Optimisation and Operations Research

Lecture 4: Algorithm Design in Matlab

Matthew Roughan
<matthew.roughan@adelaide.edu.au>
http://www.maths.adelaide.edu.au/matthew.roughan/notes/OORII/

School of Mathematical Sciences,
University of Adelaide

July 30, 2019

Section 1

Good Code

Fast Code

- Design a good algorithm
  - examples: multiplication by Schoolbook method v Karutsuba's method
  - first step is being able to calculate the complexity
- Good code:
  - implement the algorithm (not something a bit like it)
  - good data structures
    - ★ efficient to access
    - ★ don't add extra time looking things up
  - careful implementation
    - ★ optimise for common case, *e.g.*, in nested if statements, should only nest the uncommon case
    - ★ avoid things that a language is bad at (in Matlab, for loops are bad, in other languages recursion might be slow)

But fast isn't everything, always

Good code

Isn't just about speed

- Often your code just has to be "fast enough"
- But you want
  - to make it easy to debug
  - to be able to reuse all or parts of it
  - others to be able to use it or modify it
- To do that we have to learn a little about style
  - this is not a programming course, so we can't spend nearly enough time on it, but a few hints follow

Readability

Always code as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live.

Rick Osborne

Any fool can write code that a computer can understand. Good programmers write code that humans can understand.

Martin Fowler

Any code of your own that you haven't looked at for six or more months might as well have been written by someone else.

Eagleson's law

Thus spake the master programmer: "A well-written program is its own heaven; a poorly-written program is its own hell."

The Tao Of Programming

Readability

Readability makes code easier to debug and maintain

- Good variable names [BF12, Kim10]
- Good comments [BF12, Eva10, Hen10]
  - clear and helpful
  - concise (don't comment $x = x + 1$)
  - remember the compiler doesn't check comments!
- Layout matters [Fre10, Kim10]
  - we're good at pattern recognition, and layout helps
  - modern editors usually make it easy
  - *e.g.*,
    - ★ indentation
    - ★ whitespace (in formulae)
    - ★ bracket placement

DRY = Don't Repeat Yourself

Don't Repeat Yourself

- reuse tested code
- if a value is used more than once create a variable
- if a routine is used more than once create a procedure or function

Having it in one place means it will be consistent, and easy to change

- the "single source of truth" principle

Go To Statement Considered Harmful [Dij68]

http://xkcd.com/292/

Similar problems:

- global variables

Global variables

We didn't really go into details of goto's (they aren't a big deal in MATLAB), but a similar bad idea is global variables

- These are variables that are visible anywhere in your program
- They are convenient, because they make it easy to convey information from one place to another
- They are a bad idea for the same reasons as gotos
  - increase inter-dependence
  - create multiple (different) pathways to a single point in the code
  - make it harder to read the code linearly

If you use global variables in code you hand up, you **will** loose marks, and may be given **zero**!!!

Section 2

MATLAB and good code

Floating point

- Computers represent integers $\mathbb{Z}$ in binary (base-2)
- We want to represent any *real* number $\mathbb{R}$
  - there is no finite way to do this perfectly
  - the real numbers are *uncountable*
- Common approach is to use *floating point* numbers
  - We store in exponential form: a mantissa and exponent
  - *e.g.*,

$$1.245 = 1245 \times 10^{-3}$$

    so we store the two integers 1245 and -3 (and the sign)
  - There are many standards for floating point
    - ★ most common is the IEEE 754 Standard
    - ★ MATLAB uses double-precision floating point from this
    - ★ each variable takes 64 bits, or 8 bytes
    - ★ this standard includes NaN and Inf and -Inf
- Quick reading:
  `http://www.ee.columbia.edu/~marios/matlab/Fall96Cleve.pdf`

Floating point

- Most important thing to remember is that these are *approximations*!
  - there will be numerical errors in ALL of your calculations!
  - you can never assume a number is exact
- Must be careful
  - don't test for equality

