BERKELEY LAB
Bringing Science Solutions to the World

U.S. DEPARTMENT OF ENERGY
Office of Science

Julienne + Assert == Correctness-Checking for Functional Fortran

Katherine Rasmussen, Damian Rouson, Dan Bonachea

go.lbl.gov/julienne
go.lbl.gov/assert

Improving Scientific Software 2025
April 8, 2025

Table of Contents

BERKELEY LAB

Bringing Science Solutions to the World

Temperature Check

Raise your hand if you
use Fortran

Temperature Check

Raise your hand if you
write unit tests

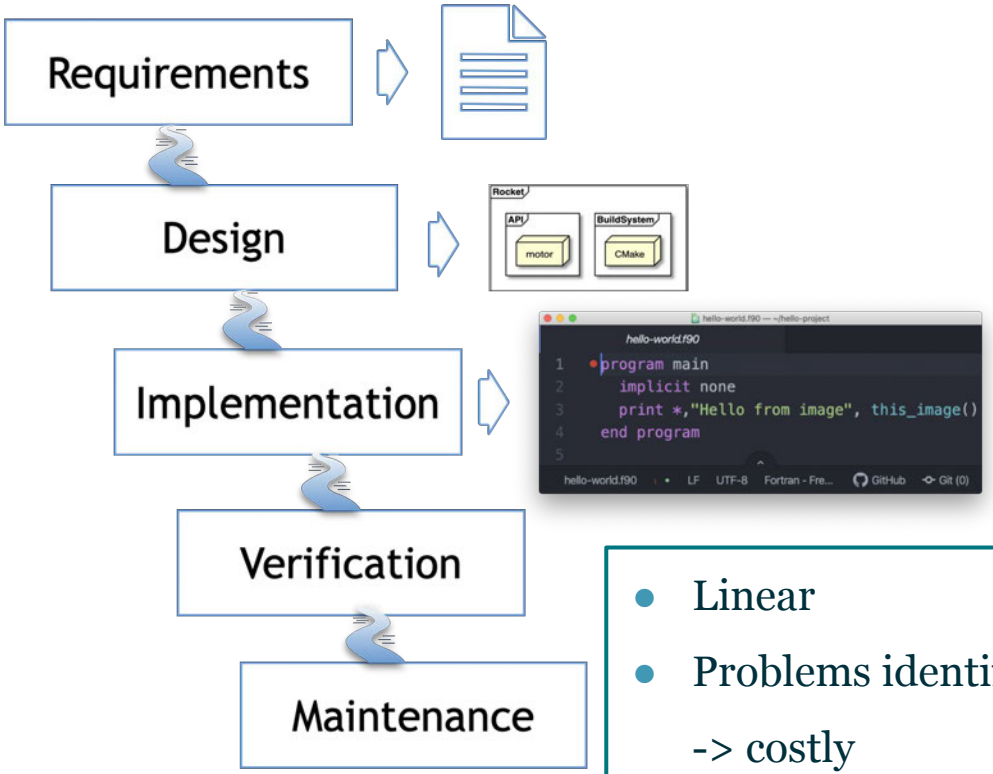
Temperature Check

Raise your hand if you work on a project trying to introduce more unit testing

Agile and Test-driven Development

NASA Mars Climate Orbiter

Traditional Development - Waterfall Model



- Linear
- Problems identified late

  -> costly

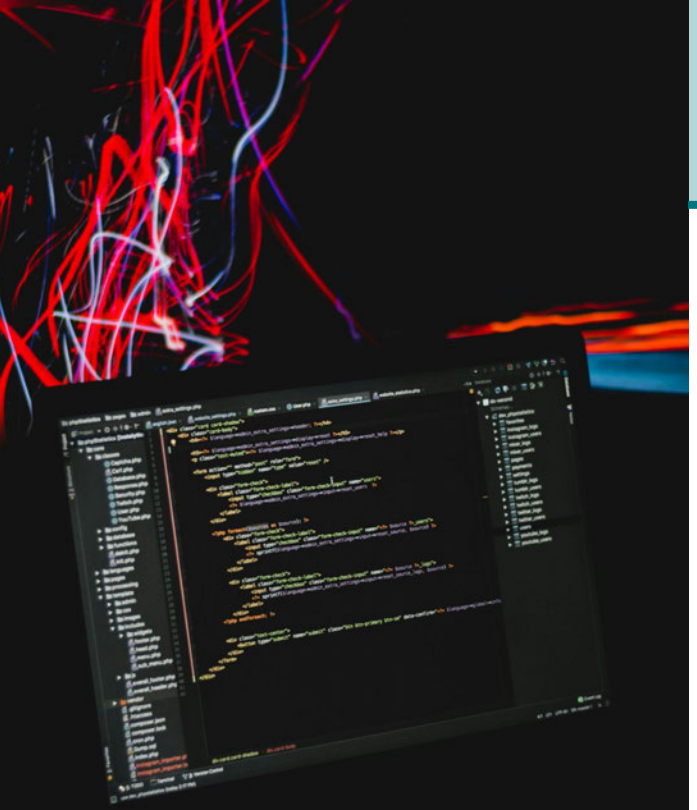


Photo by AltumCode on Unsplash

Agile Development – A Modern Approach

- Agile development practices and tools
  - Lightweight
- Iterative:
  - Release often
  - Identify & fix problems early
  - Respond to change quickly
- Practice: Pair programming
  - Work interactively
  - Collaborate with client/customer/user

