Real World Debugging Strategies

Janoś Gabler

February 17, 2020

Introduction

Goal

- Teach and show a workflow that prevents bugs
- Teach debugging under the assumption that you did not follow the workflow
- Main insights
  - It is important to know a debugger
  - Debuggers do not solve all problems
  - Good strategies, experience and discipline are even more important
- Use real examples and live coding

Meaning of Bug and Bug Free

- User = programmer
  - Code does what docstring says for all valid inputs
  - “Design by contract” (docstring = contract)
- Users = coauthors
  - \+ Meaningful error messages for invalid inputs
- Many users, some unknown
  - Want to allow creative usage
  - Sensible behavior, even with invalid inputs
  - Code does what you expect from function names (nobody reads full docstring)
  - “Design bugs”

Preventing Bugs

Make Your Code Testable

- Write functions with meaningful interfaces
  - Never use global variables inside functions
  - Define valid inputs and expected outputs in docstrings
  - Define interfaces before you write the code!
- No side-effects
  - Avoid classes that modify their state
  - Never change inputs or global variables inside a function
- Do not write 1000 lines scripts and chop them into functions after some line limit

Tutorial: testable interfaces

Test Your Code

- If you have good test cases:
  - Write test first, then write complete function
  - Run test with `--pdb`
- If you do not have test results, work in a notebook:
  - Write function call with test inputs
  - Write function interface and first line
  - Execute after each line you add and convince yourself that result is correct
  - Paste your example as doctest

Tutorial: doctests

Basic Usage of a Debugger

What is a Debugger

- A program that executes a python file and
  - Stops at breakpoints
  - Lets you execute the code line by line
  - Lets you execute arbitrary Python statements within your code
- Best command line debugger is pdb++
- Works like pdb after installing with pip or conda
- Can use an IDE, but don’t underestimate the power of the Terminal

Tutorial2: Example from EPP, debugger invoked via pytest

Summary of Commands

- sticky: Display the code around your current line
- Moving forward:
  - n(ext): Execute next line
  - s(tep): Execute next statement, stopping as early as possible
  - until: Execute code until a higher line than the current one is reached
  - c(ontinue): Execute code until the next breakpoint
- Moving backwards:
  - u(p): Move to an earlier frame in the call stack, i.e. move back
-

Summary of Commands II

- interact: start interactive interpreter with all variables is like the current scope
- !!command: execute the pdb++ command instead of a something with the same name in the current scope
  - Example: if n is a variable, you need !!n to execute the current line
- q(uit): Leave the debugger

Debugging in Different Scenarios

Making Changes to Large Codebases

Debugging While Refactoring a Large Codebase

- Check if there are enough tests
- If not: write regression tests
  - No test should run longer than 15 seconds, test subcomponents if necessary
- Small steps:
  - Make a small change
  - Run tests / make tests pass (use `--pdb`)
  - Commit as soon as tests pass ▸ Video
- Bad example: Start to use estimagic in skillmodels

When It Is Too Late

- You made too large changes
  - If smaller steps are possible: `git reset -hard`
  - Otherwise:
    - Clone repo before change
    - Set breakpoints and run regression tests with `--pdb` side by side
- You have no regression test
  - Commit or stash your changes
  - Make regression tests
- Do not make more changes unless you know exactly what the problem is

Finding Bugs Caused By Others

A Real Example

- First: I do not want to blame anybody here!
- I urgently needed new results in a project
- Needed a new estimagic feature
- After the update:
  - Optimization stopped after a few iterations, returning start params and function value of 0
  - Dashboard displayed empty plots
  - x-axis of empty plot only showed iteration 0
- Not really a case for pdb++!

