

Our toolbox

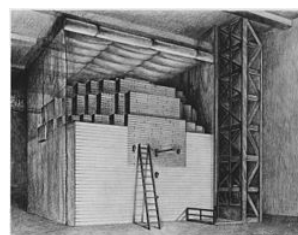
Recap

Decent summaries of Bayesian vs frequentist (likelihood / 'maximum likelihood') estimation:
<https://www.youtube.com/watch?v=0LQmZXCWMEI>
<https://www.psychologicalscience.org/observer/bayes-for-beginners-probability-and-likelihood>

The tools

- Having great tools and using them is half the battle
- Don't be afraid to ask for help, or offer it
- **Software Carpentry & Data Carpentry**
- Keep attending industry / academic meetups, seminars and hackathons
- Become multilingual but don't lose old skills
- **Be organised.**

Fermi estimation



- Order-of-magnitude (10^n) calculations can get you a long way!
- Also good for wrapping your mind around problems, which in turn will help you **design experiments** and **sanity-check results**

Big power

- Effect size = $\mu - \mu_0$ (difference from null)
- Many of the effect sizes we might want to detect are *incredibly* small.
- So we need **big** samples:

Effect size ($\mu - \mu_0$):	+1.0	+0.1	+0.01
Detect with ANOVA at:	Expected N:	N	N
$\alpha = 0.05, \beta = 0.5$	17	1,570	156,978
$\alpha = 0.01, \beta = 0.5$	25	2,336	233,580
$\alpha = 0.001, \beta = 0.5$	37	3,415	341,499

See e.g.
<http://www.sample-size.net/sample-size-mean/>
<http://www.biolins.ac.uk/biolins/psocods/teaching/coursepowerandsamplesize.pdf>

Formats

- Get used to dealing with *zillions* of formats, and converting between them
- D-R-Y still applies, so look for libraries to help you first – but always check the output(!)
- When writing research data yourself, consider the person coming after you...



INTRODUCING THE XKCD STACK

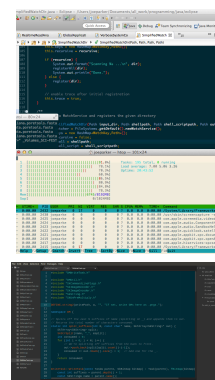
EBNF/CSS
BROKEN JAVA APPLET
ARCHIVE.ORG MIRROR
HYPERCARD.JS
OBSCURE ON RAILS
[BLOCKED BY ADBLOCKER]
MONGODB/EXCEL
SOME PIECE THAT WORKS SO NOBODY ASKS ANY QUESTIONS
TRIPLY-NESTED DOCKER
PARAVIRTUAL BOY®
A DEV TYPING REAL FAST
OLDER VERSION OF OUR SOFTWARE
MYSTERY NETWORKING HORROR
MICROSOFT BOB SERVER®
A GIANT CPU SOMEONE BUILT IN MINECRAFT

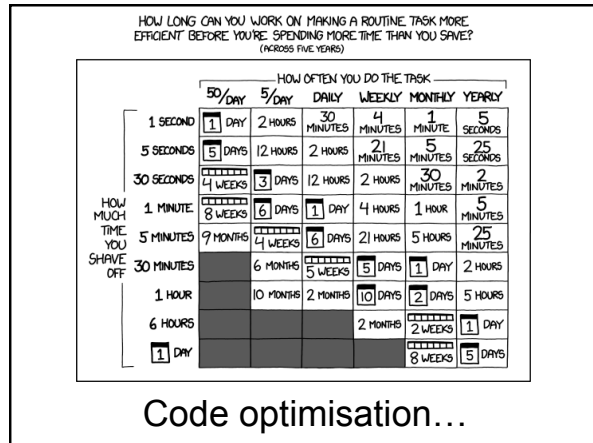
Notes, comments and metadata

- Record *everything* if you **ever** want to repeat it.
- Casual notebooks (Evernote, Notes etc) all useful
- Metadata, parameters and versions for analyses
- Good lab-book **as important** as wet science
- Ever more tools appearing to help track analyses (iPython, R notebooks, etc)
- Code needs to be well commented, well named, and well designed
- (GOD-objects)

IDEs

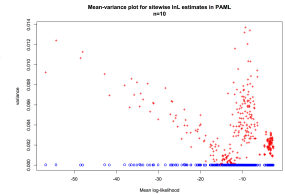
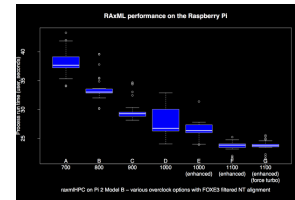
- Integrated development environment – anything that makes it easier to organise, import, write, test, build and publish code.
- Full IDEs:
 - Eclipse (mainly Java)
 - Xcode (iOS, C)
 - Rstudio
- High-powered text editors:
 - Atom
 - TextWrangler





Benchmarking

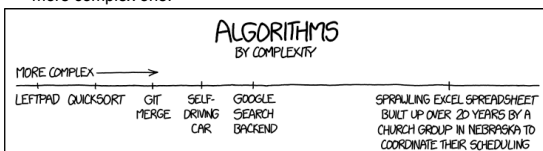
- Analyse execution times and system usage:
 - Which steps are taking the most time/RAM?
 - Has the performance improved? Worsened?
 - HOW MUCH MORE \$€ do we need...
- Measurements aren't precise, just so happens to be a... statistical problem(!)



