Employing property-based testing

Some experiences from testing linalgwrap

Michael F. Herbst
michael.herbst@iwr.uni-heidelberg.de
http://blog.mfhs.eu

Interdisziplinäres Zentrum für wissenschaftliches Rechnen
Ruprecht-Karls-Universität Heidelberg

19th January 2017

Contents

1. Property-based testing
   - Introduction
   - Demo
2. Testing in `linalgwrap`
   - Overview of `linalgwrap`
   - Testing strategies
3. Outlook

UNIVERSITÄT HEIDELBERG ZUKUNFT SEIT 1386

IWR Interdisciplinary Center for Scientific Computing

Contents

1 Property-based testing
- Introduction
- Demo

2 Testing in linalgwrap
- Overview of linalgwrap
- Testing strategies

3 Outlook

Property-based testing

- Testing technique for unit tests
- Auto-generated test cases
- Extremely useful tool to *localise* a bug
- Originally `QuickCheck` framework in Haskell
- Nowadays widely available

General idea

- Take what we know about our code
- Generate the test cases from that
- *Preconditions*:
    - Requirements before the run

  $\Rightarrow$ Data generation / state setup
- *Postconditions*:
    - Guaranteed state after the run

  $\Rightarrow$ Assertions to check for

General idea

- Take what we know about our code
- Generate the test cases from that
- *Preconditions*:
  - Requirements before the run
  - ⇒ Data generation / state setup
- *Postconditions*:
  - Guaranteed state after the run
  - ⇒ Assertions to check for

Test case generation

- Use random seed
- Generate multiple sets of input data

$\Rightarrow$ Per run: *Many* different test cases

- Execute from easy to hard
- On failure: *Shrink*

$\Rightarrow$ Try to find simplest failing case

Hierarchy of methods

- Assert postcondition properties:
  - Reversing a string twice gives back the original.
  - Cauchy-Schwarz inequality holds
- Comparative testing:
  - Same behaviour in model and system under test (SUT)
- Random *chain* of operations:
  - Do model and SUT show an equivalent state?

$\Rightarrow$ *Stateful testing*

Software and implementations

- Varying feature sets
- C++: https://github.com/emil-e/rapidcheck
- Haskell: http://www.cse.chalmers.se/~rjmh/QuickCheck
- Python: http://hypothesis.works

Advantages

- Make yourself aware of pre/postconditions
- Different test cases each run
- Test what you did *not* think about
- I.e. fuzz your own program
- True stateful testing possible

Demo

DEMO

https://github.com/mfherbst/c14h-rapidcheck-catch

Challenges

- Test cases: Hard enough, but not too hard
- Complex preconditions:
    - Existence of solution
    - Dimensionality
    - Numerical stability

$\Rightarrow$ Separate numerical and implementation error

Contents

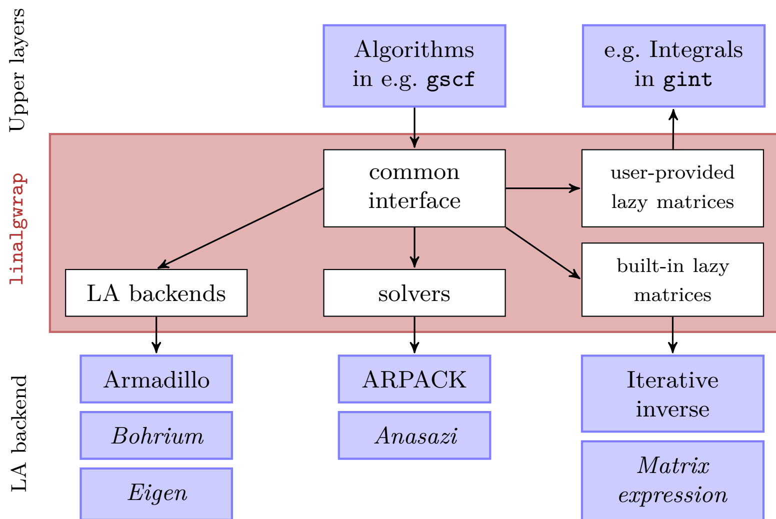
1. Property-based testing
   - Introduction
   - Demo
2. Testing in `linalgwrap`
   - Overview of `linalgwrap`
   - Testing strategies
3. Outlook

