Overview

1. Recap
2. Loops and control statements
3. Good programming practice
4. Functions
5. Good programming practice
   - Comments: readability
   - Constants: readability and reusability
   - Error checking
6. Summary

Variables and types

Variables and types

A variable is anything with a name and a value.
In MatLab they come into being when given a value.
They can be removed with the command `clear`:

```
>> clear a;
```

deletes the variable a from memory.

Variables and types

Variables and types

A variable is anything with a name and a value.
In MatLab they come into being when given a value.
They can be removed with the command `clear`:

```
>> clear a;
```

deletes the variable a from memory.
Variables can be of different types:

- integer [int8, int16, int32, int64]
- floating-point [single or double]
- logical
- character strings
- user-defined types or classes! (we will not use these)

Variables and types

Variables and types

A variable is anything with a name and a value.
In MatLab they come into being when given a value.
They can be removed with the command `clear`:

```
>> clear a;
```

deletes the variable a from memory.
Variables can be of different types:

- integer [int8, int16, int32, int64]
- floating-point [single or double]
- logical
- character strings
- user-defined types or classes! (we will not use these)

Type checking is not very important in MatLab!

Arrays

You can put variables into arrays (vectors, matrices etc).
These can represent for example

- time series of data
- fluid density or electric field at different positions
- all the students in the class!

Indices and addressing

Elements of arrays are addressed by their indices i, j etc:
`v(3)` is the third element of the vector v
`rho(i)` [$= \rho_i$] is the i-th element of the vector rho
`A(i,j)` [$= A_{ij}$] is the j-th element of the i-th row of the 2d array A

Arrays

You can put variables into arrays (vectors, matrices etc).
These can represent for example

- time series of data
- fluid density or electric field at different positions
- all the students in the class!

Indices and addressing

Elements of arrays are addressed by their indices i, j etc:
`v(3)` is the third element of the vector v
`rho(i)` [$= \rho_i$] is the i-th element of the vector rho
`A(i,j)` [$= A_{ij}$] is the j-th element of the i-th row of the 2d array A

You can assign a value to a single element of an array:
`v(3) = b;` puts the value of `b` into the third element of the vector `v`

Operators and expressions

Arithmetic operations

Arithmetic operators are `+ - * / \ ^`

- Remember precedence rules: use (brackets) when unsure!
- MatLab wants to use matrix arithmetic
  $\longrightarrow$ use `.* ./ .^` to perform element-by-element operations

Operators and expressions

Arithmetic operations

Arithmetic operators are `+ - * / \ ^`

- Remember precedence rules: use (brackets) when unsure!
- MatLab wants to use matrix arithmetic
  $\longrightarrow$ use `.* ./ .^` to perform element-by-element operations

Logical expressions

Logical expressions are used in control statements (if and while), and 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!

Control statements

if statement

```
if   logical expression
  ...
elseif   another logical expression
  ...
else
  ...
end
```

Control statements

if statement

```
if ( age<18 )
  disp('I am a child')
elseif ( age>65 )
  disp('I am retired')
else
  disp('I am an adult')
end
```

Control statements

if statement

```
if   logical expression
  ...
elseif   another logical expression
  ...
else
  ...
end
```

You can have as many `elseif` statements as you like,
and 0 or 1 `else` statements.

Loops

for loop

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

`start, step, stop` must be integers. `step` defaults to 1.

Beware of changing the value of `ctr` inside the loop

Loops

for loop

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

`start, step, stop` must be integers. `step` defaults to 1.

Beware of changing the value of `ctr` inside the loop

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

Loops

for loop

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

`start, step, stop` must be integers. `step` defaults to 1.

Beware of changing the value of `ctr` inside the loop

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

Example

```
dx=0.01;
x=0:dx:6; n=length(x);
s=zeros(1,n);
for i=2:n
  s(i) = s(i-1) + dx*cos(x(i));
end
```

Loops

for loop

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

`start, step, stop` must be integers. `step` defaults to 1.

Beware of changing the value of `ctr` inside the loop

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

Example

```
dx=0.01;
x=0:dx:6; n=length(x);
s=zeros(1,n);
for i=2:n
  s(i) = s(i-1) + dx*cos(x(i));
end
```

What does this do?

