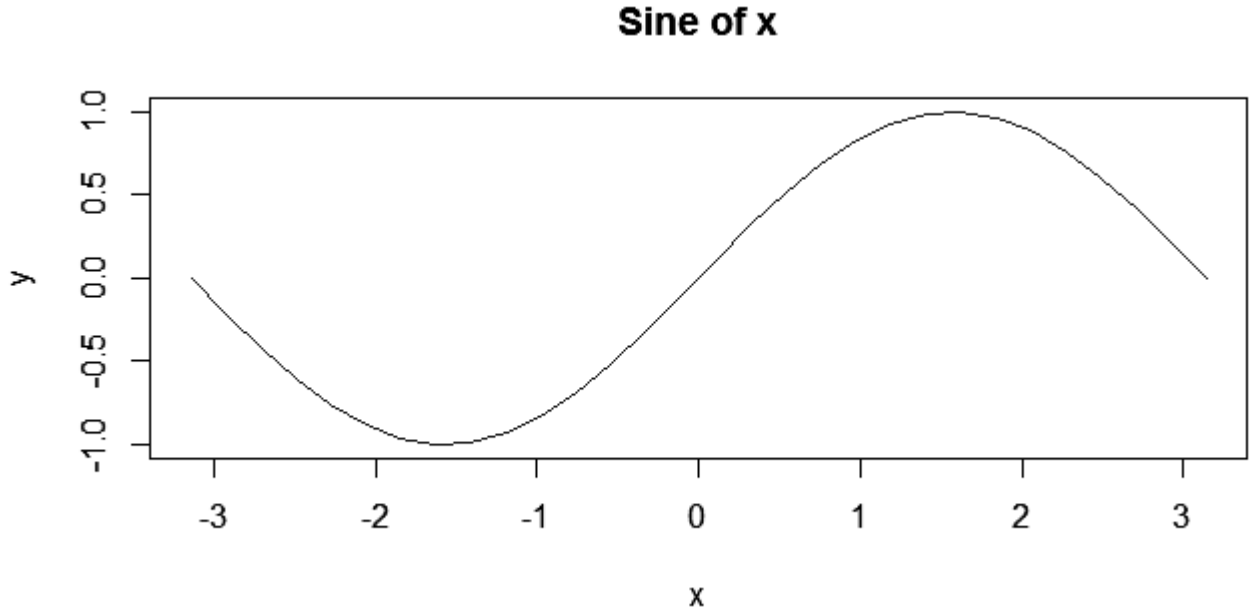

Рис. 1.2. **Графік синуса**

Наступні функції (що називаються низькорівневими) ніколи не стирають зображення, яке вже є. Їм можуть бути передані також додаткові параметри, аналогічні відповідним параметрам функції **plot**:

points(x,y) рисує додаткові точки, **lines(x,y)** рисує ломану лінію, яка з'єднує вказані точки. Цим функціям можуть бути передані додаткові аргументи, наприклад, **type=тип-графика** (за замовчуванням він дорівнює "p" для points і "l" для lines).

text(x,y,текст) додає текст. За замовчуванням він центрується навколо точки **(x,y)**. Вказані параметри можуть бути векторами. В цьому випадку буде розміщено кілька надписів. Можна вказати шрифт **font=шрифт** та ін.

abline(k,b) рисує пряму y=kx+b. Можна вказати колір і стиль лінії тощо.

abline(h=y), **abline(v=x)** – для рисування горизонтальних і вертикальних прямих. Вказуються відповідно координата **y** і **x**.

polygon(x,y) рисує багатокутник з вершинами **(x,y)**. Можна вказати колір заповнення **col=колір** і колір границі **border=колір**.

legend(x,y,легенда) додає легенду у вказану точку.

Розглянемо приклад:

```
> x = seq(-10, 10, by=0.1)
> y1 = x^2+2*x-5
> y2 = -x^2+60
> plot(x, y1, type="n",main="Графіки функцій")
> colors = c("blue", "red", "green")
> lines(x,y1, col=colors[1])
> lines(x,y2, col=colors[2], lty="dashed")
> legends = paste("y",1:2, sep="")
> legend(3, 110, legends, lty="solid",col=colors)
> abline(h=0) #Горизонтальна вісь
> abline(v=0) #Вертикальна вісь
```

