

# 作业一：Fortran 编程练习

英才 1701 赵鹏威 U201710152

2019 年 9 月 12 日

## 目录

<b>1</b>	<b>引言</b>	<b>2</b>
<b>2</b>	<b>基本的输入输出</b>	<b>2</b>
2.1	任务描述 . . . . .	2
2.2	程序实现 . . . . .	2
2.3	运行时结果 . . . . .	2
<b>3</b>	<b>矩阵乘法</b>	<b>3</b>
3.1	任务描述 . . . . .	3
3.2	程序实现 . . . . .	5
3.3	运行时结果 . . . . .	6
<b>4</b>	<b>排列数和组合数计算</b>	<b>6</b>
4.1	任务描述 . . . . .	6
4.2	程序实现 . . . . .	7
4.3	运行时结果 . . . . .	7

# 1 引言

Fortran 是最早面世的高级编程语言，名字的来源是 Formula Translator，在科学计算方面有很广泛的运用。虽然 Fortran 十分古老，但是早期许多优秀的数值计算程序都是基于 Fortran 代码编写的，因此熟悉 Fortran 语言是十分有帮助的。另外，Fortran 语言具有高效率的优势，这一点与 C/C++ 相似，但是 Fortran 中内置了许多数学函数，使用 Fortran 编写的数值计算程序要相对简洁不少。本次作业的目的就是熟悉 Fortran 语言的基础语法，包括基本的输入输出、文件读写、数组运算等内容。

## 2 基本的输入输出

### 2.1 任务描述

**任务 1.** 编写 *Fortran* 程序实现以下功能：

- 程序能读取用户输入的矩阵；
- 输入矩阵后，程序可以将矩阵保存到一个文本文件中；
- 程序能从文本文件中读取矩阵；
- 程序能按照固定的格式打印矩阵到屏幕上。

一个好的程序要有适当的泛化能力。考虑到用户输入的矩阵大小不确定，因此编写的程序需要适应各种大小的矩阵。在输入矩阵时，用户需要先输入矩阵的大小，也就是有多少行和多少列。

### 2.2 程序实现

程序需要四个功能：输入矩阵、写入文件、读取文件、打印矩阵。”输入矩阵“使用一个 function 实现，这个调用这个 function 后，程序会要求用户输入矩阵的大小，然后按行输入矩阵。这个 function 返回一个二维数组，其保存着用户输入的矩阵。让用户输入矩阵大小的功能可以用动态数组实现，也可以预先定义一个大小较大的二维数组来实现，这里采用后一种方法，也就是预先定义一个  $100 \times 100$  的二维数组，这样用户可以输入比这个大小小的任意矩阵。“写入矩阵”和“读取文件”矩阵分别使用一个 subroutine 实现。写入、读取和打印矩阵时使用格式输出，程序先获取矩阵的大小，然后按照固定的格式写入、读取和打印矩阵。由于，输入矩阵的 function 需要返回一个数组，因此将实现上面三个功能的 function 和 subroutine 封装在同一个 module 中。程序代码见附录 Listing 2。整个程序的工作流程图见图 1。

### 2.3 运行时结果

运行附录 Listing 2 所示代码，运行时结果如图 2。确保程序确实正确地将矩阵写入了 matrix.txt 文件，打开看到确实正常写入，见图 3。

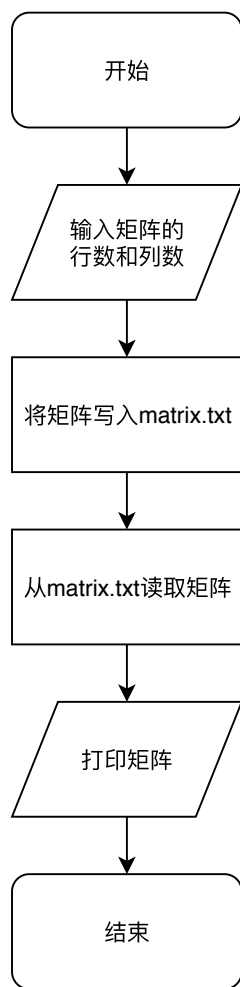


图 1: 任务 1 程序的流程图

### 3 矩阵乘法

#### 3.1 任务描述

**任务 2.** 给定矩阵  $A$  和列向量  $b$ , 使用 *subroutine* 计算  $Ab$  和  $b^T Ab$ . 其中

$$A = \begin{bmatrix} 4 & 2 & 2 & 5 & 8 \\ 2 & 5 & 1 & 3 & 4 \\ 2 & 1 & 6 & 2 & 6 \\ 5 & 3 & 2 & 1 & 3 \\ 8 & 4 & 6 & 3 & 3 \end{bmatrix} \quad b = \begin{bmatrix} 2 \\ 4 \\ 5 \\ 2 \\ 1 \end{bmatrix}$$