Loops

while loop

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

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
```

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

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 $\longrightarrow$ later!

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 $\longrightarrow$ later!

Comments and readability

- Put in comments to explain what you are doing
- Give names to all your numerical constants

Fibonacci numbers

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
```

Fibonacci numbers

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.

Fibonacci numbers

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.But what if

1. we need the 30th Fibonacci number?
2. we are using the variable name `fib` for something else?

Fibonacci numbers

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.But what if

1. we need the 30th Fibonacci number?
2. we are using the variable name `fib` for something else?

Write a function

```
function f = fibonacci(n)
f = ones(1,n)
for i=3:n
  f(i) = f(i-2) + f(i-1);
end
```

Functions

Functions

Functions allow you to easily change input parameters, and to assign your results to variables of your choice

```
function Y = myfunc( var1, var2, var3)
  ... stuff ...
  Y = result;
end
```

Fibonacci function

Function fibonacci.m

```
function f = fibonacci(n)
f = ones(1,n)
for i=3:n
  f(i) = f(i-2) + f(i-1);
end
```

If we now type

```
>> fibonacci(23)
```

we get the first 23 Fibonacci numbers.

We can also type

```
>> fib = fibonacci(30);
```

and the first 30 Fibonacci numbers will be stored in the vector `fib`.

Local variables

Function fibonacci.m

```
function f = fibonacci(n)
f = ones(1,n)
for i=3:n
  f(i) = f(i-2) + f(i-1);
end
```

Local variables

Try typing

```
>> clear        [clears all variables]
>> f5=fibonacci(5);
>> f
```

What happens?

Local variables

Function fibonacci.m

```
function f = fibonacci(n)
f = ones(1,n)
for i=3:n
  f(i) = f(i-2) + f(i-1);
end
```

Local variables

Try typing

```
>> clear        [clears all variables]
>> f5=fibonacci(5);
>> f
```

What happens?

The variable f is invisible outside the function!

Local variables

Function fibonacci.m

```
function f = fibonacci(n)
f = ones(1,n)
for i=3:n
  f(i) = f(i-2) + f(i-1);
end
```

Local variables

Try typing

```
>> clear        [clears all variables]
>> f5=fibonacci(5);
>> f
```

What happens?

The variable f is invisible outside the function!

All variables inside functions are local, so you never have to worry about them messing up anything outside!

Comments

Example

Look at `fibonacci.m` again:

```
function f = fibonacci(n)
f = ones(1,n)
for i=3:n
  f(i) = f(i-2) + f(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 n Fibonacci numbers, or just the n-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:

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

Comments

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

```
% f = fibonacci(n): returns a vector containing the n first
% Fibonacci numbers
function f = fibonacci(n)
f = ones(1,n)
% The first two Fibonacci numbers are both 1
for i=3:n
  % 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
f = fibonacci(n): returns a vector containing the n 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?

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

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=1:n
   % use rectangle formula to integrate
   s(i) = s(i-1) + dx*cos(x(i));
end
```

Error checking

Look at the Fibonacci function again:

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

What happens if

1. n < 1?
2. n is not an integer?

Error checking

Look at the Fibonacci function again:

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

What happens if

1. n < 1?
2. n is not an integer?

Garbage in, garbage out

Error messages

Solution: put in error checks in your code:

```
% f = fibonacci(n): returns the n first Fibonacci numbers
function f = fibonacci(n)
if (n<1)
  error('Illegal input value n<1')
end
if (floor(n)!=n)
  error('Non-integer value for n')
end
  ...
```

If we try an illegal input value, this will be caught:

```
>> fibonacci(1.2)
??? error using ==> fibonacci at 7
Non-integer value for n
```

Common errors

- array index out of range
- infinite loop
- illegal input values
- division by zero
- division by a tiny number
- numerical underflow or overflow
- input/output errors

Summary

- Use functions or scripts for tasks to be repeated

Summary

- Use functions or scripts for tasks to be repeated
- Use comments to make functions and scripts readable

Summary

- 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