Отчет по лабораторной работе 4

Тема: Векторное программирование

Сведения о студенте

Дата: 2025-10-10
Семестр: 2 курс 1 полугодие - 3 семестр
Группа: ПИН-б-о-24-1 (2)
Дисциплина: Технологии программирования
Студент: Макаров Роман Дмитриевич

Цель работы

познакомиться с особенностями векторного программирования в R. Решить задания в соответствующем стиле программирования. Составить отчет.

Теоретическая часть

Векторное программирование — это парадигма программирования, в которой операции выполняются над целыми массивами данных одновременно, без использования явных циклов. R является языком с естественной поддержкой векторных операций.

Основные структуры данных в R:

1. **Вектор (vector)** - базовый объект R:
    - Однородная коллекция элементов одного типа
    - Создание: `c()`, `:`, `seq()`, `rep()`
    - Индексация: `vector[condition]`, `vector[index]`
2. **Матрица (matrix)** - двумерный массив:
    - Все элементы одного типа
    - Создание: `matrix()`, `cbind()`, `rbind()`
3. **Список (list)** - гетерогенная коллекция:
    - Может содержать элементы разных типов и размеров
    - Создание: `list()`
4. **Data Frame** - таблица данных:

- Список векторов одинаковой длины
- Столбцы могут быть разных типов
- Создание: `data.frame()`

Векторизованные операции:

```
# Арифметические операции
x <- 1:5
y <- x * 2  # 2, 4, 6, 8, 10

# Логические операции
z <- x[x > 3]  # 4, 5

# Математические функции
sqrt(x)  # применяется к каждому элементу
```

Функции семейства apply:

- **`apply()`** - применение функции к строкам/столбцам матриц
- **`lapply()`** - применение к списку, возвращает список
- **`sapply()`** - упрощенная версия lapply
- **`vapply()`** - apply с предопределенным типом возврата

Пример использования apply:

```
M <- matrix(1:16, 4, 4)
apply(M, 1, mean)  # среднее по строкам
apply(M, 2, sum)   # сумма по столбцам
```

Создание пользовательских функций:

```
function_name <- function(arg1, arg2, ...) {
  # тело функции
  вычисления
  return(результат)
}
```

Обработка пропущенных значений (NA):

- `is.na()` - проверка на NA
- `!is.na()` - исключение NA
- `na.omit()` - удаление строк с NA

Преимущества векторного программирования:

- Высокая производительность
- Лаконичный и читаемый код
- Упрощение сложных операций

- Естественность для статистических вычислений

Ключевые функции для работы с данными:

- `mean()`, `sd()` - описательные статистики
- `abs()` - абсолютное значение
- `which()` - индексы элементов, удовлетворяющих условию
- `unlist()` - преобразование списка в вектор

Векторный подход в R позволяет эффективно работать с большими объемами данных, минимизируя использование циклов и повышая производительность вычислений.

Выполненные задания

Задание 1 - Предобработка данных

Предобработка данных. Создайте новый вектор `my_vector`, следующей строчкой:

```
my_vector <- c(21, 18, 21, 19, 25, 20, 17, 17, 18, 22, 17, 18, 18, 19, 19, 27, 21, 20,
24, 17, 15, 24, 24, 29, 19, 14, 21, 17, 19, 18, 18, 20, 21, 21, 19, 17, 21, 13, 17, 13,
23, 15, 23, 24, 16, 17, 25, 24, 22)
```

В векторе `my_vector` отберите только те наблюдения, которые отклоняются от среднего меньше, чем на одно стандартное отклонение. Сохраните эти наблюдения в новую переменную `my_vector2`. При этом исходный вектор оставьте без изменений.
Полезные функции: `mean(x)` – среднее значение вектора `x`; `sd(x)` – стандартное отклонение вектора `x`; `abs(x)` – абсолютное значение числа n