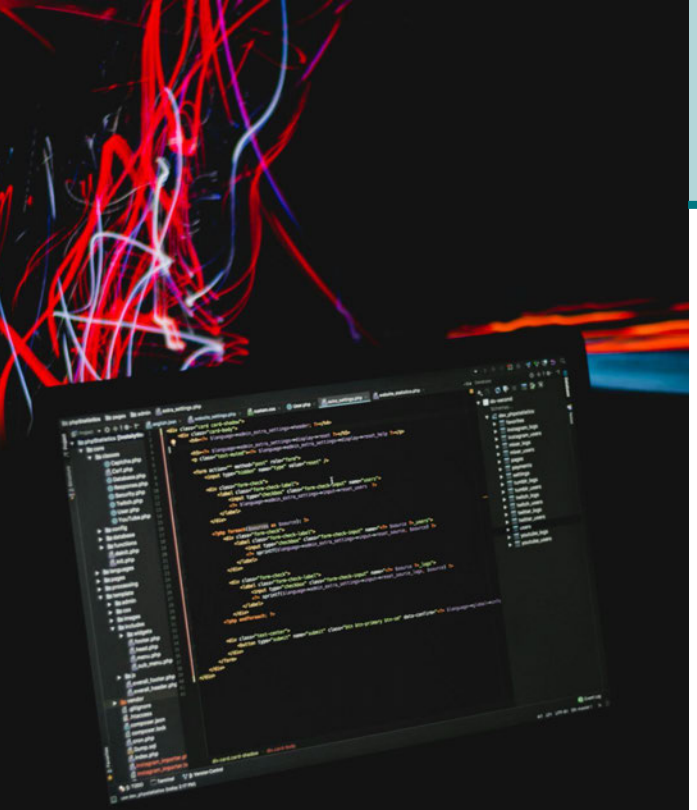


Photo by AltumCode on Unsplash

Agile Development - Tools

- Automated documentation generator
  - Examples: Doxygen, FORD
- Distributed version control
  - Git
  - GitHub, GitLab, BitBucket
- Integrated Development Environments (IDEs)

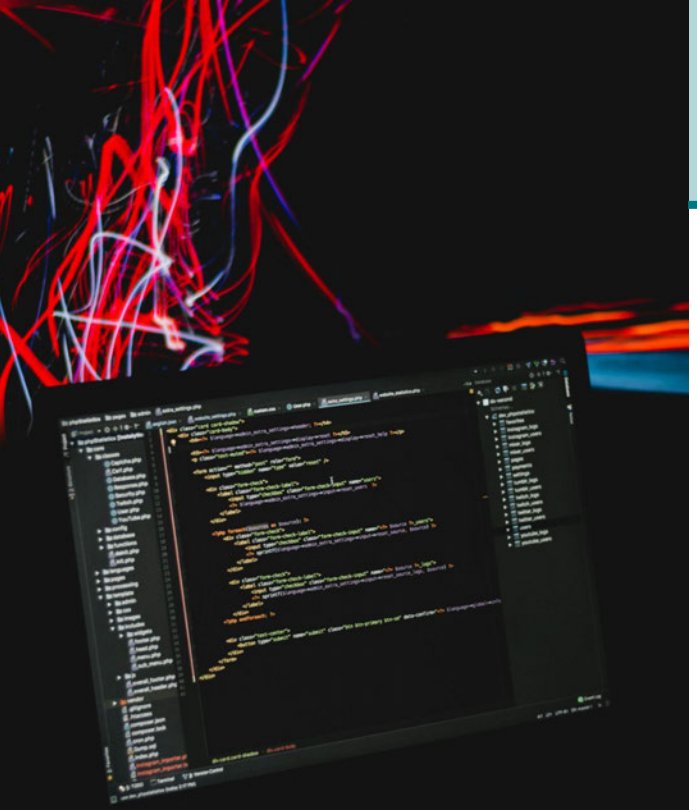


Photo by AltumCode on Unsplash

Agile Development - Practices

- Test-driven development
  - Tests = requirement specifications
  - Write unit test first
  - Write the minimal passing code
  - Rinse and repeat
- Suite of unit tests that can run in seconds or minutes
- Continuous integration/Continuous Deployment (CI/CD)
  - Merge frequently
  - Automate unit testing
  - Test every merge

A Multi-paradigm View of Modern Fortran

Fortran Support for Programming Paradigms

- Array programming
  - multi-dimensional arrays
  - allocatable arrays
  - array statements
  - elemental procedures and more
- Object-oriented programming
  - derived types, polymorphism, etc.
- Functional programming
  - `pure` and `simple` procedures

Fortran – Natively parallel language (since F2008)

- Coarrays and more (collectives, atomics, teams, events, etc)
  - SPMD (single program multiple data) parallelism
  - Uses a PGAS (partitioned global address space) shared memory abstraction
  - PRIF and Caffeine - runtime support library
    - targeting LLVM Flang and LFortran
- `do concurrent`
  - Loop-level parallelism
  - Currently 3 compilers (NVIDIA, Intel, HPE) support automatic offloading to GPUs
- Benefits include easier to write and potentially faster to run