## 3.2 程序实现

Fortran 自带矩阵乘法函数 `matmul(matrix1, matrix2)`, 和转置 `transpose(matrix)`. 列向量可以看做是  $n \times 1$  维的矩阵, 因此需要定义两个二维数组来完成这一程序. 由于 subroutine 没有返回值, 需要输出的变量都是要在主程序定义好再作为参数传递给 subroutine 的, 因此为了便于区分输入参数和输出参数, 将输出参数写在输入参数后面. 程序的流程图如图 4 所示. 代码见附录 Listing 3.

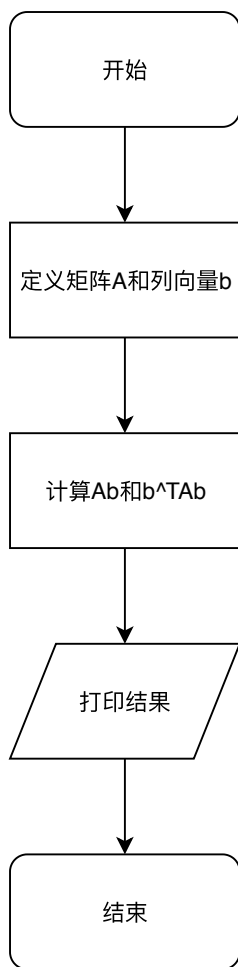


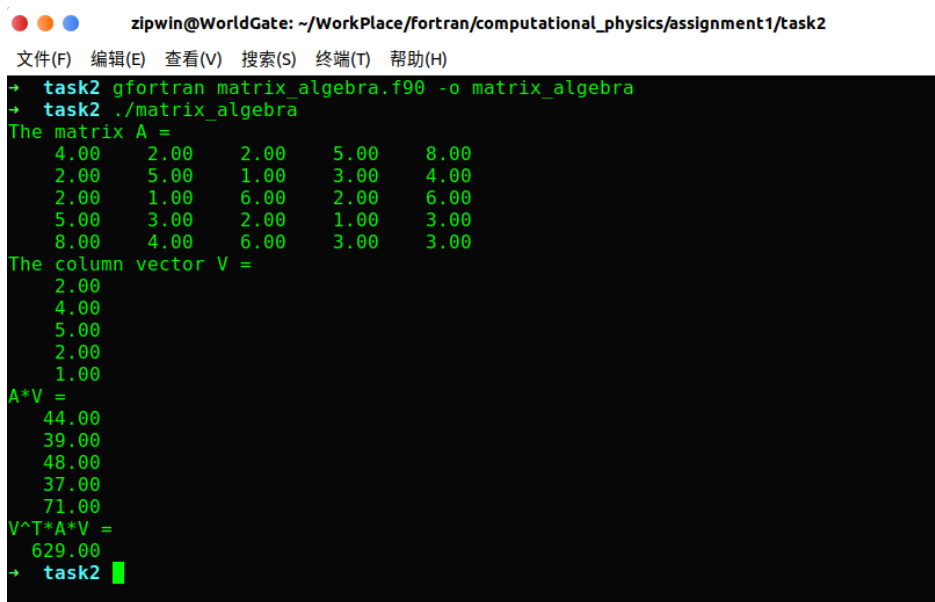
图 4: 任务 2 程序流程图

### 3.3 运行时结果

为了检验程序计算结果的正确性，手动计算得到解析结果如下

$$Ab = \begin{bmatrix} 44 \\ 39 \\ 48 \\ 37 \\ 71 \end{bmatrix} \quad b^T Ab = 629$$

