Chapter 2

Matrices and Calculations

As mentioned in the earlier chapter, you can have lists of numbers as well as of letters. These list of numbers can be either one-dimensional, in which case they are called a vector, or an array. When they are two, three, or more-dimensional they are called a matrix.

```
>>mat1=[1 54 3; 2 1 5; 7 9 0; 0 1 0]
>>mat2=[1 54 3
  2 1 5
  7 9 0
  0 1 0]
```

Take a look at mat1 and mat2. As you can see there is more than one way of entering a matrix. `mat1` and `mat2` were entered differently, but both have 4 rows and 3 columns. When entering matrices a semi-colon is the equivalent of a new line. You can find the size of matrices using the command "size.

```
>>size(mat1)
```

For a two dimensional matrix the first value in `size` is the number of rows. The second value of `size` is the number of columns.

Now try:

```
>>vect1=[1 2 4 6 3]
>>vect2=vect1'
```

Vectors can be tall instead of long, `'` (the little symbol below the double quote on your keyboard) is a ***transpose***, and allows you to swop rows and columns.
Remember there is also the command `whos` which will tell you the size of all your variables.
Use `whos` to look at the size of mat1, mat2, vect1 and vect2.

```
>>whos
```

What does `mat1'` look like?

You can perform various calculations on matrices and arrays.

You can add a single number (also called a scalar) to a vector

```
>>vect1+3
```

You can subtract

```
>> vect2-3
```

You can add a vector onto itself

```
>>vect1+vect1
```

You can also add two vectors as long as they are the same size. You can't add vect1 and vect2 together since they are different sizes.

```
>>vect1+vect2
??? Error using ==> plus
Matrix dimensions must agree.
```

The reason you got an error is that you can't multiply something with 1 row and 4 columns, with something else that has 4 rows and 1 columns.

```
>>vect3=[2 4 8 12 6]
>>vect1+vect3
```

These two vectors are the same size, so you can add them together, So far it's easy.

Point-wise multiplication and division

But when you want to add two vectors or matrices to each other you need to know that there are two sorts of multiplication and two sorts of division. The simple kind of multiplication and division is called array multiplication. You do this using the symbols `.*` and `./`

```
>>vect1.*3
>>mat1.*0.5
>>vect1./2
```

You can also multiply or divide one vector by another, element by element. Each element in the first vector is multiplied by the corresponding element in the second vector. Note that the vectors must be the same shape

```
>>vect1.*vect3
>>vect1./vect3
>>vect3./vect1
```

Watch out for the transpose in the next 2 examples:

```
>>vect1.*vect2'
>>vect1.*vect3'
??? Error using ==> plus
Matrix dimensions must agree.
```

You will get this error a lot. What an error like this means is that the two vectors or matrices that you are trying to perform an operation on (such as multiplication) aren't the right size or shape to do what you are doing. Lots of operations are fussy about making sure the sizes of the vectors or matrices are consistent. So when you get this error the first thing you should do is check the `size` of all the variables that you are tying to manipulate, and try to work out whether they might be different sizes. Often it's the case that simply transposing one of the variables is all you need to do. This is an important thing to understand, so don't rush through the examples above. Make sure you understand what's going on.

```
>> mat1./vect1
??? Error using ==> rdivide
Matrix dimensions must agree.
```

This is not going to happen for you, regardless of how you transpose mat1 and vect1. vect1 and mat1 will never be the same size, no matter what you do.

So, with point-wise multiplication and division (which is what you will be doing most of the time) you can:

1) multiply or divide a matrix or vector by a single number (either `*` or `.*` is correct to use)

2) multiply or divide a matrix or vector by a matrix/vector that is the same size (make sure you use `.*`).

It's best to get into the habit of using .* all the time, unless you are specifically using matrix multiplication (which you will only use rarely)

The second kind of multiplication and division is matrix multiplication and matrix division. There are two types of matrix multiplication.

