Scientific Workflows & the Informatics of Model-Data Fusion

Lecture 04
Forecasts must be **transparent** and accountable, and generally they must **repeated** and updated regularly. In keeping with the theme of ‘best practices’, before diving into the statistics of model-data fusion we set the stage with the informatics of model-data fusion by describing new tools that make science more transparent, repeatable, and automated. These tools are not isolated to forecasting – they should become part of all aspects of science -- but they are an essential part of a forecaster’s toolkit
• East Anglia University’s Climate Research Unit (CRU) hacked
  11/17/09
• 1000+ emails, 2000+ docs, source code
• Science out of context
The Aftermath

“...that climate scientists should take steps to make available all the data that support their work (including raw data) and full methodological workings (including the computer codes).”

House of Commons Science and Technology Committee, 2010

- Transparency
- Repeatability
- Provenance
Transparency & Repeatability

- Reproducibility is one of the fundamental tenants of science
- Analyses getting more and more sophisticated
- “Methods” section rarely provides enough information
- Proprietary, unverifiable tools
<table>
<thead>
<tr>
<th>Author</th>
<th>Analytical method(s) used</th>
<th>Analytical tool(s) used</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Waide et al. (1999)</td>
<td>linear and quadratic regressions</td>
<td>none specified</td>
<td>not repeatable</td>
</tr>
<tr>
<td>Mittelbach et al. (2001)</td>
<td>ordinary least-squares regression</td>
<td>SYSTAT 8.0</td>
<td>possibly repeatable; current version 12.0</td>
</tr>
<tr>
<td></td>
<td>Poisson regression</td>
<td>NAG statistical add-in for Excel</td>
<td>not repeatable; software discontinued</td>
</tr>
</tbody>
</table>
|                                | “Mitchell-Olds and Shaw test” (Mitchell-Olds and Shaw 1987) | none specified | not repeatable; software unavailable (but algorithm still available); which of three tests proposed by Mitchell-Olds and Shaw was also not specified
|                                | chi-square exact test                          | StatXact                      | possibly repeatable; no version given         |
|                                | meta-analysis using mixed-effects model        | MetaWin 2.0                   | repeatable; commercial software version still available |
| Whittaker and Heergard (2003)  | Poisson regression                             |                               | not repeatable                                |
| Gillman and Wright (2006)      | ordinary least-squares regression              | not specified                 | not repeatable                                |
|                                | regression on “some” data sets of Mittelbach et al. (2001) | software not specified; data sets reanalyzed | not specified |
| Pärtel et al. (2007)           | multinomial logit regression                   | Statistica 6.1                | possibly repeatable; current release 8.0      |
| Laanisto et al. (2008)         | Fisher exact tests                             | not specified                 | possibly repeatable using available algorithms |
|                                | general linear model                           | Statistica 6.1                | possibly repeatable; current release 8.0      |

Repeatability and reproducibility of ecological synthesis requires full disclosure not only of hypotheses and predictions, but also of the raw data, methods used to produce derived data sets, choices made as to which data or data sets were included in, and which were excluded from, the derived data sets, and tools and techniques used to analyze the derived data sets.

Ellison 2010
Provenance

- "Chain of custody"
- Metadata about analysis and modeling
- Pre & post-processing: transformation, interpolation, gap-filling, filtering, summarizing, exclusion, visualization, ..
Workflows

- Control the flow of information
- More intuitive than raw code
- Modular
- Heterogeneous
- Reusable
- Sharing
PROTIP: NEVER LOOK IN SOMEONE ELSE'S DOCUMENTS FOLDER.
Best Practices for Scientific Computing

Software Engineering is a >$400B industry
Don’t Reinvent the Wheel!
Ecologists Write Crappy Code

What Ecologists Think Good Code Looks Like:

```c
void* Realocate(void*buf, int os, int ns) {
    void*temp;
    temp = malloc(os);
    memcpy((void*)temp, (void*)buf, os);
    free(buf);
    buf = malloc(ns);
    memset(buf, 0, ns);
    memcpy((void*)buf, (void*)temp, ns);
    return buf;
}
```

- Highly optimized
- Not Human Readable
  - Meaningless variable names
- Undocumented
- Hard to maintain
Ecologists write Crappy Code

What Good Code Looks Like:

```c
/* compute displacement with Newton's equation x = v₀t + ½at² */
const float gravitationalForce = 9.81;
float timeInSeconds = 5;
float displacement = (1 / 2) * gravitationalForce * (timeInSeconds ^ 2)
```

- Written to be human readable
- Self-documenting variables
- Documents WHY something is done not HOW
- Define constants for re-use
Start with a Plan

Flowchart/Outline → Pseudocode

```plaintext
## Define settings, load required libraries

## Read and organize data

## Define priors and initial conditions

## Enter MCMC loop

  ## propose new parameter value
  ## run model
  ## evaluate Likelihood and prior
  ## accept or reject proposed value

## Statistical diagnostics

## Visualize outputs

Done with documentation before code!
```
Start with a Plan

Modular Design
(functions, objects, etc)

• Isolates tasks
• Allows reuse (don’t cut-and-paste)
• Only have to change in one place
• Separates purpose (inputs -> output) from implementation
Version Control

- **Update** working copy to include the latest changes in the repository (pull)
- **Merge** these changes into their working copy and resolve any conflicts
- **Edit** their working copy to add new features and fix bugs
- **Commit** these changes and communicate them back to the central repository (push, pull request)
1) Assemble project team