UNIVERSITÄT HEIDELBERG ZUKUNFT SEIT 1386

IWR Interdisciplinary Center for Scientific Computing

linalgwrap

Linear algebra wrapper library — https://linalgwrap.org



Generators

- Generator for scalars:
  - Values within $[-10^5, -10^{-5}] \cup \{0\} \cup [10^{-5}, 10^5]$
  - Extremes less likely
- Generator for vectors and matrices:
  - The higher the rank the smaller the size
- *Not* checked:
  - Norm
  - Condition number

Generators

- Generator for scalars:
  - Values within $[-10^5, -10^{-5}] \cup \{0\} \cup [10^{-5}, 10^5]$
  - Extremes less likely
- Generator for vectors and matrices:
  - The higher the rank the smaller the size
- *Not* checked:
  - Norm
  - Condition number

Numerics-aware comparison

- Based on `krims::NumComp`
- Error boundaries relative to *machine epsilon*
- Flexible interface
  - Temporarily tighten / loosen threshold
- Not fast, but informative
- Support for vectors and matrices

Numerics-aware comparison

Example

```
// Check that matrices are initialised to zero
//
using namespace krims;
const size_t size = *gen::numeric_size<2>();
Matrix m(size,size);
Matrix n(size,size);
for (auto& elem : n) elem=0.0;

// Compare and use defaults
REQUIRE( numcomp(n) == m );

// Compare and use specified tolerance
REQUIRE( n == numcomp(m).tolerance(1e-13) );

// Compare and use 0.1 times the default tolerance
REQUIRE( n == numcomp(m).tolerance(Lower) );

// The default is relative to machine epsilon
// and can be bumped or decreased locally
```

Vector and matrix operations

- Stored matrices:
  - Generate objects
  - Apply operation
  - Assert equivalence against model
- Lazy matrices:
  - Stateful testing
  - Apply random sequence of operations
  - Compare each time against stored matrix model

Eigenproblems

- No satisfactory input data generation available

⇒ Conventional hard-coded test cases

- Not easy to check results against reference
  - E.g. uniqueness up to Unitarian transformation only

⇒ Check properties instead:

  - Residual of eigenpairs
  - Size of off-diagonal elements

- Linear problems similar

Possible improvements

- More specific generators:
  - Guarantee certain properties
  - Bounds on the norm or condition number
  - Good matrices for addition/subtraction
  - Spectral properties, e.g. positive definiteness
- Eigenproblems and linear problems:
  - Family of parametrised input problems
  - Well-behaving and solvable
  - Increasing in difficulty

Contents

1. Property-based testing
   - Introduction
   - Demo
2. Testing in `linalgwrap`
   - Overview of `linalgwrap`
   - Testing strategies
3. Outlook

UNIVERSITÄT HEIDELBERG ZUKUNFT SEIT 1386
IWR Interdisciplinary Center for Scientific Computing

Summary

- Better code:
  - Different mindset when coding
- Better testing:
  - Lower influence of human error
  - Not one test, but hundreds
- Testing numerical software challenging:
  - Generation of input data
  - Avoiding test failure due to numerics

Ideas and outlook

- Current implementation specific to `linalgwrap`
- Generator procedures could be generalised
- More systematic approaches useful:
  - Exploit mathematical theorems
  - Experiment with some algorithms

Acknowledgements

- Dr. James Avery
- Prof. Andreas Dreuw and the Dreuw group

- Prof. Guido Kanschat
- HGS Mathcomp

References

- https://github.com/emil-e/rapidcheck/blob/master/doc/user_guide.md
- https://en.wikipedia.org/wiki/QuickCheck
- https://linalgwrap.org
- https://linalgwrap.org/krims

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International Licence.