Useful Intrinsic Functions/Features

- `findloc` - Find location of a specified value in an array (optional `mask` arg)
- `pack` - Pack an array into an array of rank one after `mask` is applied
- `count` - Count true values in an array
- `merge` - Merge variables based on the logic of the `mask` arg
- `index` - Find position of a substring within a string
- Note: `mask` argument (boolean logic) of intrinsic functions
- Note: Combining intrinsic function calls can be very powerful

```
55      block
56        integer i, m, num_descriptions, num_matches
57        logical substring_in_subject
58        logical, allocatable :: substring_in_description(:)
59        character(len=:), allocatable :: test_subject
60        num_descriptions = size(test_descriptions)
61        test_subject = subject()
62        substring_in_subject = .false.
63        do i = 1, len(test_subject) - len(test_description_substring) + 1
64          if (test_subject(i:i+len(test_description_substring) - 1) == test_description_substring) then
65            substring_in_subject = .true.
66            exit
67          end if
68        end do
69        if (substring_in_subject) then
70          allocate(test_results(num_descriptions))
71          do i = 1, num_descriptions
72           test_results = test_descriptions%run()
73          end do
74        else ! substring not found in subject
75          allocate(substring_in_description(num_descriptions))
76          num_matches = 0
77          do i = 1, num_descriptions
78            substring_in_description(i) = test_descriptions(i)%contains_text(test_description_substring)
79            if (substring_in_description(i)) num_matches = num_matches + 1
80          end do
81          allocate(test_results(num_matches))
82          m = 0
83          do i = 1, num_descriptions
84            if (m==num_matches) exit
85            if (substring_in_description(i)) then
86              m = m + 1
87              test_results(m) = test_descriptions(i)%run()
88            end if
89          end do
90        end if
91      end block
```

Introduction of lesser used Fortran intrinsics

Transforms code from 37 lines to ? lines

Find out [here](#).

Useful Intrinsic Functions/Features

- `all`
- `dot_product`
- `matmul`
- `maxloc`
- `maxval`
- `minval`
- `range`
- `reshape`
- `scan`
- `transpose`
- `trim`
- `unpack`
- `verify`

- More intrinsics with explanations here

Julienne

go.lbl.gov/julienne

Photo by Igor Miske on Unsplash

Thinly Sliced Vegetables

Julienne

A lightweight Unit Testing Framework

Julienne

- Repository: go.lbl.gov/julienne
- Fortran unit testing framework
- Uses Template Method pattern
- Tests can include serial or parallel features
- Unit tests reside in a collection of unit tests
  - Each collection run by `main` test program
- Testing as specification
- `fpm` package

Photo credit: Paul Hargrove

fpm

- fpm (Fortran Package Manager)
  - Package manager and build system
  - Written in Fortran
  - Maintained by Fortran developer community
  - Automatically detects file dependencies
  - Supports both Fortran and C source code
  - fpm itself is very easy to install
    - Available through package managers (Homebrew, etc)
    - If using gfortran 13 or later, can also compile a one-file version of the `fpm` source code to install it

Julienne Test Output

- Julienne output includes:
  - Test subject (e.g., a class or type-bound procedure)
  - Expected behavior
  - Test outcome (pass or fail)
    - if fails, provides diagnostic information
- Will see more examples in Use Case: Matcha

Testing with Fortran's Parallel Features

- Julienne can not run tests in parallel, however it can run parallel tests
- Parallel tests: Tests that invoke multi-image Fortran features
  - Every image runs every test
  - Only report that a test passes if it passes on all images
  - Only image 1 outputs the results of the tests

Julienne Example (content of video walkthrough)

- Create new fpm project with `fpm new name_of_proj`
- Add Julienne dependency to `fpm.toml` file

Example: `julienne = {git = "https://github.com/BerkeleyLab/julienne"}`

- Copy Julienne example from `julienne/example/example-project` into new fpm project
- Move `specimen_m.f90` into `src` dir
- Run tests with `fpm test`
- Fix purposeful error in source code so test passes

Writing Julienne Tests

- Write a unit test
  - Write a function that returns a `test_diagnosis_t` object
  - `test_diagnosis_t` object contains:
    - a condition in the form of a boolean expression that must pass for test to pass
    - a diagnostics string that is reported if failure occurs

Example unit test for a very basic function `increment()`

```
70   function check_increment() result(test_diagnosis)
71     type(test_diagnosis_t) test_diagnosis
72     type(specimen_t) specimen
73     integer, parameter :: expected_result = 8
74     integer :: actual_result
75
76     actual_result = specimen%increment(7)
77     test_diagnosis = test_diagnosis_t( &
78       test_passed = actual_result == expected_result, &
79       diagnostics_string = "expected result " // string_t(expected_result) &
80       //", actual result " // string_t(actual_result))
81
82   end function
```

Writing Julienne Tests - how to add new unit test

When working from collection of unit tests copied from Julienne `example` directory:

- Update `results` functions
  - Lists and invokes each of the unit tests
    - Create new `test_description_t` object that contains:
      - description of the new unit test
      - function pointer to the unit test function
    - Add to the array of `test_description_t` objects