Algorithm complexity

- We have a model with N parameters: How long will it take? And if $N+1$? $N+m$?
- Algorithmic complexity** attempts to analyse the likely best- worst- and average-case execution time as a function of problem size
- Simple for simple algorithms, e.g.
- Less so for complex ones
- An apparently slower method may have better complexity implications than faster, more complex one.

Method	Best	Worst
Quick sort	$O(n \log n)$	$O(n^2)$
Bubble sort	$O(n)$	$O(n^2)$
Merge sort	$O(n \log n)$	$O(n \log n)$



Parallelisation

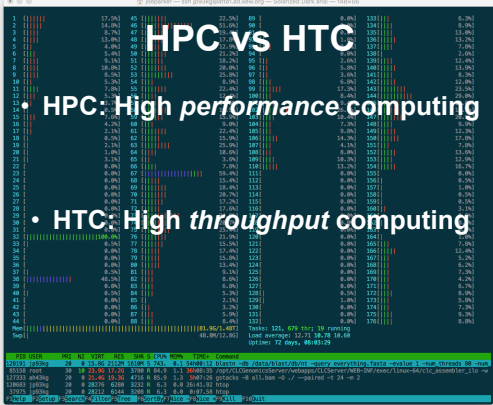
- “for (n in 1:lots)”
- Parallelisation is an obvious way to speed up / power up analyses
- Not all parallelisations are the same
- Main types:
 - Array jobs
 - Highly parallel/distributed jobs (MPI)

Array jobs

- We have multiple replicates / subsets
- Each is effectively independent – but we have a **lot** of them to get through
- Analyse separately, together, or sequentially
- Synchronisation irrelevant
- “I *could* run each of these on my personal machine, but it takes ages”
- e.g. BLAST metagenomes; parameter sweeps
- Optimisation is easy; optimise each replicate and scale

Highly parallel / MPI jobs

- A very large dataset, but our **inference depends on all of the information**
- We can/need to split the algorithm across several / 000s of cores
- Working simultaneously – **synchronisation** issues relevant
- MPI (message passing interface) / OpenMPI
- Latency (network speeds) between node and disk / each other
- e.g. particle simulations; interactome modelling
- Optimisation is **much harder**



HPC vs HTC

- HPC: High performance computing
- HTC: High throughput computing

HPC vs HTC

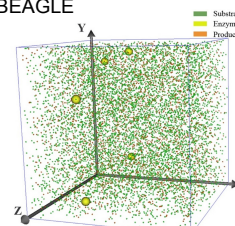
- HPC: High **performance** computing:
 - Fast cores and interconnects for MPI
 - **Lots** of RAM
 - Problems with significant memory footprint
 - e.g. *de novo* genome assemblies, protein folding
 - Each job a significant share of total capacity

HPC vs HTC

- HTC: High **throughput** computing:
 - Lots and lots (>000s) of cores
 - RAM per core more limited
 - Highly parallel / long job time / speed unimportant
 - e.g. public BLAST server, gene annotation, phylogenomics
 - Very large number of cores means capacity for multiple tasks at once

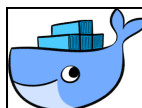
GPUs and MCMC/agent-based modelling

- Certain types of computation require so many parallel but low-level calculations they benefit from quite different architecture
- This can be run effectively on GPUs with some optimisation and libraries e.g. BEAGLE
- e.g. swarming algorithms; agent-based simulation; coalescent phylogenomics; basecalling / HMMs



Cloud

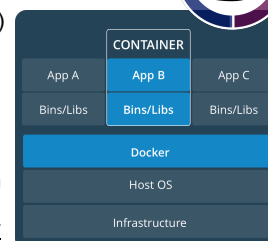
- It doesn't really matter where our machines are
- At the extreme end we can use **cloud computing**
- Rented, temporary time on remote computer
- Private (Amazon AWS; Google) and public (ResearchCloud) providers
- Usage normally paid per-unit time
- Can improve portability
- Costs need to be controlled carefully
- Still has a CO₂ footprint(!)



Docker and VMs



- **VMs** (virtual machines) provide a means to ensure the computing environment is replicable.
- Also enhances portability
- VM + something to run on it = **container**
- Docker and Singularity are just systems for creating, sharing and running containers



Docker and VMs

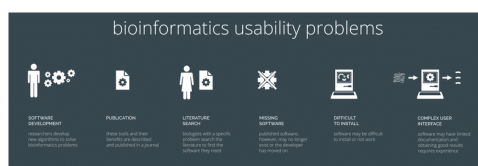
- **Containers** usually contain:
 - environment setup and dependencies
 - script(s) to ingest, analyse and report results
- But usually **don't** contain the research data itself.
- Data normally read/written from attached **volumes**
- Containers are frequently used to scale up analyses, e.g.
 - My Great Pipeline
 - Containerise: MyGreatContainer
 - Replicate:


```
MyGreatContainer(dataset_01);
MyGreatContainer(dataset_02);
MyGreatContainer(dataset_03);
```

Automated testing / integration

- Build testing and integration commonly used in enterprise computing to ensure build quality, and flag up bugs as they arrive:
- Commit change → Build → test → Integrate passing builds to production code
- Tests fired automatically whenever changes made
- Changes (so improvements) can be made continuously ('*continuous integration*')
 - Requires the existence of test methods and data, written early to be effective
- **Continuous analysis** is an emerging idea in bioinformatics, mirroring this philosophy

Bioboxes



alternative with **bioboxes**



Effectively a wrapper for containerised Docker pipeline

ResearchObject