---

Задание 2

Напишите функцию get_negative_values, которая получает на вход dataframe произвольного размера. Функция должна для каждой переменной в данных проверять, есть ли в ней отрицательные значения. Если в переменной отрицательных значений нет, то эта переменная нас не интересует, для всех переменных, в которых есть отрицательные значения мы сохраним их в виде списка или матрицы, если число элементов будет одинаковым в каждой переменной (смотри пример работы функции).

```
test_data <- as.data.frame(list(V1 = c(-9.7, -10, -10.5, -7.8, -8.9), V2 = c(NA, -10.2, -10.1
get_negative_values(test_data)

$V1
[1]  -9.7 -10.0 -10.5  -7.8  -8.9
$V2
[1] -10.2 -10.1  -9.3 -12.2
$V3
```

```
[1]  -9.3 -10.9  -9.8

test_data <- as.data.frame(list(V1 = c(NA, 0.5, 0.7, 8), V2 = c(-0.3, NA, 2, 1.2), V3 =
c(2, -1, -5, -1.2)))
get_negative_values(test_data)

$V2
[1] -0.3
$V3
[1] -1.0 -5.0 -1.2

test_data <- as.data.frame(list(V1 = c(NA, -0.5, -0.7, -8), V2 = c(-0.3, NA, -2, -1.2),
V3 = c(1, 2, 3, NA)))
get_negative_values(test_data)

       V1   V2
[1,] -0.5 -0.3
[2,] -0.7 -2.0
[3,] -8.0 -1.2
```

Ход работы

Задание 1.

Программный код:

```
my_vector <- c(21, 18, 21, 19, 25, 20, 17, 17, 18, 22, 17, 18, 18, 19, 19, 27, 21, 20,
               24, 17, 15, 24, 24, 29, 19, 14, 21, 17, 19, 18, 18, 20, 21, 21, 19, 17,
               21, 13, 17, 13, 23, 15, 23, 24, 16, 17, 25, 24, 22)

mean_val <- mean(my_vector)
sd_val <- sd(my_vector)
my_vector2 <- my_vector[abs(my_vector - mean_val) < sd_val]
```

Результат работы программы:

```
> my_vector
 [1] 21 18 21 19 25 20 17 17 18 22 17 18 18 19 19 27 21 20 24 17 15 24 24 29 19
[26] 14 21 17 19 18 18 20 21 21 19 17 21 13 17 13 23 15 23 24 16 17 25 24 22
> my_vector2
 [1] 21 18 21 19 20 17 17 18 22 17 18 18 19 19 21 20 17 19 21 17 19 18 18 20 21
[26] 21 19 17 21 17 23 23 17 22
```

Задание 2.

Программный код:

```
get_negative_values <- function(df) {
  neg_list <- lapply(df, function(x) x[!is.na(x) & x < 0])
  neg_list <- neg_list[sapply(neg_list, length) > 0]

  # Проверка на одинаковую длину для преобразования в матрицу
  lengths <- sapply(neg_list, length)
  if(length(unique(lengths)) == 1 && all(lengths > 0)) {
    return(do.call(cbind, neg_list))
  } else {
    return(neg_list)
  }
}
```

Результат работы программы:

```
> test_data1 <- as.data.frame(list(
+   V1 = c(-9.7, -10, -10.5, -7.8, -8.9),
+   V2 = c(NA, -10.2, -10.1, -9.3, -12.2),
+   V3 = c(NA, NA, -9.3, -10.9, -9.8)
+ ))
> test_data1
     V1    V2    V3
1  -9.7    NA    NA
2 -10.0 -10.2    NA
3 -10.5 -10.1  -9.3
4  -7.8  -9.3 -10.9
5  -8.9 -12.2  -9.8
> get_negative_values(test_data1)
$V1
[1]  -9.7 -10.0 -10.5  -7.8  -8.9

$V2
[1] -10.2 -10.1  -9.3 -12.2

$V3
[1]  -9.3 -10.9  -9.8

> test_data2 <- as.data.frame(list(
+   V1 = c(NA, 0.5, 0.7, 8),
+   V2 = c(-0.3, NA, 2, 1.2),
+   V3 = c(2, -1, -5, -1.2)
+ ))
> test_data2
   V1   V2   V3
1  NA -0.3  2.0
2 0.5   NA -1.0
3 0.7  2.0 -5.0
4 8.0  1.2 -1.2
> get_negative_values(test_data2)
$V2
[1] -0.3
```

```
$V3
[1] -1.0 -5.0 -1.2

> test_data3 <- as.data.frame(list(
+   V1 = c(NA, -0.5, -0.7, -8),
+   V2 = c(-0.3, NA, -2, -1.2),
+   V3 = c(1, 2, 3, NA)
+ ))
> test_data3
    V1   V2 V3
1   NA -0.3  1
2 -0.5   NA  2
3 -0.7 -2.0  3
4 -8.0 -1.2 NA
> get_negative_values(test_data3)
       V1   V2
[1,] -0.5 -0.3
[2,] -0.7 -2.0
[3,] -8.0 -1.2
```