Check PR history

Add dashboard gif 1
#105 by roecla was merged on Dec 6, 2019 • Approved 4 of 4

Refactor interfaces for minimization. 11
#103 by tobiasraabe was merged on Dec 5, 2019 • Approved

Add decorators for conversions and logging. 25
#102 by tobiasraabe was merged on Dec 8, 2019 • Approved 1 of 1

implement logging 1
#101 by janosg was closed on Nov 28, 2019 • Review required 1 of 4

Bump version 1
#100 by janosg was merged on Nov 25, 2019 • Approved

Add database functions 46
#97 by janosg was merged on Nov 25, 2019 • Approved

Add tests for different ways constraints may be specified.
#96 by roecla was merged on Dec 6, 2019 • Approved

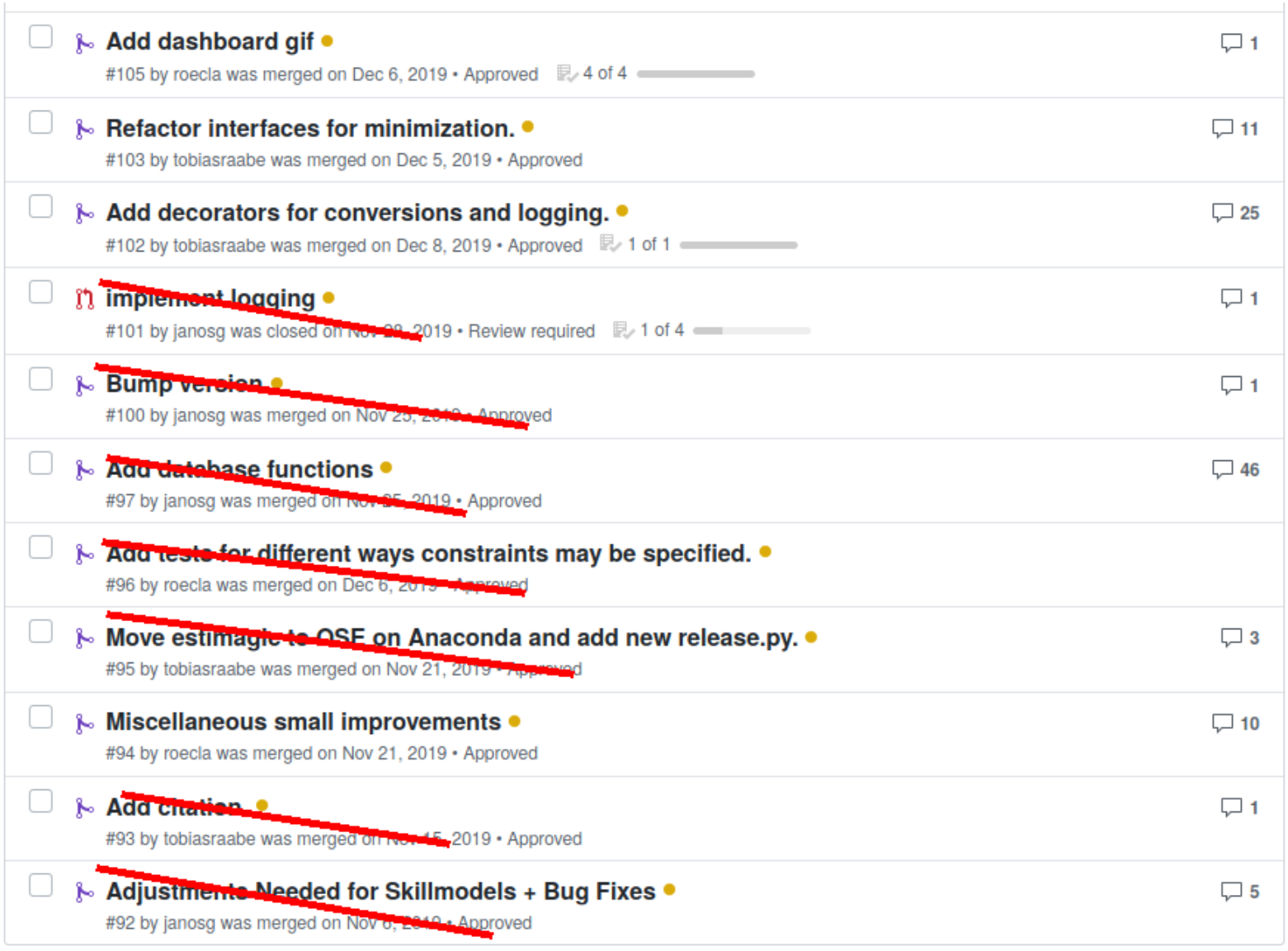
Move estimagic to OSE on Anaconda and add new release.py. 3
#95 by tobiasraabe was merged on Nov 21, 2019 • Approved