Matrix Multiplication

Outer product

Z=X*Y – the first vector is tall & thin, the second vector is short & fat**.**

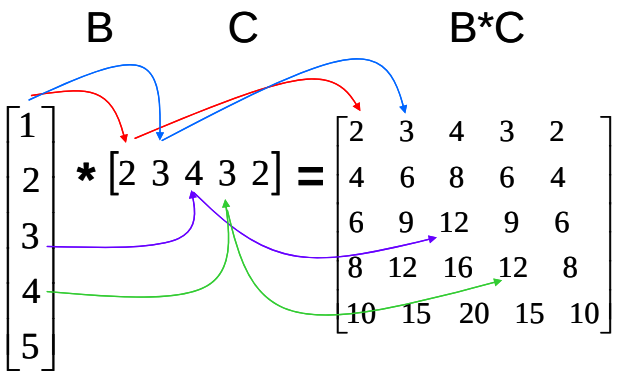
Actually matrix multiplication isn't used very often in experimental programming, though vision scientists often find it useful for making filters for image processing and so on.

```
>B = [1; 2; 3; 4; 5]
B =
     1
     2
     3
     4
     5
```

```
>C = [2 3 4 3 2]
C =
     2     3     4     3     2


>whos
  Name      Size                    Bytes  Class

  B         5x1                        40  double array
  C         1x5                        40  double array
```

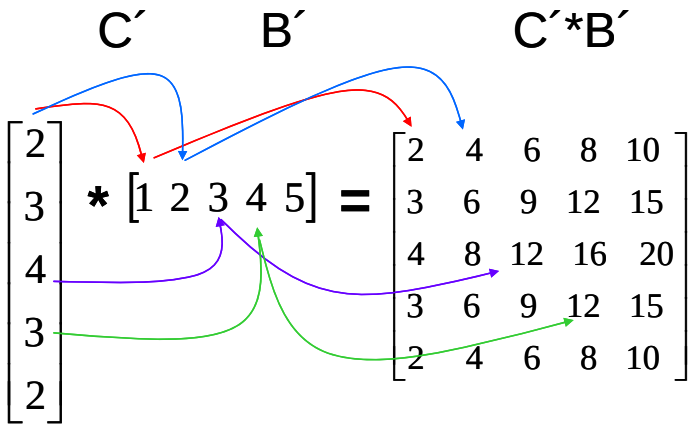
The first vector is tall and thin, and the second vector is short and fat. This means we are calculating the ***outer product*** of the two vectors:

```
>B*C

ans =
     2     3     4     3     2
     4     6     8     6     4
     6     9    12     9     6
     8    12    16    12     8
    10    15    20    15    10
```



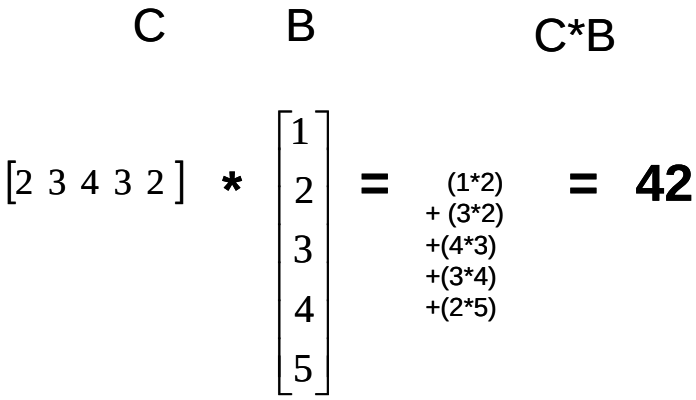
If we transpose both B and C and reverse the multiplication order, then the number of rows in C' again matches the number of columns in B', and we can calculate another outer product, which is in fact the transpose of the previous outer product:

```
> C'*B'

ans =

     2     4     6     8    10
     3     6     9    12    15
     4     8    12    16    20
     3     6     9    12    15
     2     4     6     8    10
```