程序运行时结果如图 4 所示. 可见程序计算结果正确.



```
zipwin@WorldGate: ~/WorkPlace/fortran/computational_physics/assignment1/task2
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
→ task2 gfortran matrix_algebra.f90 -o matrix_algebra
→ task2 ./matrix_algebra
The matrix A =
  4.00    2.00    2.00    5.00    8.00
  2.00    5.00    1.00    3.00    4.00
  2.00    1.00    6.00    2.00    6.00
  5.00    3.00    2.00    1.00    3.00
  8.00    4.00    6.00    3.00    3.00
The column vector V =
  2.00
  4.00
  5.00
  2.00
  1.00
A*V =
  44.00
  39.00
  48.00
  37.00
  71.00
V^T*A*V =
  629.00
→ task2
```

图 5: 任务 2 运行时结果

## 4 排列数和组合数计算

### 4.1 任务描述

**任务 3.** 给定  $n = 12, m = 8$ , 编写 *subroutine* 计算排列数  $P_n^m$  和组合数  $C_n^m$

$$P_n^m = \frac{n!}{(n-m)!} \quad C_n^m = \frac{n!}{m!(n-m)!}$$

如果直接计算阶乘，会遇到溢出的错误. 4 字节整数最大能储存  $2^{15} - 1 = 32767$ , 8 字节整数最大能储存  $2^{31} - 1 = 2.1475e + 09$ . 而  $8! = 40320$  已经超过了 4 字节整数的容量，并且  $13! = 6.2270e + 09$  也超过了 8 字节整数的容量. 可见不能直接计算阶乘，而应该在计算之前尽可能约分.

## 4.2 程序实现

为了方便与简洁, 先写了一个名为 `mul2` 的函数, 这个函数可以计算  $n, m$  之间所有整数的积 (包括  $n, m$ ). 这个子程序使用递归实现, 代码如下 (Listing 1)

Listing 1: `mul2`

```
recursive function mul2(n, m) result(r)
    implicit none
    real(8), intent(in) :: n, m
    real(8) :: r

    if (n < m) then
        r = 1.0
    else
        r = n * mul2(n-1, m)
    end if

    return
end function
```

这个函数在  $n$  的值比  $m$  小时输出 1.

按照上面所分析的, 在计算排列数和组合数之前先约分的方法, 可以把排列数的公式改写成

$$P_n^m = \text{mul2}(n, n - m + 1)$$

同样组合数也可以改写为

$$C_n^m = \begin{cases} \frac{\text{mul2}(n, m+1)}{\text{mul2}(n-m, 1)} & m \geq \frac{n}{2} \\ \frac{\text{mul2}(n, n-m+1)}{\text{mul2}(m, 1)} & m < \frac{n}{2} \end{cases}$$

这样就可以尽可能地防止内存溢出. 另外, 计算排列数和组合数的子程序在输入不合法时 ( $m < 0$  或  $n < m$ ) 会输出错误信息. 整个程序的流程图如图 6 所示, 代码见附录 Listing 4.

## 4.3 运行时结果

输入  $n = 12, m = 8$ , 程序计算得到的结果如图 7 所示. 得到的结果是, 排列数为 19958400, 组合数为 495. 利用 Octave 计算的值也分别是 19958400 和 495, 可见程序的计算结果是正确的.