Update `results()` function to call new test (with gfortran)

```
38  function results() result(test_results)
39    type(test_result_t), allocatable :: test_results(:)
40    type(test_description_t), allocatable :: test_descriptions(:)
41    procedure(diagnosis_function_i), pointer :: check_zero_ptr, check_increment_ptr
42    check_zero_ptr => check_zero
43    check_increment_ptr => check_increment
44
45    test_descriptions = [ &
46      test_description_t("the type-bound function zero() producing a result of 0", check_zero_ptr), &
47      test_description_t("the type-bound function increment() producing the correct incremented integer", &
48      check_increment_ptr) ]
49
50    test_descriptions = pack( &
51      array = test_descriptions, &
52      mask  = test_descriptions%contains_text(test_description_substring) .or. index(subject(), &
53              test_description_substring)/=0)
54    test_results = test_descriptions%run()
55  end function
```

Update `results()` function to call new test (with LLVM Flang)

```
38  function results() result(test_results)
39    type(test_result_t), allocatable :: test_results(:)
40    type(test_description_t), allocatable :: test_descriptions(:)
41
42
43
44
45    test_descriptions = [ &
46      test_description_t("the type-bound function zero() producing a result of 0", check_zero     ), &
47      test_description_t("the type-bound function increment() producing the correct incremented integer", &
48      check_increment     ) ]
49
50    test_descriptions = pack( &
51      array = test_descriptions, &
52      mask  = test_descriptions%contains_text(test_description_substring) .or. index(subject(), &
53              test_description_substring)/=0)
54    test_results = test_descriptions%run()
55  end function
```

Writing Julienne Tests - new collection of unit tests

When creating a new collection of unit tests:

- Create new test object for a new collection of unit tests
  - Extends the `test_t` type from the framework
- Write a `subject` function
  - Describes the collection of unit tests
- Write a `results` functions
  - Lists and invokes each of the unit tests through the `test_description_t` objects
- Invoke the new collection of tests by calling the `report` function on the new test object in the test `main` program

Julienne test `main` program

```
 6 program main
 7   use julienne_m, only : command_line_t
 8   use specimen_test_m, only : specimen_test_t
 9   implicit none
10
11   type(specimen_test_t) specimen_test
12   integer :: passes=0, tests=0
13
14   call print_usage_and_stop_if_help_requested
15   call specimen_test%report(passes, tests)
16   call report_tally_and_error_stop_if_test_fails
```

Skipping Tests

- Julienne provides functionality to skip some tests
- When to skip a test?
  - The test triggers a compile-time or runtime crash
  - Example: gfortran runtime bug in Julienne test suite for the Julienne repository

Skipping Tests

- To skip a test:
  - When writing constructing the list of `test_description_t` objects, don't pass a function name (or procedure pointer) to the `test_description_t` constructor
- Example:

```
#ifndef __GFORTRAN__
   ,test_description_t('constructing bracketed strings', brackets_strings_ptr) &
#else
   ,test_description_t('constructing bracketed strings'                        ) &
#endif
```

```
A format string
   passes on yielding a comma-separated list of real numbers.
   FAILS  on yielding a comma-separated list of double-precision numbers.
      diagnostics: expected 2.718281828459,3.141592653590,1.618033988750;
actual 4.218281828459,4.641592653590,3.118033988750
   passes on yielding a space-separated list of complex numbers.
   passes on yielding a comma- and space-separated list of character value
s.
   passes on yielding a new-line-separated list of integer numbers.
 4 of 5 tests pass.  0 tests were skipped.

The test_result_t type
   passes on constructing an array of test_result_t objects elementally.
   passes on reporting failure if the test fails on one image.
 2 of 2 tests pass.  0 tests were skipped.

The vector test description t type
   SKIPS  on  finding a substring in a test description.
   SKIPS  on  not finding a missing substring in a test description.
 0 of 2 tests pass.  2 tests were skipped.

To also test Julienne's command_line_t type, append the following to your
fpm test command:
-- --test command_line_t --type

_________ In total, 6 of 9 tests pass.  2 tests were skipped. _________
```

Julienne example output

Includes:
6 tests passing
1 test failure
2 tests skipped

Filtering Tests

- Julienne provides functionality to run only a subset of tests
  - Can run a specific collection of unit tests
  - Can run specific unit tests
  - Invoke by using `--contains` flag
    - `fpm test -- --contains string_t`
    - `fpm test -- --contains comma`

```
~/julienne> $ fpm test -- --contains number
```

```
Running only tests with subjects or descriptions containing 'number'.

A format string
   passes on yielding a comma-separated list of real numbers.
   passes on yielding a comma-separated list of double-precision numbers.
   passes on yielding a space-separated list of complex numbers.
   passes on yielding a new-line-separated list of integer numbers.
 4 of 4 tests pass.  0 tests were skipped.

The test_result_t type
 0 of 0 tests pass.  0 tests were skipped.

The vector_test_description_t type
 0 of 0 tests pass.  0 tests were skipped.

To also test Julienne's command_line_t type, append the following to your fpm
 test command:
-- --test command_line_t --type

_________ In total, 4 of 4 tests pass.  0 tests were skipped. _________
```

Filtering tests

Use Case: Matcha

Matcha: T-cell Motility Simulator

- Matcha
  - Motility Analysis of T-Cell Histories in Activation
  - go.lbl.gov/matcha
  - Developed at Berkeley Lab in collaboration with Northern New Mexico College
  - Goal: design virtual T-cells that move like biological T-cells