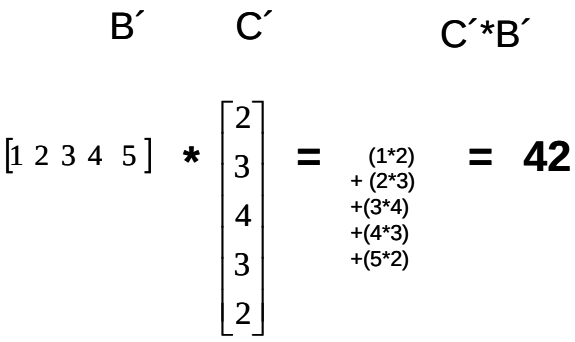


Inner product

Here the first vector is short & fat, and the second vector is tall & thin.
>C*B



With inner products you can again transpose both vectors and swop the order of the multiplication. This time you get the same answer – 42.



3d-4d matrices & more matrix manipulations

Matrices can be 3, 4 or more dimensions, though some commands won't work for matrices with more than 2 or 3 dimensions

```
>mat(1, :,:) =[2 4 3 ; 5 6 7];
>mat(2, :, :)=[1 3 2; 6 1 2];
>size(mat)
>mat
```

Sometimes it's useful to convert back and forth between vectors and matrices.

```
>mat=[1 2 3; 4 5 6]
mat =

     1     2     3
     4     5     6
>vect=mat(:)
vect =

     1
     4
     2
     5
     3
     6
```

Strings the matrix mat out a as a vector. Note that the vector first lists all the rows in the first column, then lists all the rows in the second column, and so on …

`>ind=sub2ind(size(mat), 2, 2)`
calculates the index in vect corresponding to the 2nd row and 2rd column of mat. So:

```
>vect(ind)
>mat(2,2)
```

Should give you the same answer. Check out `ind2sub` and work out what it does.

Program 2 - Calculations.m

Create a new m-file and save it in the MatlabClass/Misc folder as "Calculations.m".

You'll notice in the code below that some of the lines are rather long. When you are writing code and a line is longer than your page you can break it using three dots ... Without those three dots Matlab will treat each part of the line as a separate command and give you an error message.

```
1      % Calculations.m
2      %
3      % Example program that carries out a series of
4      % calculations on two numbers
5      %
6      % written by IF and GMB 4/2005
7
8      clear;
9      n1=input('choose the first number ... ');
10     n2=input('choose the second number ... ');
11
12     % spit responses out onto the command line
13     disp(['first number is ', num2str(n1)])
14     disp(['second number is ', num2str(n2)])
15     disp([num2str(n1),'+', num2str(n2), ...
16      '=', num2str(n1+n2)]);
17     disp([num2str(n1), '-', num2str(n2), ...
```

```
18      '=', num2str(n1-n2)]);
19      disp([num2str(n1), '*', num2str(n2), ...
20       '=', num2str(n1.*n2)]);
21      disp([num2str(n1), '/', num2str(n2), ...
22       '=', num2str(n1./n2)]);
23      if (round(n1)==n1)
24         disp(['n1 is an integer, n2 rounded = ', ...
25             num2str(round(n2))]);
26      end
27      if (round(n2)==n2)
28         disp(['n2 is an integer, n1 rounded = ', ...
29             num2str(round(n1))]);
30      end;
31      disp([num2str(n1), ' to the power of ', ...
32         num2str(n2), ' is ' ,num2str(n1.^n2)]);
33      disp(['the smallest number of ',num2str(n1), ...
34          ' and ',num2str(n2), ' is ', num2str(min(n1, n2))]);
35      if n1>0 & n2>0
36         disp('Both n1 and n2 are greater than zero');
37      end
```

You can run Calculations.m in two ways. One is to type the name of the program into the command window:

```
>Calculations
```

Alternatively you can go to the menu bar and choose Debug-Run.