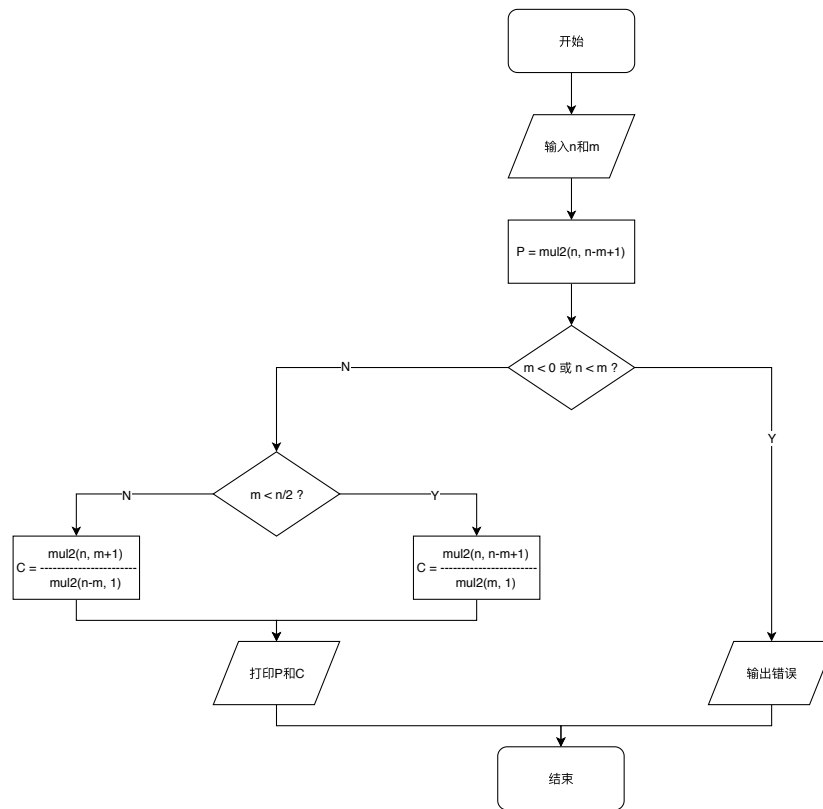


图 6: 任务 3 程序流程图

```

zipwin@WorldGate: ~/WorkPlace/fortran/computational_physics/assignment1/task3
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
+ task3 gfortran perm_and_comb.f90 -o perm_and_comb
+ task3 ./perm_and_comb
n = 12.0 m = 8.0
Permutation: 19958400.0
Combination: 495.0
+ task3 █
  
```

图 7: 任务 3 运行时结果



## 附录

Listing 2: 任务 1 基本输入输出 io.f90

```
module basic_io

contains

function input_matrix(ndim1, ndim2) result(matrix)
    ! Input matrix manually
    implicit none
    integer, intent(in) :: ndim1, ndim2
    integer, parameter :: ndim = 100
    integer :: i
    real*8, dimension(ndim, ndim) :: matrix

    do i = 1, ndim1
        print "(a17,i2,a8)", "Please input the ", i, 'th row: '
        read *, matrix(i, :ndim2)
    end do

    return
end function input_matrix

subroutine write_matrix(matrix, file_name)
    ! Save matrix to file
    implicit none
    real*8, intent(in) :: matrix(:, :)
    character(10), intent(in) :: file_name
    integer :: i
    character(10) :: row_len

    open(file=file_name, unit=10, action="write")
    write(row_len, "(i10)") size(matrix, dim=2)
    do i = 1, size(matrix, dim=1)
        write(unit=10, fmt="(/row_len//\"f8.3)\") matrix(i, :)
    end do
    close(unit=10)

    return
end subroutine write_matrix

subroutine read_matrix(matrix, file_name)
    ! Read matrix from file
```

```

        implicit none
        real*8, intent(out) :: matrix(:, :)
        character(10), intent(in) :: file_name
        integer :: i
        character(10) :: row_len

        open(file=file_name, unit=10, action="read")
        write(row_len, "(i10)") size(matrix, dim=2)
        do i = 1, size(matrix, dim=1)
            read(unit=10, fmt="//row_len//f8.3") matrix(i, :)
        end do
        close(unit=10)

        return
    end subroutine read_matrix

end module basic_io

program io
    use basic_io

    implicit none
    integer :: ndim1, ndim2
    integer, parameter :: ndim = 100
    real*8, dimension(ndim, ndim) :: matrix, matrix2
    integer :: i
    character(10), parameter :: file_name = "matrix.txt"
    ! Save the size of the 2nd dimension of matrix for format print
    character(10) :: row_len

    print "(a37)", "Please input the size of the matrix: "
    read *, ndim1, ndim2

    matrix = input_matrix(ndim1, ndim2)

    print "(15a)", "The input matrix is: "
    ! Translate the size of 2nd dimension to string
    write(row_len, "(i10)") size(matrix(:, ndim1), dim=2)
    do i = 1, size(matrix(:, ndim1), dim=1)
        print "//row_len//f8.3", matrix(i, ndim1)
    end do

    call write_matrix(matrix(:, ndim1), file_name)