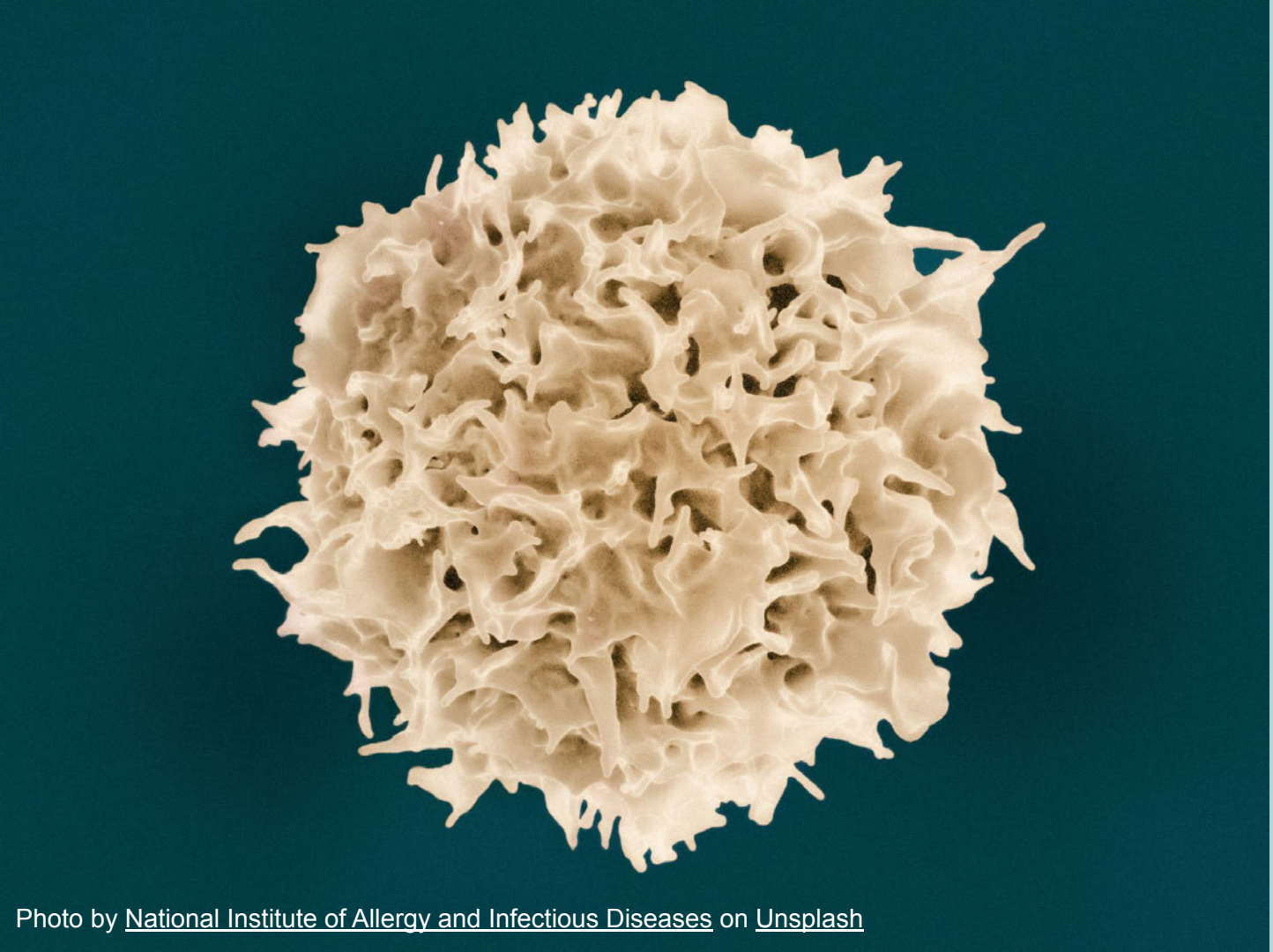


Photo by National Institute of Allergy and Infectious Diseases on Unsplash

Matcha needs to ensure:

A `subdomain_t` object correctly computes a concave Laplacian for a spatially constant operand with a step down at boundaries and more...