Lines 9-10. `input` prints a string onto the command window and waits for a response. The expected response needs by default to be a number. Try running the program a second time, and this time type a letter in (a *character*) rather than a number – you will get an error:

```
??? Error using ==> input
Undefined function or variable 'r'.
```

It's possible to get input to accept characters and strings instead of numbers by explicitly telling it that the input won't be a number. Try this at the command window:

```
> yourname=input('What is your name? ', 's')
```

Here the `'s'` tells `input` that the input will be a string instead of a number.

Lines 12-21. num2str converts the number n1 into a string. So:

```
['first number is ', num2str(n1)]
```

is one long string, which is then displayed by disp.

Line 23. The == checks to see if round(n1) is the same number as n1. Try

```
>3==3
```

This will give you a 1 since it is true

```
>3==3.4
```

This will give you a 0 because it is untrue.
It's important to remember the difference between == and =,
The single = tells Matlab to make the variable x be 3.

```
>x=3

x =

     3
```

The double == asks Matlab to check whether or not x is equal to 3, and returns and answer of 1 if x *does* equal 3 and an answer of 0 otherwise.

```
>x=3.4;
```

```
>x==3
ans =

     0
```

Remember how we explained that numbers could either be integers (e.g. 3) or be doubles (3.12). On Line 22 we check to see whether n1 and n2 are round numbers by seeing if the rounded version of the number is the same as the number itself. If this is confusing try:

```
>round(3.15)
>round(3)
>round(3.14)==3
>round(3.14)==4
```

Remember that the way `==` works is that you get an answer of 1 if the numbers on either side of the `==` are the same, and a 0 otherwise.

`if` loops work in the following way: The if statement checks the condition that follows the if. For this if statement the condition is (round(n1)==n1). If the output of that condition is a 0 (the condition is false) then the statement between the `if` and the `end` (line 23-24) is **not** executed. If the output of the condition is anything but 0 (the condition is true), then the statement after the `if` is executed. Traditionally a true condition is represented by a 1, but in Matlab an `if` loop will be executed unless the condition following the `if` results in a 0.

Line 35 The & operator is used when you only want to carry out a loop only when more than one condition are both true. & checks to see if *both* the statement before and after the & are true. I.e. `Condition 1 & Condition 2` gives you a 1 if both conditions are true, and gives you a 0 otherwise. Try the following:

```
> 3>0 & 2>0
> 3>0 & -1>0
> -1>0 & 3>0
```

Matrix manipulation

There are other ways of entering matrices besides entering every number. Create a new program called MakingMatrices.m and save it in your MatlabClass/Misc folder

```
1      % MakingMatrices.m
2      %
3      % this program provides examples of cunning ways that you can  %
4      create matrices.
5      % written by IF 3/2007
6
7      mat1=zeros(10, 8);
8
9
10     mat2=mat1;
11     mat2(:, 3)=1;
12
13     mat3=mat1;
14     mat3(4, :)=1;
15
16     for i=1:5
17         mat5(i,:)=[0 0 1 1 2 2 3 3]+i;
18     end
19
20     for i=1:5
```

```
21          mat6(:, i)= [0 0 1 1 2 2 3 3]+i;
22      end
23
24      mat7=zeros(7);
25      for i=1:7
26          mat7(i, i)=i;
27      end
28
29      mat8=zeros(5);
30      for i=1:5
31          for j=1:5
32              mat8(i, j)=i+j;
33          end
34      end
35
36      mat9=zeros(5);
37      for i=1:5
38          for j=1:5
39              mat9(i, j)=((i-1)*5)+j;
40          end
41      end
42
43      mat10=zeros(6);
44      mat10(2:5, 2:3)=1;
45
```