Результат наведений на рис. 1.3

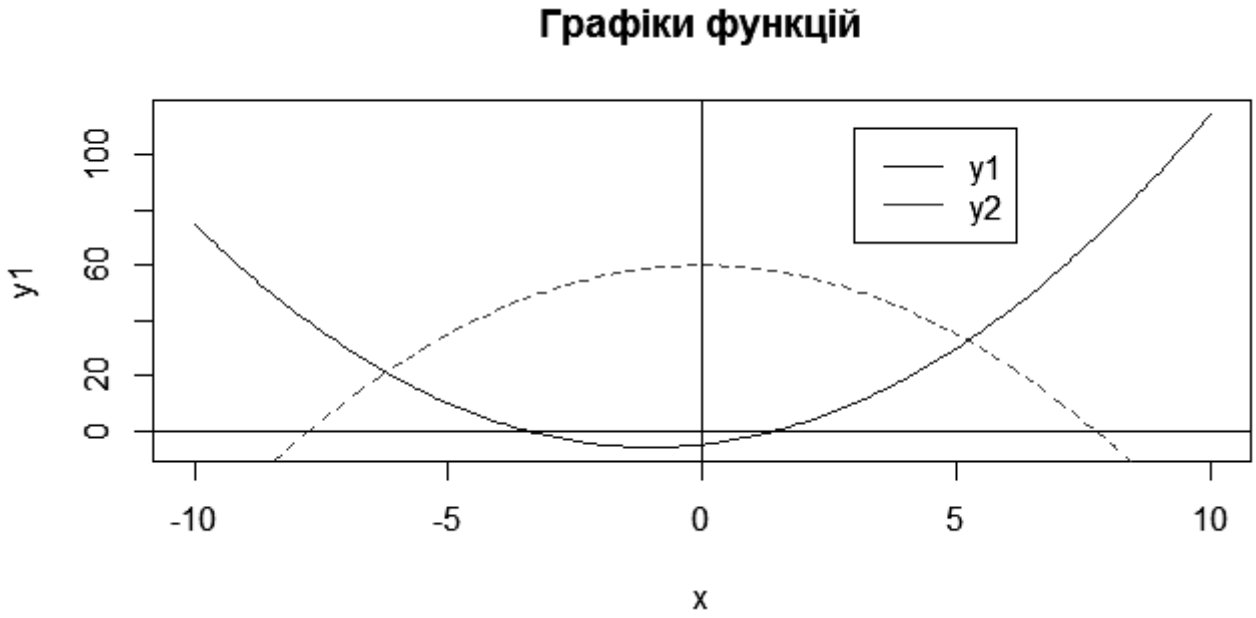

Рис. 1.3. **Два графіка в одному графічному вікні**

1.2.5. Управляючі конструкції

Кожна команда чи вираз у мові R повертає деякий результат. Навіть привласнення – це вираз, результат якого – привласнюване значення. Вираз може бути частиною іншого виразу. Зокрема, можливі множинні привласнювання, наприклад:

```
> x = y = z = 0
```

Перевірка умов здійснюється так, як і в мові С:

```
> if(умова) {вираз1} else {вираз2}
```

де *умова* – це логічний вираз, результатом якого є скалярне логічне значення.

Цикли з управляючою змінною реалізуються з допомогою конструкції:

```
> for (змінна in вираз1) {вираз2}
```

Результатом *вираз1* має бути вектор, при цьому змінна на кожній ітерації циклу приймає значення чергового елемента цього вектора. Кількість ітерацій дорівнює кількості елементів у векторі.

```
> s=0
> for(i in 1:20)  s = s+i
> s
[1] 210
```

Відмітимо, що той же результат швидше можна отримати за допомогою команди. **sum(1:20)**

Цикли з передумовою реалізуються за допомогою конструкції:

```
> while (умова) {вираз}
```

Передчасний вихід з циклу здійснюється командою **break**. Для переривання поточної ітерації і переходу до наступної служить команда **next**. Ці команди можна також використовувати і з конструкцією **for**.

Безкінечний цикл реалізує команда **repeat**:

```
> repeat {вираз}
```

Всередині цієї конструкції можна використовувати команди **break** і **next**.

1.2.6. Функції користувача

Формат визначення функції такий:

```
> ім'я = function(arg1,arg2, ...) {вираз}
```

Тут список формальних аргументів (формальних параметрів) **арг1, арг2,...** може мати довільну довжину (в тому числі, бути пустим). Як правило, вираз (тіло функції) є блоком. Значення, яке повертає функція, це значення цього виразу. Дострокове повернення з функції здійснює команда **return(вираз1)**. У цьому випадку функція повертає значення вказаного виразу.