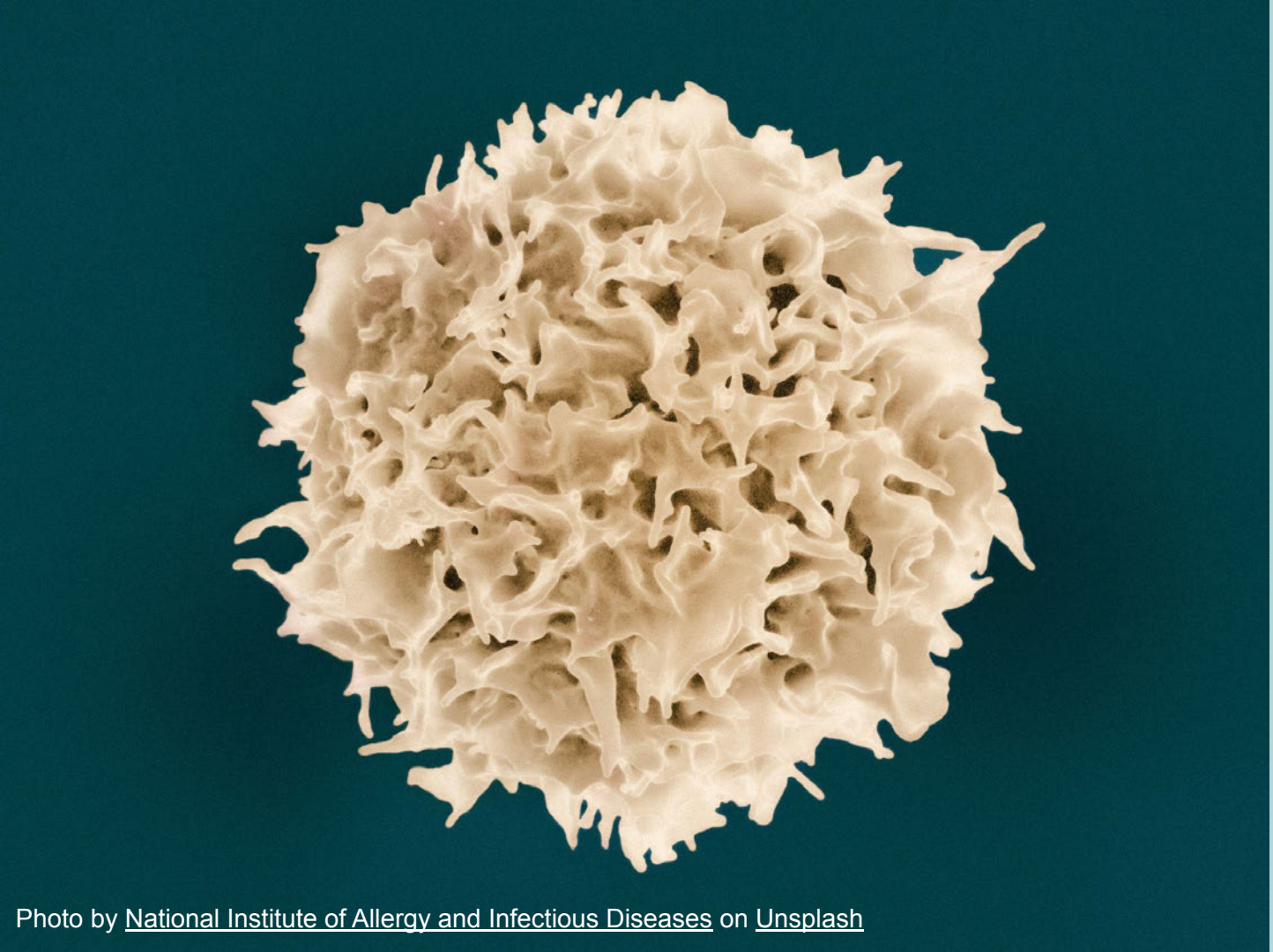


Photo by National Institute of Allergy and Infectious Diseases on Unsplash

Matcha needs to ensure:

A `t_cell_collection_t` object correctly constructs its positions in the specified domain and more...

