

Matrix Operations

David Arnold

March 16, 2003

Abstract

Fortran 90 offers a wealth of array handling features. In this activity, we will examine matrix operations, including addition and subtraction of matrices, multiplying a matrix by a scalar, as well as the multiplication of matrices. We will also compare legacy Fortran 77 coding techniques with the modern techniques provided by Fortran 90.

Introduction

In the programs that follow, we will have need to print the output of numerous array building commands. We will use a module crafted in earlier activities for that purpose. We repeat the module here for completeness.

```
module matrix_ops_mod

  public :: mv_print
  private :: m_print, v_print

  interface mv_print
    module procedure m_print, v_print
  end interface

contains

  subroutine m_print(A)
    real, dimension(:,:), intent(in) :: A
    integer :: m, n, i, j
    m=size(A,dim=1)
    n=size(A,dim=2)
    do i=1,m
      print "(9f10.4)", (A(i,j),j=1,n)
    end do
  end subroutine m_print
end module matrix_ops_mod
```

```

    end do
    print *
end subroutine m_print

subroutine v_print(b)
    real, dimension(:), intent(in) :: b
    integer :: n, i
    n=size(b)
    do i=1, n
        print "(f10.4)", b(i)
    end do
    print *
end subroutine v_print

end module matrix_ops_mod

```

This module will be used with all of the demonstration code in this activity, but will not be repeated again. It is assumed that you will include this module in the appropriate manner in all programs that follow in this activity.

Addition and Subtraction of Arrays

If two arrays are conformable (have the same shape), then they are added element by element. That is, if A and B are arrays having the same shape, then we can form the array $C = A + B$ using a simple rule. The element in the i th row and j th column of array C is found by adding the element in the i th row and j th column of array A to the element in the i th row and j th column of array B . In symbols, we write

$$c_{ij} = a_{ij} + b_{ij}. \quad (1)$$

For example,

$$\begin{pmatrix} 6 & 1 & 2 \\ -2 & 6 & 6 \\ 4 & 7 & -7 \end{pmatrix} = \begin{pmatrix} 1 & 2 & -3 \\ -2 & 1 & 4 \\ 2 & 5 & -6 \end{pmatrix} + \begin{pmatrix} 5 & -1 & 5 \\ 0 & 5 & 2 \\ 2 & 2 & -1 \end{pmatrix}. \quad (2)$$

that each element in the first array is the sum of the elements in the corresponding positions of the second and third arrays.

Legacy fortran code takes advantage of the equation (1) to add the arrays in the example above.

```

program matrix_add

    use matrix_print_mod

```

```

real, dimension(3,3) :: A, B, C
integer :: i, j

A=reshape((/1,-2,2,2,1,5,-3,4,-6/), (/3,3/))
B=reshape((/5,0,2,-1,5,2,5,2,-1/), (/3,3/))

do i=1,3
  do j=1,3
    C(i,j)=A(i,j)+B(i,j)
  end do
end do

print *, "C: "
call mv_print(C)

end program matrix_add

```

The output of the program captures the matrix C of equation (2).

```

C:
  6.0000    1.0000    2.0000
 -2.0000    6.0000    6.0000
  4.0000    7.0000   -7.0000

```

At the heart of the code are the nested `do` loops

```

do i=1,3
  do j=1,3
    C(i,j)=A(i,j)+B(i,j)
  end do
end do

```

which iterate over each row and column, adding $A(i,j)$ and $B(i,j)$ and storing the result in $C(i,j)$. However, the more modern Fortran 90 dialect allows us to replace this doubly nested loop with the single command $C=A+B$, as shown in the following code. Note that we've also removed the declaration `integer :: i, j` from the previous code.

```

program matrix_add

  use matrix_print_mod

  real, dimension(3,3) :: A, B, C

```

```

A=reshape((/1,-2,2,2,1,5,-3,4,-6/),(/3,3/))
B=reshape((/5,0,2,-1,5,2,5,2,-1/),(/3,3/))

C=A+B

print *, "C: "
call mv_print(C)

end program matrix_add

```

Readers should make this adjustment to their code, compile, the run to check that the data stored in the array **C** is identical to that of the previous program.

