Overview

1. Recap
2. Basic programming elements
   - Arrays, vectors and matrices
   - Control statements
3. Scripts
4. Good programming practice
   - Comments: readability
   - Constants: readability and reusability
5. Summary

Recap

- Two types of numbers:
  - integers (exact)
  - floats (finite precision)
- Rounding errors can occur in floating point arithmetic, especially when subtracting large numbers
- Variables are used to store numbers (and other quantities)

Numbers in MatLab

MATLAB is designed to calculate, so assumes everything is a double-precision floating-point unless you tell it otherwise:

```
>> clear
>> a=2;
>> whos
  Name        Size                Bytes  Class     Attributes

  a           1x1                     8  double
```

Usually this is ok ... but beware!

Rounding errors

- There is no point trying to get answers to greater precision than machine precision
- Critical when subtracting large numbers!

Example

Solution of $ax^2 + bx + c = 0$,

$$x = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

can be useless if $b^2 \gg ac$

- We often use increased precision when summing up lots of numbers

Arrays, vectors and matrices

In physics applications we are often interested in arrays (rows) of numbers, for example

- the position of an object as a function of time
- the value of the electric field as a function of x and y
- the temperature, pressure and wind speed in different places
- the wave function of a quantum mechanical particle

Discretisation

Computers know nothing about the continuum

— we must tell it to compute values at discrete positions or times x_i, t_i

$$\vec{r}(t) \to \vec{r}(t_i) \qquad \to (x_i, y_i, z_i)$$

$$\vec{E}(x, y) \to \vec{E}(x_i, y_j) \qquad \to (E_x^{ij}, E_y^{ij}, E_z^{ij})$$

etc

Vectors in MatLab

We want to operate on vectors, matrices and higher-dimensional arrays
Many computer languages are not very good at this
But this is exactly that MatLab is designed for!
You can create a vector by just typing in a row of numbers

```
>> v = [0 2 5 6]
v =
     0 2 5 6
```

or, more usefully, by supplying start, end and increment:

```
>> w = [0:0.1:1]
w =
  Columns 1 through 7
   0    0.1000   0.2000   0.3000   0.4000   0.5000   0.6000
  Columns 8 through 11
   0.7000   0.8000   0.9000   1.0000
```

This discretises the interval [0,1] into points with separation 0.1

Vector operations

The i'th element of a vector `v` is addressed as `v(i)`

```
>> v(1)
ans =
        0
>> v(3)
ans =
        5
```

MatLab also allows you to access a subset of a vector, or extend it:

```
>> w(2:4)
ans =
      0.1000    0.2000    0.3000
>> v = [v 8]
v =
     0   2   5   6 8
```

Vector arithmetic

MatLab allows you to do arithmetic directly on vectors:

```
>> u = 2*v
u =
     0   4   10   12   16
>> y = sin(w)
y =
  Columns 1 through 7
   0    0.0998   0.1987   0.2955   0.3894   0.4794   0.5646
  Columns 8 through 11
   0.6442   0.7174   0.7833   0.8415
```

BUT ...

Matrices

MatLab is really at MATrix LABoratory!
— its default mode is to apply the rules of linear algebra to everyting
It treats a vector as a $1 \times n$ matrix, so will try to use rules of matrix multiplication if you tell it to multiply two vectors.

Element-by-element operations

Perform element-by-element multiplication, division or exponentiation by preceding the operator with a . (dot): `.*`, `./`, `.^`

Matrix operations

transpose	A'
inverse	`inv(A)`
solve `Ax=b`	`x=A\b`
solve `yA=c`	`y=c/A`
eigenvalues	`eig(A)`

And more!!

Control statements

Most algorithms consist of repeating certain tasks either

1. a certain number of times
   (eg for all elements in an array/vector), or
2. until certain conditions are fulfilled
   (until we have converged to the right answer)

1. is a `for` loop:

```
for i=1:n
    ... (statements/operations) ...
end
```

2. is a `while` loop

```
while ( conditions )
    ... stuff ...
end
```

Logical expressions

The conditions in while loops (and in if statements) are made up of logical expressions

They can also be used to create logical arrays

- Relational operators: `< <= > >= == ~=`
- Logical operators: `& && | || ~ xor` (and, or, not, exclusive-or)
- Use `&& ||` in control statements, `& |` in array operations
- Remember precedence rules: use (brackets) when unsure!