Miscellaneous small improvements 10
#94 by roecla was merged on Nov 21, 2019 • Approved

Add citation. 1
#93 by tobiasraabe was merged on Nov 15, 2019 • Approved

Adjustments Needed for Skillmodels + Bug Fixes 5
#92 by janosg was merged on Nov 6, 2019 • Approved

Check PR history



First Bug: Empty Plots

- Strategy: Fix dashboard first, it helps for rest
- Try to pin it down to one small PR ⇒ too many
- Ask Klara ⇒ I'm in Houston, she sleeps
- Generate hypothesis:
  - Plots work for ordered logit but not skillmodels
  - Could be a scaling issue
- It has worked before ⇒ need to find recent change that affected scaling

First Bug: Empty Plots II

- Search all .py files in `dashboard` for "scaling" (no result) and "scale" (result in convergence_tab.py)
- Check history on GitHub

First Bug: Empty Plots II

- Search all .py files in dashboard for "scaling" (no result) and "scale" (result in convergence_tab.py)
- Check history on GitHub

```
        tab = Panel(
-           child=column(children=plots, sizing_mode="scale_width"),
+           child=column(children=plots, sizing_mode="scale_height"),
            title="Convergence Plots",
        )
```

- Changing it back helps
- But now I see that all datapoints are plotted at iteration=0

Second Bug: Iteration Counter

- I know how the iteration counter works
- It’s in optimize.py. Search recent diffs for "counter"

```
c = np.zeros(1, dtype=int)


def internal_criterion(x, counter=c):
    p = reparametrize_from_internal(
        internal=x,
        fixed_values=params["_internal_fixed_value"].to_numpy(),
        pre_replacements=params["_pre_replacements"].to_numpy().astype(int),
        processed_constraints=constraints,
        post_replacements=params["_post_replacements"].to_numpy().astype(int),
        processed_params=params,
    )
c = 0
```

- We increase counter by `counter += 1`, does not work for int

Third Bug: Weird Criterion Value

- Likelihood function works if called outside optimizer
- Likelihood always zero if called by optimizer
- Take a walk first!
- Make a list of differences between situations
  - Optimizer calls function several times
  - During optimization, function gets decorated
  - process_arguments makes a deepcopy of arguments
- Saw immediately, that the copy was the problem
- Otherwise I would have tried them one by one

Guidelines

When to use debuggers?

- Debuggers are great if
  - You get an error and want to find the source
  - You know in which function the problem is (i.e. less than 20 lines)
  - You have a theory where the problem starts
  - You debug a regression test
- Otherwise your brain might be the better debugger
- Discipline is more important than smartness!

Are you Done?

- Did you really find the bug or just a symptom?
  - If you get an error from a check, don’t delete it
  - Do you understand why your solution works and the old code did not?
  - Is your fix good and in the right place?
- Why didn’t you find it earlier?
  - Write a test that would have located the bug
- Did you make similar mistakes somewhere else?
  - Wrong assumptions about a function / library?
  - Talk to the person who introduced it!

Debugging Commandments

- Don’t make it worse: Commit before any change
- Shorten the debug cycle: Simplify your example until it runs in a few seconds.
- Think!
  - Understand the problem before you try solutions
  - Formulate questions before you collect data
- Use a debugger instead of print statements
- Only stop when you are really done!

More Information

- Raymond Hettinger's Talk
- pdb++ Documentation