Research Tools and Methods for the Mathematical Science

Lecture 4: Writing Software for the Mathematical Sciences

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

School of Mathematical Sciences,
University of Adelaide

March 30, 2015

Intro

Proof by cumbersome notation: Best done with access to at least four alphabets and special symbols.

> I look forward to the day when a Pulitzer Prize will be given for the best computer program of the year.
>
> *Donald E. Knuth [KLR89, p.21]*

> Grand Master Turing once dreamed that he was a machine. When he awoke he exclaimed: "I don't know whether I am Turing dreaming that I am a machine, or a machine dreaming that I am Turing!"
>
> *The Tao Of Programming*

Mathematicians MUST code

You all need to learn to code (write programs)

Mathematicians MUST code

You all need to learn to code (write programs)

Yes you!

Mathematicians MUST code

You all need to learn to code: examples

- your research
  - these days much mathematical research is done using computers
  - simulation, computational algebra, computer proofs
  - computers may even be your research (e.g., my work)
- producing papers
  - figures are often done using code, e.g., Matlab
  - LaTeX'ing can be enhanced if you know some coding
- dealing with data
  - cleaning it
  - visualising it
- automating your everyday tasks
  - analysing marks from your students
  - script to filter/clean bibtex
  - processing LaTeX in non-standard ways (e.g., creating outlines)

Coding is like

Coding is like writing [English],

- it makes your ideas concrete, and in doing so irons out the bugs
- it has a grammar, but just getting the grammar right won't make you good at coding
- you will find endless advice, but you must find your own path

If all you have is a hammer

- Use the right tool for the job
  - don't need the full weight of C++ to do a little data cleaning
  - scripting language for a major piece of software?
- Use the right approach for the job
  - software engineering styles are adapted to the task
  - small and agile approaches for small, rapidly changing code
    - ★ e.g., research code
  - big, slow approaches for production systems that have to be reliable

Programming paradigms

imperative: series of statements that change state

- **procedural** specify steps (procedures) to reach desired state: so its imperative, but with more structure.
- e.g.

functional: avoid mutable data: program state is result of execution of functions

- e.g.

object-oriented: organise around "objects" with data and methods

- e.g.

event driven: flow is determined by events

- used for GUI programing, games, real-time systems, ...

declarative: specify computation, without flow

- e.g. spreadsheets, markup

Script vs Compile

- interpreted vs compiled languages
  - interpreter coverts program to executable line by line
  - compiler passes through whole program multiple times to create executable
  - interpreted may be slower (in execution) than compiled
  - scripts are more portable (in some sense)
  - not always so strict: e.g. byte compiled languages (Java, Matlab, Python)
- script vs "program"
  - scripts have less baggage
  - easy access to/from other programs
  - interpreted languages usually easier to get going
    - ★ often scripts have soft, or implicit types
- use the right tool
  - scripts as glue to connect "programs"
  - programs for big projects

What I think you need

- A scripting language: bash, Perl, Python, ...
  - too useful
- A data crunching tool: Matlab, R, ...
  - too useful
- A hardcore (compiled) language: C, C++
  - speed
  - for the concepts you learn as part of it
- A computer algebra tool: Maple, Matlab (through its symbolic manipulation toolbox), Mathematica, Maxima, Magma, ...
  - you are seriously asking why?
- Other useful "programming" tools
  - Makefiles (Gnu make or Apache Ant or RAKE)
  - SQL
  - spreadsheets

What I know

- A lot of Matlab
- A lot of Perl
- Decent amount of C and Julia
- Enough Unix tools (e.g., gmake, bash, sed, ...) to get by
- Enough Excel to get by
- Enough Maple to get by
- A smattering of many other languages: TCL, IDL, C++, Python, Haskell, Erlang, SQL, Java, PHP, Lisp, ...
- Various bits of markup: LaTeX, Markdown, HTML, XML, GML, ...

What I plan to learn when I get time :-)

- Javascript
  - web programming is great for getting ideas across to a wider audience
- Julia (better)
  - Matlab replacement
- Python (better)
  - maybe a better scripting language than Perl?
  - lots of my collaborators use it
  - libraries: NetworkX, Twisted, ...
- Haskell (better)
  - designed for maths
  - functional programming
- A couple of the programming languages intended to teach kids
  - for outreach work
- Maybe R
  - blame Jono
- Maybe Magma
  - designed for algebra, number theory, algebraic geometry, ...

Learn fundamentals not syntax

Learn "tricks" that apply across languages

- control: loops, conditionals, ...
- modularisation: functions, procedures, subroutines, ...
- data structures: lists, hashes, arrays, ...
- techniques: hashing, caching, indirection, ...
- regexps

http://rosettacode.org/wiki/Rosetta_Code

Regular expressions are your friend

Allow you to match "text" using a pattern

- great for replacing text (with something else)
- great for searching
- great for parsing input data (and cleaning it)

regexp example

Lady Bracknell: To lose one parent, Mr. Worthing, may be regarded as a misfortune; to lose both looks like carelessness. Who was your father? He was evidently a man of some wealth. Was he born in what the Radical papers call the the purple of commerce, or did he rise from the ranks of the aristocracy?