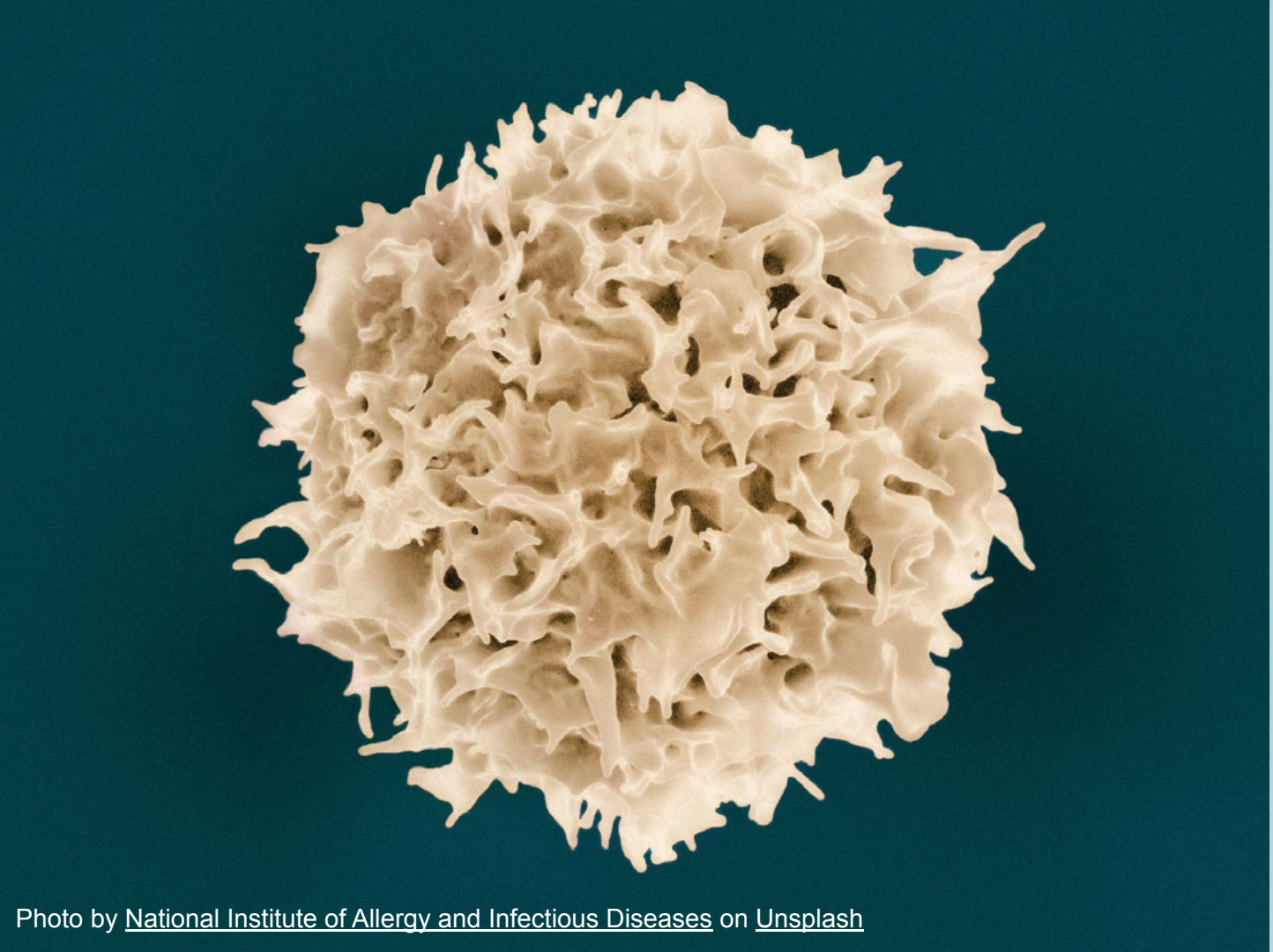


Photo by National Institute of Allergy and Infectious Diseases on Unsplash

Matcha needs to ensure:

A `matcha_t` object can match simulated distributions to empirical distributions

Matcha uses Julienne unit tests to help validate these behaviors

One unit test for Matcha

```
79      function compare_image_distributions() result(test_diagnoses)
80        logical test_passes
81        type(test_diagnosis_t), allocatable :: test_diagnoses(:)
82        type(output_t) output
83
84        integer, parameter :: speed=1, freq=2 ! subscripts for speeds and frequencies
85        double precision, parameter :: tolerance = 1.D-02
86
87        associate(input => input_t())
88          output = output_t(input, matcha(input))
89          associate( &
90            empirical_distribution => input%sample_distribution(), &
91            simulated_distribution => output%simulated_distribution() &
92          )
93            associate( &
94              diffmax_speeds=> maxval(abs(empirical_distribution(:,speed)-simulated_distribution(:,speed))), &
95              diffmax_freqs => maxval(abs(empirical_distribution(:,freq)-simulated_distribution(:,freq))) &
96            )
97              test_diagnoses = [ &
98                test_diagnosis_t( &
99                  test_passed = diffmax_freqs < tolerance .and. diffmax_speeds < tolerance &
100                ,diagnostics_string = "expected max freq < " // string_t(tolerance) // ", actual " // string_t(diffmax_freqs) &
101               ) &
102              ,test_diagnosis_t( &
103                 test_passed = diffmax_freqs < tolerance .and. diffmax_speeds < tolerance &
104               ,diagnostics_string = "expected max speeds < " // string_t(tolerance) // ", actual " // string_t(diffmax_speeds) &
105             ) &
106             ]
107           end associate
108         end associate
109       end associate
110     end function
```

One unit test for Matcha

```
97          test_diagnoses = [ &
98            test_diagnosis_t( &
99              test_passed = diffmax_freqs < tolerance .and. diffmax_speeds < tolerance &
100            ,diagnostics_string = "expected max freq < " // string_t(tolerance) // ", actual " // string_t(diffmax_freqs) &
101           ) &
102           ,test_diagnosis_t( &
103             test_passed = diffmax_freqs < tolerance .and. diffmax_speeds < tolerance &
104            ,diagnostics_string = "expected max speeds < " // string_t(tolerance) // ", actual " // string_t(diffmax_speeds) &
105           ) &
```

Result of unit test:

One or more `test_diagnosis_t` objects

`test_diagnosis_t` object contains:

a condition for the test to pass

a diagnostics string for if it doesn't

Unit Test collection for `matcha_t` type

```
60      function results() result(test_results)
61        type(test_result_t), allocatable :: test_results(:)
62        type(test_description_t), allocatable :: test_descriptions(:)
63        procedure(diagnosis_function_i), pointer :: &
64           compare_image_distributions_ptr &
65          ,compare_global_distributions_ptr
66
67           compare_image_distributions_ptr => compare_image_distributions
68           compare_global_distributions_ptr => compare_global_distributions
69
70        test_descriptions = [ &
71           test_description_t("matching simulated distributions to empirical distribution", compare_image_distributions_ptr) &
72          ,test_description_t("matching simulated global distributions to empirical distribution", compare_global_distributions_ptr) &
73        ]
74        test_results = test_descriptions%run()
75      end function
```

Execution of Julienne test suite for Matcha

```
Running all tests.
(Add '-- --contains <string>' to run only tests with subjects or descriptions containi
ng the specified string.)

A subdomain_t object
   passes on computing a concave Laplacian for a spatially constant operand with a ste
p down at boundaries.
   passes on reaching the correct steady state solution.
   passes on functional pattern results matching procedural results.
 3 of 3 tests pass.  0 tests were skipped.

A t_cell_collection_t object
   passes on constructing positions in the specified domain.
   passes on distributing cells across images.
 2 of 2 tests pass.  0 tests were skipped.

A matcha_t object
   passes on matching simulated distributions to empirical distribution.
   passes on matching simulated global distributions to empirical distribution.
 2 of 2 tests pass.  0 tests were skipped.

_________ In total, 7 of 7 tests pass.  0 tests were skipped. _________
```