Звернення до функції має вид **ім'я(вираз1, вираз2, ...)**. Тут значення виразів **вираз1, вираз2,...** є фактичними аргументами, які підставляються замість відповідних формальних аргументів.

При зміні значень формальних аргументів всередині функції, не змінюються відповідні значення фактичних аргументів. Єдиний спосіб зробити це – використовувати в тілі функції <сильне> привласнювання: **arg1 <-- вираз**.

Якщо фактичні аргументи задані в <іменованому> вигляді **ім'я = вираз**, то вони можуть бути перераховані в довільному порядку.

Більш того, список фактичних аргументів може починатися з аргументів, представлених у звичайній позиційній формі, після чого можуть іти аргументи в іменованій формі.

Наприклад, є функція **fun**, задана таким чином:

```
> fun = function(arg1,arg2,arg3,arg4)
+   {
+     #тіло функції опущене
+   }
```

Тоді **fun** може бути викликана численними еквівалентними способами, наприклад:

```
> ans = fun(d,f, 20, TRUE)
> ans = fun(d,f, arg4=TRUE, arg3=20)
> ans = fun(arg1=d, arg3=20, arg2=f, arg4=TRUE)
```

У багатьох випадках при визначенні функції деяким аргументам можуть бути привласнені значення за замовчуванням. Тоді при виклику функції ці аргументи можуть бути опущені. Припустимо, що функція **fun** визначена як

```
> fun = function(arg1, arg2, arg3=20, arg4=TRUE)
+ {
+   #тіло функції опущене
+ }
```

Тоді звертання до цієї функції виду **fun(d,f)** еквівалентне трьом, приведеним вище. Наступне звертання

```
> fun(d,f, arg4=FALSE)
```

змінює одне із значень за замовчуванням.

Відзначимо, що в значеннях за замовчуванням можна використовувати будь-які вирази, у тому числі такі, що містять інші аргументи функції.

Усі символи, що зустрічаються в тілі функції, діляться на три групи: це формальні аргументи, локальні змінні й вільні змінні.

Формальні аргументи (формальні параметри) – це аргументи, перераховані в заголовку функції (в круглих дужках після ключового слова **function**).

Локальні змінні – це змінні, які не є формальними аргументами, значення яких визначаються під час виконання функції.

Змінні, які не є формальними аргументами і локальними змінними, є вільними змінними.

Наприклад, у функції

```
> f = function(x)
+ {
+   y = 2*x
+   print(x)
+   print(y)
+   print(z)
+ }
```

x – формальний аргумент, **y** – локальна змінна, **z** – вільна змінна.

Область видимості формальних аргументів і локальних змінних – лише сама ця функція. Це означає, що зміни цих змінних всередині функції ніяк не відображаються на змінних з такими ж іменами у зовнішній функції. Область видимості вільних змінних розповсюджується до зовнішньої функції, в якій вони були визначені. Зміна таких змінних у тілі функції впливає також на відповідні змінні в цій зовнішній функції.

1.3. Порядок виконання роботи і варіанти завдань

1.3.1. Зміст звіту

У практичній частині роботи необхідно:

- виконати всі приклади наведені в розділі 1.2;
- виконати завдання наведене нижче;
- привести текст складеної програми та результати її виконання.

1.3.2. Завдання

Запишіть функцію, яка приймає на вхід числовий вектор **x** і число розбиття інтервала **k** (за замовчуванням дорівнює кількості елементів вектора, поділеному на 10) і виконує таке: знаходить мінімальне і максимальне значення елементів вектора **xmin** й **xmax**, ділить отриманий відрізок **[xmin; xmax]** на **k** рівних інтервалів і підраховує кількість елементів вектора, що належать кожному інтервалу. Далі має будуватися графік, де по осі абсцис – середини інтервалів, по осі ординат – кількість елементів вектора, що належать інтервалу, поділене на загальну кількість точок. Проведіть експеримент на даній функції, де x – вектор довжиною 5 000, згенерований із нормально розподіленої випадкової величини з математичним сподіванням $a = 5.5$ і середньоквадратичним відхиленням – $\sigma = 1.5$ (функція **rnorm**).

1.4. Контрольні запитання

1. Які є засоби в системі R для створення та роботи з векторами та матрицями?

2. Які є засоби в системі R для побудови графіків?
3. Які є засоби в системі R управляючі конструкції при програмуванні?
4. Як описати функцію користувача в системі R?