Final USRA Report

Ray Wu

University of British Columbia

August 22, 2018

Outline

Motivation

- Motivation for testing numerical algorithms:

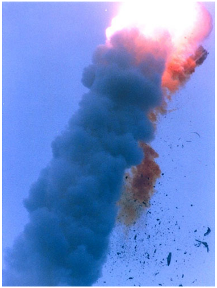


Figure 1: Ariane 5

Figure 2: Patriot

- Cost of mistakes: $500 million | 28 dead
- Due to discretization, error can be difficult to detect

Literature Overview

- Some of the existing methods for numerical testing as listed in [Roy05]:
    - expert judgment
    - error quantification
    - consistency/convergence
    - order of accuracy
- mutation testing: used in other domains to measure strength of test sets.

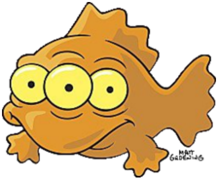
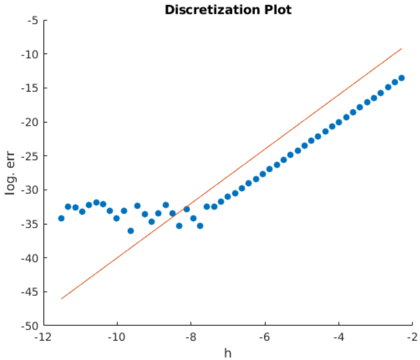


Figure 3: Mutation

Problem Description

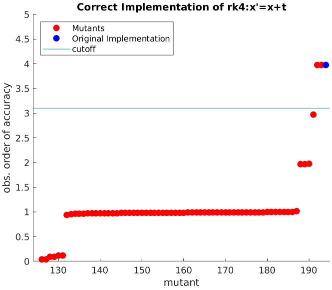
- The abovementioned methods for numerical testing all depend on certain thresholds.
  - Is this convergence analysis acceptable?

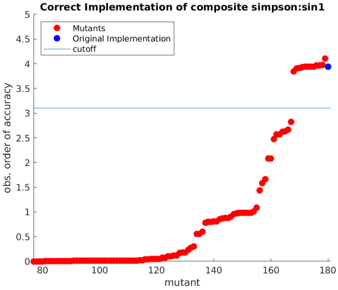


- Introduced $\mathcal{O}(h^{3.5})$ error into Simpson's quadrature rule.

Contributions

- Applied mutation testing to generate mutants.
- Then applied traditional numerical analysis techniques to the mutant code.
- Used metrics to evaluate and compare mutant code with target code





Methodology

- used MATmute to create mutations
- subject each mutant to test problems at different discretization steps
- measure various statistics such as
  - maximum error
  - regression slope
  - R^2 measure of linearity
  - ... and others
- two criterion proposed to measure code correctness:
  - Comparison with target code
  - Multiple criterion removal (mutants must satisfy multiple criterion to pass)
  - both lead to the binary classification of mutants in terms of correct and incorrect mutants; hope to use statistics on these to predict code correctness

Results

- Tested numerical differentiation, integration, spline, and runge-kutta implementations.
- Additionally, ∼60 student implementations of `rk4` and `bdf2` were tested, each target generated around 400-500 mutants and about 200-300 were viable
  - a *viable mutant* is one that doesn't throw an error when it is executed.
  - example of non-viable mutants are those which result in division by 0, accessing a variable that is not declared, etc.
- some examples of interesting mutant codes are as follows:
  - doubling/halving/squaring h, the discretization parameter

Results

- Two examples of interesting mutants:

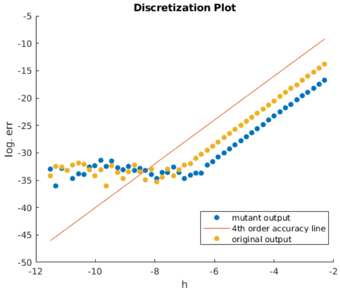


Figure 4: Halving *h*

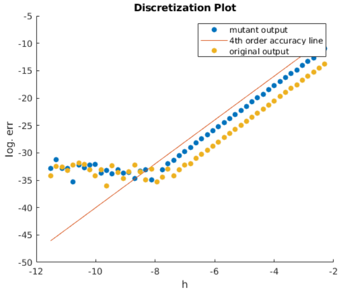
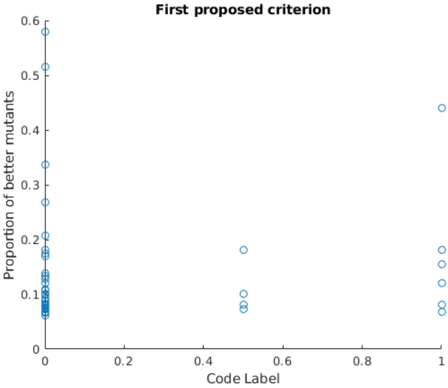


Figure 5: Doubling *h*

Analysis

- wish to use the number of mutants in each category to determine program correctness
- wish to be able to explain why certain mutants do not get removed



Future Work

- Generalization of our testing process to the following:
  - Where test examples are hard to generate
  - Where the algorithm doesn't use a discretization step
- Automatic detection of discretization range using segmented least squares
- Running the same analysis on buggy targets
- Use as an autograder in numerical analysis courses

Conclusion

- Examining the mutation in the surviving mutants is instructive about the code's implementation.
- It is not clear if thresholds can be removed.
- more reasonable p-val vs. as long as error ¡ 0.05

References

Christopher J Roy, *Review of code and solution verification procedures for computational simulation*, Journal of Computational Physics **205** (2005), no. 1, 131–156.