Example

Replace

```
if x == y
```

with

```
if abs(x-y) < epsilon
```

  - take care when dividing by something that is near zero
    - ★ errors may have a large impact on output

Floating point weirdness

Just a short list of weirdnesses caused by floating point

- $+0$ and -0 are different numbers
- 0.1 cannot be exactly represented in a float.
  If you don't believe me, try `fprintf('%.40f\n', 0.1)`
- A change in the order of operations (that should be equivalent) can change the result.
- All floats can be exactly represented by a finite decimal number, but it might take more than 100 digits.

https://randomascii.wordpress.com/2012/04/05/floating-point-complexities/ https://randomascii.wordpress.com/2012/01/11/tricks-with-the-floating-point-format/

Floating point tricks and tips

- Never ask "is x equal to y?", always ask "is x close to y?"
  - *i.e.*, replace

    `if (x == y)` with `if (abs(x-y) < epsilon)`
  - what should ε be? That's actually a tricky question
    https://randomascii.wordpress.com/2012/02/25/comparing-floating-point-numbers-2012-edition/
  - most of the time, for us, it isn't too hard – just pick a reasonable small number, *e.g.*, 10^{-10}
  - but note, lots of my code allows ε as a parameter
- Always be aware of numerical errors, *e.g.*, try in MATLAB

  $$\sin(pi)$$

  The result should be 0. It isn't.
- Floating point arithmetic used to be much slower than integer arithmetic, but these days most CPUs have a dedicated FPU
  - still, for really, really fast code

Vectorisation

- Vectorisation is a common trick to speed up MATLAB code
- MATLAB is slow at some things, and fast at others
  - slow at "loops"
  - fast at vector and matrix operations
- This is good for us
  - lots of what we do in optimisation is Linear Algebra
  - it vectorises nicely
- Its a MATLAB trick, not a fundamental way of speed up code
  - you are still performing the same number of operations
- Readability
  - sometimes good: *e.g.*, for Linear algebra
  - sometimes bad: *e.g.*, sometimes you twist your code in knots to write it this way

Modularisation [Gar10]

We break code into blocks or modules in order to

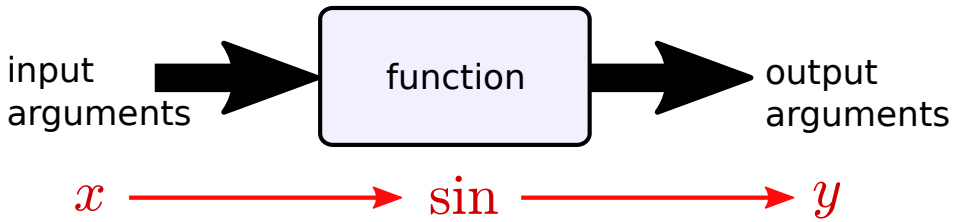
- *hide information*: a user should be able to use the code block, without knowing how it is implemented, and in fact that might change in the future.
- create *clean interfaces:* *i.e.*, limit inter-dependencies or *coupling* between components that make programs complex to understand and debug.
- create *cohesion:*, *i.e.*, related processes are grouped, and *separate concerns*, *e.g.*, so that multiple people can each code up their part of a project.

Modularisation in MATLAB

Functions

MATLAB's main mechanism for modularisation is the `function`

- It's similar to the idea of a function in Mathematics
  - *e.g.*, $y = \sin(x)$



- Except the inputs and outputs can be anything
  - *e.g.*, strings
- And in a program a function can have "side effects"
  - though we don't like them

Modularisation in MATLAB

Arguments

- The term *argument* comes from the inputs to a function in Mathematics
  - but it is also used to describe the outputs in MATLAB
- Sometimes inputs are called *parameters*
- Sometimes outputs are called *return values*
- Sometimes a distinction is made between the value you prescribe when you define the function (a parameter) and the value actually passed to the function at run time (an argument), but this distinction isn't going to be used here.