The Importance of Being Earnest, Oscar Wilde

regexp example

Lady Bracknell: To lose one parent, Mr. Worthing, may be regarded as a misfortune; to lose both looks like carelessness. Who was your father? He was evidently a man of some wealth. Was he born in what the Radical papers call the the purple of commerce, or did he rise from the ranks of the aristocracy?

The Importance of Being Earnest, Oscar Wilde

regexp example

how to find double words

Perl's regexp for finding double words:

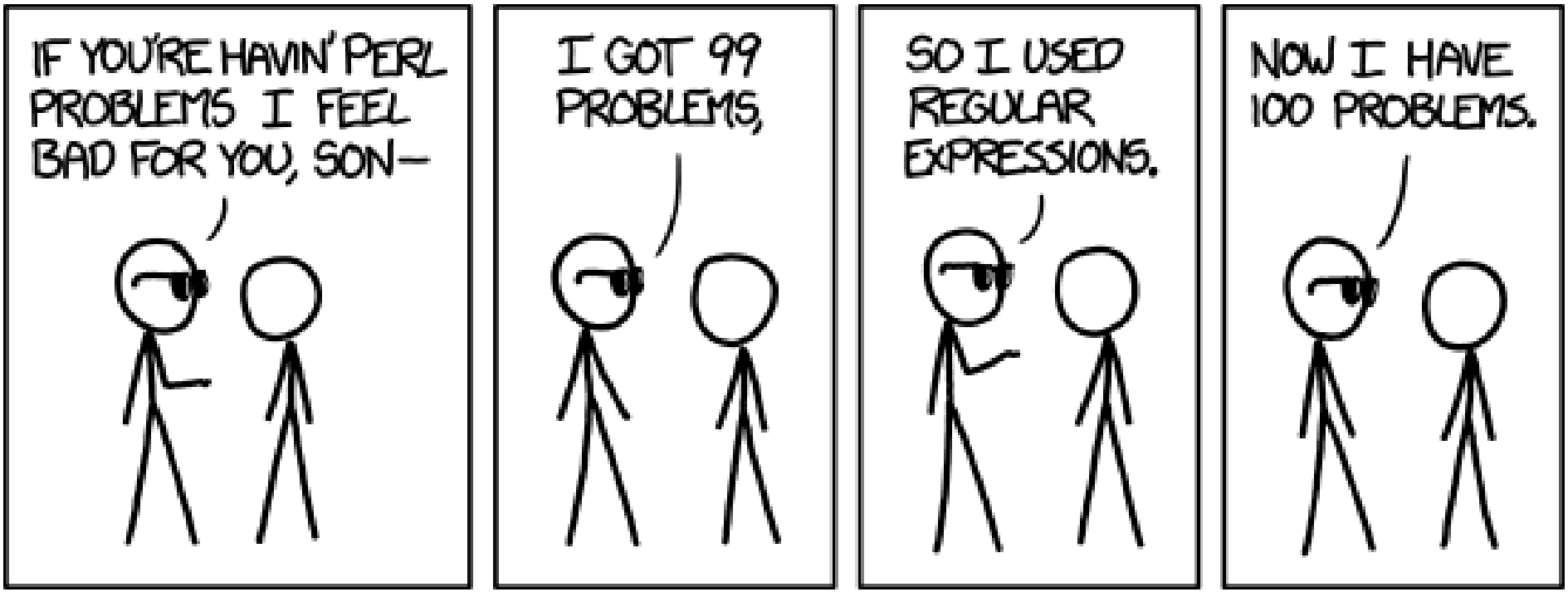
```
#!/usr/bin/perl -n00
# dupwords.pl - find duplicate words in the input stream
print "$ARGV: para $.: ($1)\n"
    while /(\b(\w+)\b\s+\b\2\b)/isg;
```

http://blog.moertel.com/posts/2006-03-01-finding-duplicate-words-in-writing-a-handy-perl-script.html

But don't autoreplace them, because some repeated words are fine, e.g., Buffalo buffalo buffalo buffalo buffalo.

regexps are your friend?

Unfortunately there are a few (similar) syntaxes (I'm used to Perl's) and they require a bit of time to learn.



http://xkcd.com/1171/

Where to start

What do you do, if you don't know where to start?

- Lots of MOOCS and other online courses on specific languages
- Bits and pieces that sit around code: http://software-carpentry.org/
- Sit in on some CS courses
- Notes on getting started in Matlab at http://www.maths.adelaide.edu.au/matthew.roughan/class_notes.html
- Style guides, e.g., Google's

Often good to set yourself a little project to do to motivate learning something.

Summary

- You must learn to code
- You should learn enough to be able to choose the right tool

Assignment

> When someone says, "I want a programming language in which I need only say what I want done," give him a lollipop.
>
> *Epigrams in Programming, 93., Alan Perlis*

- Code Hello World in three different languages (your choice).
- Or, go to Project Euler, pick a problem, and implement the solution to that problem using three languages.

> Hofstadter's Law: It always takes longer than you expect, even when you take into account Hofstadter's Law.
>
> *Douglas Hofstadter, Gödel, Escher, Bach: An Eternal Golden Braid*

Further reading I

Donald E. Knuth, Tracy L. Larrabee, and Paul M. Roberts, *Mathematical writing*, Mathematical Association of America, 1989, jmlr.csail.mit.edu/reviewing-papers/knuth_mathematical_writing.pdf, contains a huge amount of very good advice, but loosely organised (just reports of a set of lectures).