Результаты

Выводы

1. В ходе лабораторной работы освоены принципы векторного программирования в R, позволяющие эффективно обрабатывать данные без использования явных циклов. Продемонстрирована работа с базовыми структурами данных: векторами, матрицами и датафреймами.

2. Практическое применение векторизованных операций и функций семейства apply показало их преимущество в производительности и читаемости кода по сравнению с итеративными подходами. Созданные функции для обработки данных подтвердили эффективность векторных вычислений для задач анализа и предобработки.

3. Полученные навыки векторизованного программирования составляют основу для эффективной работы с большими объемами данных в R и являются фундаментальными для дальнейшего изучения статистического анализа и машинного обучения.

Ответы на контрольные вопросы

1. **Векторизация**

Ответ: *Векторизация* – поэлементное одновременное выполнение действий над всеми элементами.

2. **Основные объекты языка R**

Ответ: *Вектор* – основной объект языка R. Вектор может содержать более одного значения, все объекты в векторе имеют одну природу.
Лист – элемент языка R который может содержать разные по размеру и типу данных векторы.
Дата фрейм – элемент языка R который может содержать одинаковые по размеру, но разные по типу данных векторы. Дата фрейм является разновидностью листа.

3. **Создание собственных функций**

Ответ: Для создания собственных функций в R используется следующая конструкций:
Конструкция создания функций в R:

```
function_name <- function(argument_1, argument_2){
  function_body
  ...
  return(value)
}
```

Пример создания и вызова функции «Площадь круга»:

```
area_circle <- function(r){
  area <- r^2*pi
  return(area)
}
# Вызов функции
area_circle(5)
```

4. **Векторизованные функции семейства apply**

Ответ: *Векторизованные функции:*
Характерной особенностью R является векторизация вычислений. Векторизация представляет собой один из способов выполнения параллельных вычислений, при котором программа определенным образом модифицируется для выполнения нескольких однотипных операций одновременно.
Принцип векторизованных вычислений применим не только к векторам как таковым, но и к более сложным объектам R - матрицам, спискам и таблицам данных (для R разницы между последними двумя типами объектов не существует – фактически таблица данных является списком из нескольких компонентов-векторов одинакового размера). В базовой комплектации R имеется целое семейство функций, предназначенных для организации векторизованных вычислений над такими объектами. В названии всех этих функций имеется слово apply (англ. применить), которому предшествует буква, указывающая на принцип работы той или иной функции.
Функция apply() - используется в случаях, когда необходимо применить какую-либо функцию ко всем строкам или столбцам матрицы (или массивов большей размерности):

Пример работы с функцией apply:

```
# Создадим обычную двумерную матрицу
M <- matrix(seq(1, 16), 4, 4)
```

```
# Найдем минимальные значения в каждой строке матрицы
apply(M, 1, min)
# Найдем минимальные значения в каждом столбце матрицы
apply(M, 2, max)
```

Ответы на вопросы для поиска и письменного ответа

1. **Особенности языка R**

Ответ: интерпретируемость, богатая статистическая функциональность, мощная система визуализации, активное сообщество пользователей.

2. **Языки с векторизацией**

Ответ: MATLAB, Julia, Python (с библиотеками NumPy, Pandas), APL.

3. **CRAN**

Ответ: Comprehensive R Archive Network — основная система распространения пакетов и документации для R.

4. **Плюсы R**

Ответ: богатая статистическая функциональность, качественная визуализация, активное сообщество. Минусы: высокое потребление памяти, относительно низкая скорость выполнения по сравнению с компилируемыми языками.