- git add project.txt
- git commit -m "Start project"
- push

project.txt
1) Assemble project team
2) Flowchart / outline

Project.txt

```bash
     | git add project.txt
     | git commit -m "Plan project"
     | push
     | clone
     |

github.com/me/foo

<table>
<thead>
<tr>
<th>Fork</th>
</tr>
</thead>
<tbody>
<tr>
<td>------</td>
</tr>
</tbody>
</table>

github.com/you/foo

<table>
<thead>
<tr>
<th>New pull request</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

Local

---

Owner

Collaborator

---
1) Assemble project team
2) Flowchart / outline
3) Module Design

```
git add project.txt
git commit -m "Modules"
```
Make Incremental Updates

Commit often!!!!
Automate Tests
Continuous Integration
Pull Request #9  Answers to exercise #2

Commit 17258f1
#9: Answers to exercise #2
parevalo authored and committed

This job ran on our new platform for Precise builds. Please read our blog post for more information.

$ git clone --depth=50 https://github.com/EcoForecast/EF_Activities.git

Setting environment variables from .travis.yml

$ export BOOTSTRAP_LATEX="1"

$ export CC=gcc

$ gcc --version

gcc (Ubuntu/Linaro 4.6.3-1ubuntu5) 4.6.3

Copyright (C) 2011 Free Software Foundation, Inc.
## Convert NARR files to CF files

@name met2CF.NARR
@title met2CF.NARR
@export

@param in.path
@param in.prefix
@param outfolder
@param start_date the start date of the data to be downloaded (will only use the year part of the date)
@param end_date the end date of the data to be downloaded (will only use the year part of the date)
@param overwrite should existing files be overwritten
@param verbose should output of function be extra verbose

### met2CF.NARR

#### Description

Convert NARR files to CF files.

#### Usage

```
met2CF.NARR(in.path, in.prefix, outfolder, start_date, end_date, 
    overwrite = FALSE, verbose = FALSE, ...)
```

#### Arguments

- `in.path`
- `in.prefix`
- `outfolder`
- `start_date` the start date of the data to be downloaded (will only use the year part of the date)
- `end_date` the end date of the data to be downloaded (will only use the year part of the date)
- `overwrite` should existing files be overwritten
- `verbose` should output of function be extra verbose

#### Author(s)

Elizabeth Cowdery, Rob Kooper

[Package PEoAn.data.atmosphere version 1.4.4]
Done is better than perfect.

Sheryl Sandberg
Best Practices for Scientific Computing

1. Write programs for people, not computers.
   1. A program should not require its readers to hold more than a handful of facts in memory at once.

2. Names should be consistent, distinctive, and meaningful.

3. Code style and formatting should be consistent.

4. All aspects of software development should be broken down into tasks roughly an hour long.
2. Automate repetitive tasks.
   1. Rely on the computer to repeat tasks.
   2. Save recent commands in a file for re-use.
   3. Use a build tool to automate their scientific workflows.

3. Use the computer to record history.
   1. Software tools should be used to track computational work automatically.

4. Make incremental changes.
   1. Work in small steps with frequent feedback and course correction.
5. Use version control.

1. Use a version control system.

2. Everything that has been created manually should be put in version control.

6. Don't repeat yourself (or others).

1. Every piece of data must have a single authoritative representation in the system.

2. Code should be modularized rather than copied and pasted.

3. Re-use code instead of rewriting it.
7. Plan for mistakes.

1. Add assertions to programs to check their operation.
2. Use an off-the-shelf unit testing library.
3. Turn bugs into test cases.
4. Use a symbolic debugger.

8. Optimize software only after it works correctly.

1. Use a profiler to identify bottlenecks.
2. Write code in the highest-level language possible.
9. **Document the design and purpose of code rather than its mechanics.**

1. Document interfaces and reasons, not implementations.
2. Refactor code instead of explaining how it works.
3. Embed the documentation for a piece of software in that software.

10. **Conduct code reviews.**

1. Use code review and pair programming when bringing someone new up to speed and when tackling particularly tricky design, coding, and debugging problems.
2. Use an issue tracking tool.
Open source code and community models

- Open Source
  - free as in freedom
  - free as in no (direct) cost
  - public license (e.g. GNU GPL, CC)
- Community tools & models
  - Rules and mechanisms for contribution, evolution
Hampton et al. 2015 “Tao of Open Science for Ecology”
Hampton et al. 2015 “Tao of Open Science for Ecology”
And now for something completely different....
Absent: beta, binomial, gamma, exponential, Laplace, Pareto, Bernoulli, geometric, hypergeometric, Wishart
Which Hypothesis is more Likely to have generated this Data?
Likelihood

\[ L = P(X = x | \theta) = P(\text{data} | \text{model}) \]

- Probability of observing a given data point \( x \) conditional on parameter value \( \theta \)

- Likelihood principle: a parameter value is more likely than another if it is the one for which the data are more probable
Likelihood Profile

\[ L = P(x = 1.5 | \mu, \sigma^2) = N(1.5 | \mu, \sigma^2) \]
What $\mu$ maximizes $L$?

\[ L = N(x|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{\sigma^2}} \]

\[ -\ln L = \ln 2\pi + 2\ln \sigma + \frac{(x - \mu)^2}{\sigma^2} \]

\[ -\frac{\partial \ln L}{\partial \mu} = -2 \frac{(x - \mu)}{\sigma^2} = 0 \]

\[ \mu_{MLE} = x \]
Maximum Likelihood

1. Write down the Likelihood
2. Take the log
3. Take the derivatives w.r.t. each parameter
4. Set equal to 0 and solve for parameter

Maximum Likelihood Estimate (MLE)