Computational Projects

Lecture 8 : programming advice, etc

Dr Rob Jack, DAMTP

http://www.maths.cam.ac.uk/undergrad/catam/part-ia-lectures

What makes a good program?

A good program should do a specific job, reliably and efficiently.

ideally, it should be flexible: it should work in general cases, not only for simple cases. It should provide output that is easy to understand.

For any given task, there are would be many possible programs that can accomplish this

> *When you write a program, think of it primarily as a work of literature. You're trying to write something that human beings are going to read. Don't think of it primarily as something a computer is going to follow. The more effective you are at making your program readable, the more effective it's going to be: You'll understand it today, you'll understand it next week, and your successors who are going to maintain and modify it will understand it.*
>
> Donald Knuth

> *When you write a **proof**, think of it primarily as a work of literature. You're trying to write something that human beings are going to read. [...] The more effective you are at making your proof comprehensible, the more effective it's going to be.*

> *When you write an **answer to an exam question**, think of it primarily as a work of literature. You're trying to write something that human beings are going to read. [...] The more easily the examiner can see what you meant, the more effective it's going to be.*

> *When you write a **CATAM report**, think of it primarily as a work of literature. You're trying to write something that human beings are going to read. [...] The more easily the assessor can see what you meant, the more effective it's going to be.*

From first lecture

Like a mathematical proof, an algorithm needs to be unambiguous: the computer will execute the algorithm that you give, not the one that you "obviously" meant to write(!)

Examiners [and CATAM project assessors] will only mark the work that you submit : we can't give credit for things that you "obviously" understood but never wrote down

(this is particularly relevant when giving partial credit)

Standard programming advice

Give names to your variables and functions that make your program easy to read

> In the old days there were reasons to use short variable names and people often used single letters, `i,j,k` etc
>
> *... no longer a good idea...*

Include plenty of comments

Use functions sensibly, avoid copy-pasting similar sections of code...

Don't be tempted to do "something clever" unless you are really sure that it makes your program more efficient *(... keep it simple)*

something clever: Recursion

A recursive function is a function whose definition includes the function itself

```
function [ y ] = recursiveFactorial( x )
%recursiveFactorial compute x factorial

  % check x>=0, else we can get stuck in an infinite loop
  if x < 0
      error('factorials of positive numbers only')
  end

  if x <= 1
      y = 1;
  else
      y = x * recursiveFactorial( x-1 );
  end

end
```

compute the factorial of x-1

"to understand recursion, you must first understand recursion"

something clever(?): Recursion

... keep it simple

```
function [ y ] = simpleFactorial( x )
%simpleFactorial compute x factorial

  if x < 0
      error('factorials of positive numbers only')
  end

  y=1;
  for k=1:x
      y = y*k;
  end

end
```

... easier to understand, just as efficient

(Also, recursive functions are often harder to debug and test; they can be useful in some situations but caution is advised...)

Debugging

As you practise programming, you will find that sometimes your programs don't work as you hoped...

The process of fixing the problems ("bugs") is called debugging

Most people agree that debugging is the hardest and most frustrating aspect of programming...

> The best way to help with debugging is:
> **write programs with fewer bugs in the first place**
>
> ... but there are other useful strategies...

Avoiding bugs

Keep it simple

Add comments to your program *as you write it*, don't think of this as something to do at the end (... once it's working)

Break the problem into functions and test each function separately (make sure that you know what will happen if a function gets an unexpected input... consider adding a check)

Sometimes you can solve the problem by breaking it into several parts and combining them together at the end...

... when combining, it will help a lot if you have used sensible variable names and you have included comments.

Debugging

There is no perfect debugging method but a good plan is:

Try to guess what might be wrong, and *test this hypothesis*

A classical method is to print out the values of variables that might be wrong, and check them

Eg for LU decomposition, maybe you want to find a simple example, print out all the matrices $A^{(k)}$, and check that the first k rows and columns are indeed full of zeros

Once you find one thing that is wrong, try to trace things backwards: where did the first mistake appear?

Debugging

... in a similar vein:

If you are doing something complicated, try to gradually make it simpler, and see if the problem survives

... if you deal with matrices, is the problem still there if you use a 3x3 matrix, where you can check everything by hand?

... if you do something iteratively, is the problem already there in the first iteration? At what iteration does it appear?

... can you tie down the problem to one specific function that is getting 'correct' input but producing 'incorrect' output?
(try to make a *minimal reproducible example* of the problem)

minimal reproducible example

```
function [ y ] = simpleFactorial( x )
%simpleFactorial : compute x factorial

... write the function here ...
... this function is self-contained
    (no other function calls inside) ...

end

display(simpleFactorial(4))
```

If this code produces any output other than 24 then it would be a minimal reproducible example (MRE) of a bug

-- It is clear what the function should do (and that it fails).

-- Anyone can check this

-- No outside information is needed to understand the problem

Breakpoints, stepping, etc

MATLAB can help you with debugging

The workspace window will tell you the current values of all variables

You can use "breakpoints" to make scripts stop in the middle, so that you can check the values of variables

You can "step" through programs, one line at a time, to see what is happening

In other programming languages you can often get this functionality through an IDE (integrated development environment)

Debugging summary

If you can come up with easy-to-read programs that do computations in a clear and logical way then they will be easier to develop, easier to use, and easier to modify

It helps if you **test everything** as you are writing it

This sounds easy but the only way to be good at it is to practise!

Bugs are just mistakes, we all make them, the trick is to find them early, when they are easy to fix...

Programming languages

You can do the CATAM projects in any programming language

The examples in this course used MATLAB

could have used python instead...

... or Julia , C++ , LISP, java , ...

What are the advantages / disadvantages ?

MATLAB vs python

MATLAB and python are similar in many ways

(as long as we use python's `numpy` and `scipy` libraries)

Easy and flexible to use, efficient with matrix operations and other math tasks

(solving ODEs, root finding, finding minima of functions...)

python is free, MATLAB is sold by a private company
(we have free access)

both are reasonably efficient but neither runs super-fast

what about Julia ?

MATLAB and python are very well-established
... easy to use, lots of help online

a slightly newer development is called Julia

still growing, faster / more efficient than python
... also easy to use, but a bit less help available so far

Prof Stephen Eglen's resources on using Julia for CATAM :

https://sje30.github.io/catam-julia/

the old-fashioned way...

you might hear about "lower-level" languages like C , java, ...

in these languages, you have to tell the computer more precisely what it should do

(for example, you have to specify whether each variable is an integer or a real number or a vector or a matrix, contrary to MATLAB)

in general : this can be good for experts and for difficult problems,

probably not needed for CATAM etc

... however, you do need to keep track of your variable types

(getting this wrong is a very common way to generate bugs)

functions as black boxes

"Modern" languages are much more flexible for adapting code

Functions are very important for this :
generally-applicable tools that can be re-used.

... might come from different sources ...

To re-use a function, we often don't care how it works
(it is a "black box") https://en.wikipedia.org/wiki/Black_box

Do we need to know what happens "inside the black box" ?
(Usually, no. But we have seen that sometimes it matters...)

the End

-- Good luck with exams (now) and CATAM projects (next year) !

-- Questions / comments / suggestions ?