Exercise 1 *You also subtract two arrays on an element-by-element basis. Thus,*

$$\begin{pmatrix} -1 & 4 & 1 \\ 1 & 8 & 3 \end{pmatrix} = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} - \begin{pmatrix} 2 & -2 & 2 \\ 3 & -3 & 3 \end{pmatrix}. \quad (3)$$

Write a program called `matrix_sub` that subtracts the arrays given in equation (3). Your program should use both of the techniques demonstrated in the previous addition example. Save the file as `assign9/matrix_sub.f90`

Scalar Multiplication

When you multiply a scalar times an array, you are required to multiply the scalar times each element of the array, as shown in the following example.

$$\begin{pmatrix} 5 & 20 \\ 10 & 25 \\ 15 & 30 \end{pmatrix} = 5 \begin{pmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{pmatrix}. \quad (4)$$

Thus, if $B = 5A$, each element in the array B is computed by multiplying the corresponding element in the array A by the scalar 5. In symbols, we write the element in the i th row and j th column of matrix B as follows.

$$b_{ij} = 5a_{ij}. \quad (5)$$

Again, legacy fortran code would implement scalar multiplication with nested `do` loops.

```

program scalar_mult

use matrix_print_mod

```

```

real, dimension(3,2) :: A, B
integer :: i, j

A=reshape((/1,2,3,4,5,6/),(/3,2/))

do i=1,3
  do j=1,2
    B(i,j)=5*A(i,j)
  end do
end do

print *, "B: "
call mv_print(B)

end program scalar_mult

```

When compared with (4), note that the program `scalar_mult` provides the correct result.

```

B:
  5.0000  20.0000
 10.0000  25.0000
 15.0000  30.0000

```

However, Fortran 90 greatly reduces the coding required for the operation of multiplying a matrix by a scalar. You simply replace the nested `do` loops with the single command `B=5*A`.

```

program scalar_mult

  use matrix_print_mod

  real, dimension(3,2) :: A, B

  A=reshape((/1,2,3,4,5,6/),(/3,2/))

  B=5*A

  print *, "B: "
  call mv_print(B)

end program scalar_mult

```

Readers should make this adjustment to their code, recompile, then run and check the result against the result of the previous code sample.

Exercise 2 (A Surprising Result) *Technically, the operation*

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} + 7 \quad (6)$$

is illegal, as adding a scalar to a matrix makes no sense. However, Fortran 90 allows this operation, “broadcasting” the addition of 7 to each element of the matrix. Save the following code as `assign9/scalar_add.f90`.

```
program scalar_add

  use matrix_print_mod

  real, dimension(2,3) :: A, B

  A=reshape((/1, 4, 2, 5, 3, 6/),(/2,3/))

  B=A+7

  print *, "B: "
  call mv_print(B)

end program scalar_add
```

Run the code, noting how the code adds 7 to each entry of matrix A.

Operators and Intrinsic Functions are Array Smart

Many of Fortran 90's builtin operators and intrinsic functions are “array smart.” That is, application of these operators and functions proceeds on an element-by-element basis. For example, one would hope that `A**2` would square each element of the array `A`, as one would equally hope that `sqrt(A)` would take the square root of each element of the array `A`. It is a simple matter to experiment with the array handling features of the operators and intrinsic functions built into Fortran 90. Here is one such experiment.

```
program ops_and_funcs

  use matrix_print_mod

  real, dimension(2,2) :: A, B, C

  A=reshape((/1, 2,3,4/),(/2,2/))
```

```

B=A**2
print *, "B: "
call mv_print(B)

C=sqrt(B)
print *, "C: "
call mv_print(C)

end program ops_and_funcs

```

The output is as expected.

```

B:
  1.0000    9.0000
  4.0000   16.0000

C:
  1.0000    3.0000
  2.0000    4.0000

```

The Dot Product

Although addition and subtraction of matrices is straightforward, as is scalar multiplication, the multiplication of two matrices is downright weird. Before attempting to multiply matrices, we must first master the concept of the *dot product* of two vectors. Whether we use row vectors or column vectors makes no difference, but row vectors take up less space on the typewritten page, so we opt for the former. Let

$$\mathbf{a} = (a_1, a_2, \dots, a_n) \quad \text{and} \quad \mathbf{b} = (b_1, b_2, \dots, b_n)$$

be two row vectors. The dot product of \mathbf{a} and \mathbf{b} is the scalar

$$\mathbf{a} \cdot \mathbf{b} = a_1 b_1 + a_2 b_2 + \dots + a_n b_n. \tag{7}$$

For example, if $\mathbf{a} = (1, 2, 3)$ and $\mathbf{b} = (4, 5, 6)$, then the dot product of \mathbf{a} and \mathbf{b} is

$$\mathbf{a} \cdot \mathbf{b} = (1)(4) + (2)(5) + (3)(6) = 32. \tag{8}$$

Legacy fortran code might take the dot product of the vectors \mathbf{a} and \mathbf{b} as follows.

```

program dot_prod

real, dimension(3) :: a, b
real :: s

```

```

integer :: i

a=(/1,2,3/)
b=(/4,5,6/)

s=0.0
do i=1,3
    s=s+a(i)*b(i)
end do

print *, "The dot product is: "
print *, s

end program dot_prod