Line 7. The command `zeros(10, 8)` creates a matrix containing all zeros with 10 rows and 8 columns. Try the command `ones.` Remember that the first dimension in a matrix is the rows, and the second dimension is the columns.
Line 10. Here you are setting all the rows in the third column to be 1. The colon basically means to include the entire row. Another way of writing this command would be `mat2(1:10, 3)=1;`
Line 13. Here you are setting all the columns in the fourth row to be 1. The colon now means to include the entire column. Another way of writing this command would be `mat3(4, 1:8)=1;`
Lines 15-17. Here, we gradually go through five rows of a matrix filling the columns with a vector that depends on which row it is.
Lines 20-22. Here, we gradually go through five columns of a matrix filling the rows with a vector that depends on which column it is.
Lines 24-27. Here we create a matrix that is 7 by 7 zeros (it is a little weirdness of this command that the convention is that simply telling it be 7 makes it a square matrix). Then we go through each row of the matrix, and replace the place in the matrix that is the i-th row and i-th column with the number i.
Lines 29-34. This is what is called a ***nested loop***. Basically it is one (or many) `if` or `for` statements inside another one. So we basically go down each row, and for each row we work our way through the columns, and replace that position in the matrix with the value of `i+j`.
Lines 36-41. This another ***nested loop***. Except in this case we put in the value `((i-1)*5)+j.` Think about this – when we are on the first row, the columns will be labeled as 1,2,3,4,5, when we are on the second row the columns will be labeled as 6,7,8,9,10, and so on …

Being able to do these sorts of manipulations is one of the keys to being able to write good code in Matlab, so take your time and make sure you understand how these examples work.

EXERCISES

1) Add two lines to Calculations.m displaying the square roots of n1 and n2.
Use `>>help sqrt` to get you started

2) Add a line displaying the max of n1 and n2 using the command `max`
3) find the cos and the sin of vect1 and vect2, and display them in degrees (not radians). The command `pi` will make this more accurate.
4) Try to add lines to Calculations.m that use the following operators:

```
>>a<=b
```

This gives you a 1 if a is less than or equal to b.

```
>>a | b
```

This gives you a 1 if either a or b are nonzero, 0 otherwise

```
>>a~=b
```

This gives you a 1 if a is not equal to b, and gives you a 0 otherwise. E.g.

```
>>3~=4
>>3~=3
```

5) Add code to create the following matrices to `MakingMatrices.m`
Emat1 =

```
   1    6   11   16   21
   2    7   12   17   22
   3    8   13   18   23
   4    9   14   19   24
   5   10   15   20   25
```

```
Emat2 =

     1     0     1     0     1     0     1     0
     1     0     1     0     1     0     1     0
     1     0     1     0     1     0     1     0
     1     0     1     0     1     0     1     0
     1     0     1     0     1     0     1     0
     1     0     1     0     1     0     1     0
     1     0     1     0     1     0     1     0
     1     0     1     0     1     0     1     0
```

(You actually know two ways to do this, remember the transpose command earlier in the chapter? See if you can create this matrix both ways)

```
Emat3 =

     1     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
     0     0     1     0     0     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     1     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     1     0
     0     0     0     0     0     0     0     0
```

```
Emat4 =

     0     1     2     3     4
     1     2     3     4     5
     2     3     4     5     6
     3     4     5     6     7
     4     5     6     7     8
```

```
Emat5 =

     1     2     3     4     5
     2     4     6     8    10
     3     6     9    12    15
     4     8    12    16    20
     5    10    15    20    25

Emat6 =

     1     6    11    16    21
     2     7    12    17    22
     3     8    13    18    23
     4     9    14    19    24
     5    10    15    20    25
```

Hints for the next two matrices
1) Remember the command ones
2) Remember that as well as indexing a single place in a matrix, `mat(1, 1),` or indexing a whole column, `mat(1, :)` you can also index a part of a column or row, using things like `mat(2, 1:3)`

```
Emat7 =

     1     1     1     1     1     1
     0     0     0     0     0     0
     0     0     0     0     0     0
     0     0     0     0     0     0
     1     1     1     1     1     1
     1     1     1     1     1     1

Emat8 =

     1     1     1     1     1     1
     1     0     0     0     1     1
     1     0     0     0     1     1
     1     0     0     0     1     1
     1     1     1     1     1     1
     1     1     1     1     1     1
```