Modularisation in MATLAB

Pass by "clever"

Arguments are variables that are "passed" to the function

- *pass by value*: the variable is copied into a new variable inside the function
- *pass by reference:* a reference or pointer to the variable is copied, and the function uses this to work with the original
- They each have advantages and disadvantages
  - MATLAB tries to do the best of both
  - it passes by reference, unless a copy is needed
  - *i.e.*, pass by "clever"

http://au.mathworks.com/matlabcentral/answers/96960-does-matlab-pass-parameters-using-call-by-value-or-call

Modularisation in MATLAB

Scope

Variables defined inside the function are different from those outside

- inside the function, you don't have access to the whole world
  - only input arguments (and globals)
- but you can create your own variables
  - if you create X inside the function, it is a different variable to the X outside
- the places you can see a particular variable are called its *scope*

Modularisation in MATLAB

Putting it together

```
function [out1, out2,...  ,outN] = ...
                function_name(inp1, inp2,...  ,inpM)
% comments to be displayed by help
statements
out1 = expression1;
out2 = expression2;
...
outN = expression2;
```

The function would typically be called by typing

```
[o1, o2, ... , oN] = function_name(i1, i2, ... ,iM);
```

Modularisation in MATLAB

Nesting, recursion, ...

There are lots of precise rules for setting up functions (see Practical 2), *e.g.*,

- Functions can have other function defined inside them (*i.e.*, nested)
- A function can call itself (*i.e.*, recursion)
- A function can drop out before its finished by "throwing an error"
- Comment lines at the start of a function definition are used in MATLAB's `help` operation.

Have a go at all this in Practical 2.

Section 3

Debugging

Debugging

It's not at all important to get it right the first time. It's vitally important to get it right the last time.

Andrew Hunt and David Thomas

Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it.

Brian Kernighan

Beware of bugs in the above code; I have only proved it correct, not tried it.

Donald Knuth

There are two ways to write error-free programs; only the third one works.

Epigrams in Programming, 40., Alan Perlis

Debugging

Debugging is the hardest part of coding

- Error messages are often arcane and confusing
  - but read them carefully – they contain a lot of information
  - Google keywords from the message – see what other people did about it
- MATLAB's IDE (Integrated Development Environment) has tools
  - When you have an error, it will point you in the direction of the error
    - its important to realise that the actual error may be elsewhere though
  - You can trace through code step by step to see what is going on
    http://au.mathworks.com/help/matlab/matlab_prog/debugging-process-and-features.html
- Create simple "test cases" to make sure your code is working, and debug it when it isn't
  - this is actually a big and *important* topic in its own right
- You often have to "Sherlock" around your code a bit to find an error

Debugging

See the handout for more details

Takeaways

- Write good code
- MATLAB functions are a critical feature, you need to be able to use
- Debugging, hard but inevitable

Further reading I


Dustin Boswell and Trevor Foucher, *The art of readable code*, O'Reilly, 2012.


Edsger W. Dijkstra, *Go to statement considered harmful*, Communications of the ACM **11** (1968), no. 3, 147–148.


Cal Evans, *97 things every programmer should know*, ch. A Comment on Comments, pp. 32–33, O'Reilly, 2010.


Steve Freeman, *97 things every programmer should know*, ch. Code Layout Matters, pp. 26–27, O'Reilly, 2010.


Edward Garson, *97 things every programmer should know*, ch. Apply Functional Programming Principles, pp. 4–5, O'Reilly, 2010.


Kevlin Henney, *97 things every programmer should know*, ch. Comment Only What the Code Cannot Say, pp. 34–35, O'Reilly, 2010.

Further reading II

Yechiel Kimchi, *97 things every programmer should know*, ch. Coding with Reason, pp. 30–31, O'Reilly, 2010.