```

The output is as expected.

```

The dot product is:
32.0000000

```

However, based on what we've done thus far in this activity, the command `a*b` ought to multiply the vectors on an element-by-element basis, giving a vector whose first entry is $a(1)*b(1)$, whose second entry is $a(2)*b(2)$, and whose third entry is $a(3)*b(3)$. If we then sum the elements of this result with Fortran 90's builtin function `sum`, that would save a lot of coding.

```

program dot_prod

real, dimension(3) :: a, b
a=(/1,2,3/)
b=(/4,5,6/)

print *, "The dot product is: "
print *, sum(a*b)

end program dot_prod

```

Finally, because taking the dot product is a common event, Fortran 90 provides a builtin command for the dot product.

```

program dot_prod

real, dimension(3) :: a, b

```

```

a=(/1,2,3/)
b=(/4,5,6/)

print *, "The dot product is: "
print *, dot_product(a,b)

end program dot_prod

```

Matrix Multiplication

Matrix multiplication is a strange operation. We begin by posting a correct computation.

$$\begin{pmatrix} 7 & 1 & 8 & 7 \\ -1 & 8 & 7 & 8 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 2 \\ 2 & 3 & -1 \end{pmatrix} \begin{pmatrix} 1 & 2 & 3 & -1 \\ 0 & 1 & 1 & 4 \\ 3 & -1 & 2 & 2 \end{pmatrix} \quad (9)$$

Note that this result has the form $C = AB$, where

$$C = \begin{pmatrix} 7 & 1 & 8 & 7 \\ -1 & 8 & 7 & 8 \end{pmatrix}, \quad A = \begin{pmatrix} 1 & 1 & 2 \\ 2 & 3 & -1 \end{pmatrix}, \quad \text{and} \quad B = \begin{pmatrix} 1 & 2 & 3 & -1 \\ 0 & 1 & 1 & 4 \\ 3 & -1 & 2 & 2 \end{pmatrix}.$$

Note that the shape of matrix A is 2×3 . The shape of matrix B is 3×4 . Can you explain why the following code will not work?¹

```

program matrix_mult

  use matrix_print_mod

  real, dimension(2,3) :: A
  real, dimension(3,4) :: B
  real, dimension(2,4) :: C

  A=reshape((/1,2,1,3,2,-1/), (/2,3/))
  B=reshape((/1,0,3,2,1,-1,3,1,2,-1,4,2/), (/3,4/))

```

¹The command $A*B$, being an array operation, should multiply the entries in the corresponding positions of A and B . However, because the arrays A and B do not have the same shape, this is not possible. If you try to compile the program, the compiler will present an error message essentially stating that the shapes are different and the multiplication is not possible. It is important to note that even if the shapes of the matrices A and B were identical, the command $C=A*B$ is not going to produce the desired result, as this command computes the ij th entry of C by multiplying the corresponding entries in arrays A and B . This is not the type of multiplication desired in this instance.

```

C=A*B

print *, "C: "
call mv_print(C)

end program matrix_mult

```

So, how to proceed? Let's examine the computation in equation (9). The entry in the first row and column of matrix C is a 7. Note that if you take the dot product of the first row of A with the first column of B , the result is 7.

$$(1, 1, 2) \cdot (1, 0, 3) = (1)(1) + (1)(0) + (2)(3) = 7$$

Let's try that again. The entry in the second row and third column of matrix C is also a 7. If we take the dot product of the second row of A with the third column of B , the result is 7.

$$(2, 3, -1) \cdot (3, 1, 2) = (2)(3) + (3)(1) + (-1)(2) = 7$$

In general, the element in the i th row and j th column of matrix C is computed by taking the dot product of the i th row of A with the j th column of B . Note that these dot products are only possible if the number of columns of A equals the number of rows of B .

Now, suppose that matrix A has m rows and n columns.

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{i1} & a_{i2} & \cdots & a_{in} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}$$

Suppose that matrix B has n rows and p columns.

$$B = \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1j} & \cdots & b_{1p} \\ b_{21} & b_{22} & \cdots & b_{2j} & \cdots & b_{2p} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{nj} & \cdots & b_{np} \end{pmatrix}$$

Note that the number of columns of matrix A equals the number of rows of matrix B , so the multiplication is possible.² To compute the ij th entry of matrix C , we take the dot

²The dimensions of matrix A are $m \times n$ and the dimensions of matrix B are $n \times p$. Note that the inner dimensions of A and B "match up," which says that the number of columns of A and the number of rows of B both number n , so the multiplication is possible. The outer dimensions of A and B give the dimension of the answer, in this case $m \times p$.

product of the i th row of matrix A with the j th column of matrix B .

$$\begin{aligned}c_{ij} &= (a_{i1}, a_{i2}, \dots, a_{in}) \cdot (b_{1j}, b_{2j}, \dots, b_{nj}) \\ &= a_{i1}b_{1j} + a_{i2}b_{2j} + \dots + a_{in}b_{nj}.\end{aligned}$$

Using summation notation, this can be written much more concisely.

$$c_{ij} = \sum_{k=1}^n a_{ik}b_{kj} \tag{10}$$

The Program Assignment

You are to write a program (save your program as `assign9/matrix_mult.f90`) that will compute the matrix product given in equation (9). Initialize matrices A and B in your main program, then write a *function* that will compute the product of two *arbitrary* matrices. Of course, you must place the function in your program module, along with the routines for printing arrays given at the beginning of this article.