```

```
print "(26a, 10a)", "Matrix have been saved in ", file_name

call read_matrix(matrix2(:ndim1, :ndim2), file_name)
print "(17a, 10a)", "Read matrix from ", file_name
do i = 1, size(matrix2(:ndim1, :ndim2), dim=1)
    print ("//row_len//f8.3)", matrix2(i, :ndim2)
end do

end program io
```

Listing 3: 任务 2 矩阵乘法 matrix\_algebra.f90

```

program matrix_algebra
  implicit none
  real(8), dimension(5, 5) :: matrix
  real(8), dimension(5, 1) :: vector
  real(8), dimension(5, 1) :: vector_result
  real(8) :: scalar_result
  integer :: i
  integer :: ndim = 5

  matrix = reshape((/ 4, 2, 2, 5, 8, &
                     2, 5, 1, 3, 4, &
                     2, 1, 6, 2, 6, &
                     5, 3, 2, 1, 3, &
                     8, 4, 6, 3, 3 /), (/ 5, 5 /))
  vector = reshape((/ 2, 4, 5, 2, 1 /), (/ 5, 1 /))

  call calculate(matrix, vector, ndim, vector_result, scalar_result)

  print "(a14)", "The matrix A ="
  print "(5f8.2)", (matrix(i, :), i=1,5)
  print "(a21)", "The column vector V ="
  print "(f8.2)", (vector(i, 1), i=1,5)
  print "(a5)", "A*V ="
  print "(f8.2)", (vector_result(i, 1), i=1,5)
  print "(a9)", "V^T*A*V ="
  print "(f8.2)", scalar_result

end program

subroutine calculate(matrix, vector, ndim, vector_result, scalar_result)
  !-----
  ! Given a matrix A and a vector V, this routine compute A*V and
  ! V^T*A*V, where * is the matrix multiplication.
  !
  ! Arguments:
  !       matrix: a square matrix with shape (ndim, ndim)
  !       vector: a column vector with shape (ndim, 1)
  !       ndim: the size of matrix and vector
  !       vector_result: a column vector with shape (ndim, 1) to save the result A*V
  !       scalar_result: a scalar to save the result V^T*A*V
  !-----
  implicit none
  integer, intent(in) :: ndim

```

```
real(8), dimension(ndim, ndim), intent(in) :: matrix
real(8), dimension(ndim, 1), intent(in) :: vector
real(8), dimension(ndim, 1), intent(out) :: vector_result
real(8), dimension(1, 1) :: fs
real(8), intent(out) :: scalar_result

vector_result = matmul(matrix, vector)
fs = matmul(transpose(vector), vector_result)
scalar_result = fs(1, 1)

return
end subroutine
```

Listing 4: 任务 3 排列数和组合数 perm\_and\_comb.f90

```

program perm_and_comb
  implicit none
  real(8) :: n, m, P, C
  n = 12.0
  m = 8.0

  call permutation(n, m, P)
  call combination(n, m, C)

  print "(a4, f4.1, a6, f4.1)", "n = ", n, " m = ", m
  print "(a12, f16.1)", "Permutation:", P
  print "(a12, f16.1)", "Combination:", C

end program

subroutine permutation(n, m, P)
  implicit none
  real(8), intent(in) :: n, m
  real(8), intent(out) :: P
  real(8), external :: mul2

  if (m < 0 .or. n < m) then
    print "(a)", "Illegal input!"
    return
  else
    P = mul2(n, n - m + 1)
  end if

  return
end subroutine

subroutine combination(n, m, C)
  implicit none
  real(8), intent(in) :: n, m
  real(8) :: P
  real(8), intent(out) :: C
  real(8), external :: mul2

  if (m < 0 .or. n < m) then
    print "(a)", "Illegal input!"
    return
  
```

```

else
    if (m >= n/2) then
        C = mul2(n, m+1) / mul2(n-m, 1.0d0)
    else
        C = mul2(n, n-m+1) / mul2(m, 1.0d0)
    end if
end if

return
end subroutine

recursive function mul2(n, m) result(r)
    implicit none
    real(8), intent(in) :: n, m
    real(8) :: r

    if (n < m) then
        r = 1.0
    else
        r = n * mul2(n-1, m)
    end if

    return
end function

```