Crashcourse in Matlab - Part I: Variables and Arrays

Variables

Varibles are symbols that we can define in matlab and typically asign a value to.

```
a = 5e2;
b = 3;  % scientific notation, means 3 10^2
```

Now the variables are defined and in the Workspace. The semicolon, simply prevents Matlab from printing. The variable b is defined in scientific notation, $b = 3 \cdot 10^2$.

Once the variables are defined we can work with them.

```
a*a
```

```
ans = 250000
```

```
c = a+b
```

```
c = 503
```

```
d = (c/a)^b
```

```
d = 1.0181
```

Variable names can be complicated and contain numbers and underscores.

```
AgeStudent1 = 18.3;
AgeStudent2 = 20.7;
AgeStudent3 = 21.1;
AgeStudent4 = 20.4;

mean_student_age = (AgeStudent1+AgeStudent2+AgeStudent3+AgeStudent4)/4
```

```
mean_student_age = 20.1250
```

Note that variables can have different type. That means that they can be numbers of different kinds (integers, floats (single and double precision), booleans) or strings (letters). In classical programming languages one has to be carefule when mixing different number, but Matlab takes care of most of this.

In this class we won't use strings very much, but it is good to be aware that a string can contain numbers. Typically we use strings for file names when saving simulation data. Such a variable might look like this

```
file_name = 'time_134_dys.mat'
file_name =
'time_134_dys.mat'
```

Arrays

In the above case it makes sense to store all the ages together in an array.

```
StudentAge = [AgeStudent1 AgeStudent2 AgeStudent3 AgeStudent4]
```

```
StudentAge =
   18.3000   20.7000   21.1000   20.4000
```

Here the baracket indicates the beginning and the end of the array. In this case the the array has the from of a row vector - if you want you can put commas between the entries but the result is the same

```
StudentAge_row = [AgeStudent1, AgeStudent2, AgeStudent3, AgeStudent4]
```

```
StudentAge_row =
   18.3000   20.7000   21.1000   20.4000
```

If you separate entries by semi-colons you get a column vector

```
StudentAge_col = [AgeStudent1; AgeStudent2; AgeStudent3; AgeStudent4]
```

```
StudentAge_col =
   18.3000
   20.7000
   21.1000
   20.4000
```

You can go between row and column vectors with the transpose

```
StudentAge_col'
```

```
ans =
   18.3000   20.7000   21.1000   20.4000
```

Arrays can also be matrices. In this class we typically denote matrices by capital letters

```
A = [1 2 3;4 5 6; 7 8 9]
```

```
A =
     1     2     3
     4     5     6
     7     8     9
```

Matrices do not need to be square!

```
B = [1 2; 3 4; 5 6; 7 8; 9 10]
```

```
B =
     1     2
     3     4
     5     6
     7     8
     9    10
```

and they can also be transposed

```
B'
```

```
ans =
     1     3     5     7     9
     2     4     6     8    10
```

Composing arrays

You can form larger arrays by composing existing matices and vectors, as long as the dimensions match

```
I = [1 0 0; 0 1 0; 0 0 1]; % Identity matrix
e = [1 1 1];

C = [I I;I I]
```

```
C =
     1     0     0     1     0     0
     0     1     0     0     1     0
     0     0     1     0     0     1
     1     0     0     1     0     0
     0     1     0     0     1     0
     0     0     1     0     0     1
```

```
D = [I e'; e 0]
```

```
D =
     1     0     0     1
     0     1     0     1
     0     0     1     1
     1     1     1     0
```

Indexing Arrays

It is very common that we need to access elements of an array. The elements of arrays are numbered starting from 1, not 0 as in most programming languages (C++). The age of the third student is accessed a follows

```
StudentAge_row(3)
```

```
ans = 21.1000
```

```
StudentAge_col(3)
```

```
ans = 21.1000
```

You can access multiple entries at the same time as follows

```
StudentAge_row
```

```
StudentAge_row =
   18.3000   20.7000   21.1000   20.4000
```

```
StudentAge_row([1 2 4])
```

```
ans =
   18.3000   20.7000   20.4000
```

```
ind = [3 2 4];
StudentAge_row(ind)
```

```
ans =
   21.1000   20.7000   20.4000
```

Matrices can be indexed in two ways:

1. With indices, the first corresponding to the row the second to the column.
2. With a single index which assumes the matrix is strected to a vector by stacking the columns

```
B
```

```
B =
     1     2
     3     4
     5     6
     7     8
     9    10
```

```
B(3,2)
```

```
ans = 6
```

```
B(8)
```

```
ans = 6
```

Colon operator

In Matlab the colon ":" is a powerful operator that will make your live easier. Yoiu can use it to gererate arrays of integers

```
d = [4:9]
```

```
d =
     4     5     6     7     8     9
```

```
e = [0:5:20]
```

```
e =
     0     5    10    15    20
```

The colon is commonly used in indexing to access several elements that are next to each other

```
StudentAge_row(1:3)
```

```
ans =
   18.3000   20.7000   21.1000
```

If you use the colon by itseld you get the entire array

```
StudentAge_row
```

```
StudentAge_row =
   18.3000   20.7000   21.1000   20.4000
```

```
StudentAge_row(:)
```

```
ans =
   18.3000
   20.7000
   21.1000
   20.4000
```

However, it turns row vectors into colum vectors. If the colon is used to index a matrix it also turns it into a column vector by stacking the columns

```
B(:)
```

```
ans =
     1
     3
     5
     7
     9
     2
     4
     6
     8
    10
```

Most commonly the colon is used to index columns or rows of matrices. For example the first row of C is simply

```
C
```

```
C =
     1     0     0     1     0     0
     0     1     0     0     1     0
     0     0     1     0     0     1
     1     0     0     1     0     0
     0     1     0     0     1     0
     0     0     1     0     0     1
```

```
C(1,:)
```

```
ans =
     1     0     0     1     0     0
```

The 2nd to 4th columns of C are

```
C(3:6,2:4)
```

```
ans =
     0     1     0
     0     0     1
     1     0     0
     0     1     0
```