- Your function should begin with the following line.

```
function matrix_multiply(A,B) result (C)
```

- Quoting from *The F Programming Language*, by Michael Metcalf and John Reid, Oxford Science Publications, “Unless the function result is a pointer, the bounds must be explicit expressions and they are evaluated on entry to the function.” Because we haven’t addressed pointers as yet, I recommend the following line.

```
real, dimension(size(A,dim=1),size(B,dim=2)) :: C
```

- Check the dimensions of matrix A and B to see if the multiplication is possible (the number of columns of A must equal the number of rows of B). If not, print an appropriate error message and exit the function.
- Use nested do loops and equation (10) to evaluate each ij th entry of matrix C .

Upon return to the main program, print the result from the function `matrix_multiply`; that is, print the product of matrices A and B .

Finally, as a check, use the builtin function `matmul` to print the product of A and B . This is easily accomplished with the following commands.

```
print *, "C: "  
print *, matmul(A,B)
```

Extra Credit

Here's an opportunity to expand upon your knowledge while earning a little extra credit.

1. Write a second program (`assign9/matrix_mult2.f90`) where the function definition uses array sections and the builtin `sum` command to compute the ij th entry of the matrix product.
2. Write a third program (`assign9/matrix_mult3.f90`) that utilizes the builtin `dot_product` function to compute the ij th entry of the matrix in your function. Of course, array sections are also a must in this instance.

The Grading Rubric

The following rules will apply for this program, after which we will discuss and adjust the rubric during class.

1. (30 points) Will be awarded for adequate comments. Comments should include:
 - (a) A description of the program's purpose.
 - (b) Name, date, version or revision number.
 - (c) A complete dictionary of all variables and parameters used in the program.
 - (d) Interprogram comments should proceed any code snippets explained by the comments. These should be adequately sprinkled throughout your code.
2. (50 points) Will be awarded if the program works and does what it was asked to do.
3. (10 points) Will be awarded for good program style. This includes good indentation practices, etc.
4. (10 points) Will be awarded for creativity and extra effort. Did you just do the bare minimum? Or did you stretch and reach a little higher? Did you put something cute or clever into your program that nobody else seemed to think of?

Penalties

Each program that is assigned during the term will have a due date. On that date, the program must be on the instructor's desk before the start of class. Penalties will be assessed as follows.

1. (10 points) There will be a 10 point deduction for any program that is handed in after the class has begun.

2. (20 points) There will be a 20 point deduction per class period. That is, if you hand the program in one class period late, there is an automatic 20 point deduction. Two class periods warrants a 40 point deduction, etc. To be clear, if the program is in the instructor's hands before the beginning of the next class, that is a 20 point deduction. If the program is in the instructor's hands before the start of the second class period past the due date, that is a 40 point deduction, etc.

Managing Files and Folders

Each of you has been given personal space on the sci-math server to store your work. Typically, this space is mapped to the drive letter H. If you open the Windows Explorer (the file manager, not the internet browser), you can see that the drive letter has been mapped to your login name.

In this folder, create a new folder called **fortran**. Note that you must **never** use spaces in filenames. In the Windows operating system, filenames are not case-sensitive, which is exactly opposite what happens in Unix and Linux, where filenames are case-sensitive.

In `/home/username/fortran`, create another folder called **assign9**. It is in this folder that you are to place the source code and executables for this current project. Please name your program `matrix_mult.f90`. When you receive your next project, create a new folder called `/home/username/fortran/assign10` to hold that project, etc

If you work at home, I still want you to place copies of your work in the space reserved for you on our system. Simply copy your home files onto a floppy disk and bring them with you to school.³ Copy your file to the folder `/home/username/fortran/assign9`. Do not copy executables to the school drive. Rather, copy your source, then compile on the school computer to produce the executable.

If everyone follows these simple rules, I can easily access your work from my office machine for purposes of assigning a grade.

Caveat

On this project, if you stop by my office with hardcopy of your program before the due date of this assignment, I will give a quick glance and critique of your source code. Somewhat like receiving a grade on a draft before submitting your final draft for assessment.

³If you use Putty on your home system, you can access your account on our server. That way you don't have to carry files back and forth on floppies.