Loops

for loop

```
for ctr = start[:step]:stop
   ...
end
```

This will execute the statements inside the loop for each value of `ctr`.
`start, step, stop` must be integers. `step` defaults to 1.
Beware of changing the value of `ctr` inside the loop

Example

```
for n=1:10
    sq(n) = n^2;
end
```

This puts the first 10 square numbers into the vector `sq`

It is often faster, better to use vector arithmetic than for loops in MatLab!

Loops

while loop

```
while logical expression
  ...
end
```

Make sure that

- the expression can be sensibly evaluated at start
- the loop will terminate at some point

Example

```
x=2;
while (x > 0.05)
 x=x/2
end
```

Loops

for and while

The for loop is equivalent to a while loop:

```
ctr = start;
while ctr <= stop
  ...
  ctr = ctr+step;
end
```

getting out of loops

There are two commands to 'get out' of loops:

`continue`	skips to the end (start) of loop
`break`	exits completely from the loop

Scripts

You don't want to type in the same series of commands every time you want to perform a particular task. Instead, you want to run the same commands again and again.

Scripts and functions

The simplest way to do this is with a script:
Just write your commands into a file, then run that file!
A more powerful way is to write a function $\longrightarrow$ next week!

Example

The following code will calculate the first 20 Fibonacci numbers:

```
fib = ones(1,20);
for i=3:20
 fib(i) = fib(i-2) + fib(i-1);
end
```

If we want to use the Fibonacci numbers at some later time, we can type this into a script `fibonacci.m` and run it then.

Comments

Example

Look at `fibonacci.m` again:

```
fib = ones(1,20)
for i=3:20
  fib(i) = fib(i-2) + fib(i-1);
end
```

Assume someone else has given it to you, and you do not know what it does. Does it return all the first 20 Fibonacci numbers, or just the 20-th one? How easy is it to see?

Comments

You can (and should!) put in comments in the code to tell yourself and others what it does:

```
% fibonacci: returns a vector fig containing the 20 first
% Fibonacci numbers
f = ones(1,20)
% The first two Fibonacci numbers are both 1
for i=3:20
  % The i'th number is the sum of the two previous ones
  f(i) = f(i-2) + f(i-1);
end
```

In MatLab, the comment at the top also gives you the help text:

```
>> help fibonacci
fibonacci: returns a vector fib containing the 20 first
Fibonacci numbers
```

Readability

Example

```
x=0:0.01:6;
s=0*x;
for i=2:601
 s(i) = s(i-1) + 0.01*cos(x(i));
end
```

What does this do?

Putting in comments can really help readability:

```
% Script to compute the integral of cos(x) for 0<=x<=6
x=0:0.01:6;
s=0*x;      % initialise vector s to zero
for i=2:601
   % use rectangle formula to integrate
   s(i) = s(i-1) + 0.01*cos(x(i));
end
```

Readability and reusability

The integration script is still not very readable:

1. Where does the number 601 come from?
2. Where does the 0.01 in the rectangle formula come from?

Also, it can be hard to modify if you just want to compute something slightly different:

1. Use a coarser or finer grid for integrating
2. Change the start and end points

We should give all our numerical constants names to remedy both problems!

Constants

```
% Script to compute the integral of cos(x)
% for xmin<=x<=xmax with grid spacing dx
xmin=0;
xmax=6;
dx=0.01;
x = xmin:dx:xmax;
n = length(x);
s = zeros(1,n);    % or s=0*x; - initialise to zero
for i=2:n
   % use rectangle formula to integrate
   s(i) = s(i-1) + dx*cos(x(i));
end
```

Summary

- Vectors and arrays lie at the heart of computational physics
  - MatLab is well suited to operating on vectors and matrices!
- Loops allow you to perform repetetive tasks
- Use functions or scripts for tasks to be repeated
- Use comments to make functions and scripts readable
- Remember to initialise your variables!
- Define constants upfront rather than using raw numbers