Subject of the collection of unit tests

Execution of Julienne test suite for Matcha

```
Running all tests.
(Add '-- --contains <string>' to run only tests with subjects or descriptions containi
ng the specified string.)

A subdomain_t object
   passes on computing a concave Laplacian for a spatially constant operand with a ste
p down at boundaries.
   passes on reaching the correct steady state solution.
   passes on functional pattern results matching procedural results.
 3 of 3 tests pass.  0 tests were skipped.

A t_cell_collection_t object
   passes on constructing positions in the specified domain.
   passes on distributing cells across images.
 2 of 2 tests pass.  0 tests were skipped.

A matcha_t object
   passes on matching simulated distributions to empirical distribution.
   passes on matching simulated global distributions to empirical distribution.
 2 of 2 tests pass.  0 tests were skipped.

_________ In total, 7 of 7 tests pass.  0 tests were skipped. _________
```

single unit test

describes expected behavior

Execution of Julienne test suite for Matcha

```
Running all tests.
(Add '-- --contains <string>' to run only tests with subjects or descriptions containi
ng the specified string.)

A subdomain_t object
   passes on computing a concave Laplacian for a spatially constant operand with a ste
p down at boundaries.
   passes on reaching the correct steady state solution.
   passes on functional pattern results matching procedural results.
 3 of 3 tests pass.  0 tests were skipped.

A t_cell_collection_t object
   passes on constructing positions in the specified domain.
   passes on distributing cells across images.
 2 of 2 tests pass.  0 tests were skipped.

A matcha_t object
   passes on matching simulated distributions to empirical distribution.
   passes on matching simulated global distributions to empirical distribution.
 2 of 2 tests pass.  0 tests were skipped.

_________ In total, 7 of 7 tests pass.  0 tests were skipped. _________
```

Results of collection of unit test

Execution of Julienne test suite for Matcha

```
Running all tests.
(Add '-- --contains <string>' to run only tests with subjects or descriptions containi
ng the specified string.)

A subdomain_t object
   passes on computing a concave Laplacian for a spatially constant operand with a ste
p down at boundaries.
   passes on reaching the correct steady state solution.
   passes on functional pattern results matching procedural results.
 3 of 3 tests pass.  0 tests were skipped.

A t_cell_collection_t object
   passes on constructing positions in the specified domain.
   passes on distributing cells across images.
 2 of 2 tests pass.  0 tests were skipped.

A matcha_t object
   passes on matching simulated distributions to empirical distribution.
   passes on matching simulated global distributions to empirical distribution.
 2 of 2 tests pass.  0 tests were skipped.

_________ In total, 7 of 7 tests pass.  0 tests were skipped. _________
```

All results

A Breather...

Assert

go.lbl.gov/assert

A Functional Programming Pattern

- Recommendations
  - Write pure procedures — especially pure functions with no side effects
  - Referentially transparent
    - same arguments —> same result
  - Define immutable state
    - `associate` with expressions (for example: function invocations)
- Consequences
  - ✅: `error stop`
  - ❌: `write`, `print`, `stop`
- Fortran provides `pure` and `simple` keywords

A Functional Programming Pattern

- Pros:
  - Clarifies data dependencies
  - Easier to parallelize
    - Only `pure` procedures can be invoked inside `do concurrent`
- Cons
  - No output when debugging during development
    - Exceptions:
      - `error stop`
      - Or with the existence of a useful utility (wink, wink)

Programming by Contract

- Programming by contract enforces constraints
  - Preconditions: correctness requirements - must be true before a procedure executes
  - Postconditions: correctness requirements - must be true after a procedure executes
  - Invariants: correctness conditions that must always be true (applies to a whole class, i.e., every procedure).
- In C & C++: Enforce constraints using `<assert.h>`
- In Fortran, constraints can take the form of logical expressions and can be enforced by a utility (wink, wink)

Assert

- Fortran assertion utility that allows for diagnostic output
- go.lbl.gov/assert
- Motivations for utility:
  - To mitigate against a reason Fortran developers often cite for not writing pure procedures: their inability to produce output in normal execution.
  - To promote the enforcement of programming contracts.

Assert

- Can be called in pure procedures
- The provided assertions are function-like preprocessor macros that by default get replaced by nothing
- Workflow:
  - Normal development: no assertions
    - Code runs faster without assertions because no runtime overhead
  - When needing to debug: use `-DASSERTIONS` flag
    - Could also have a special CI run that always uses assertion flag

Getting Started with Assert

- https://github.com/BerkeleyLab/assert/blob/main/example/invoke-via-macro.F90
- Statements required in file where assertions are to be used
  - `#include "assert_macros.h"`
  - `use assert_m`

```
call_assert(1==1)

call_assert_describe(2>0,"example assertion invocation via macro")

call_assert_diagnose(1+1==2,"example with scalar diagnostic
data",1+1)
```

What to expect upon failure

Source Code: Assertion that will fail

```
call_assert_describe(.true. .eqv. .false.,"Breaking the boolean basics")
```

Result:

```
ERROR STOP Assertion "Breaking the boolean basics in file example/invoke-via-macro.F90,
 line 34" failed on image 1
```

What to expect when assertions pass

{nothing}

٩(◕‿◕。)۶

Use Case: Fiats

Fiats: Deep Learning with Fortran

- Machine learning and AI impacts on HPC
- Fiats (Functional inference and training for surrogates)
  - go.lbl.gov/fiats
  - Alternative name: Fortran inference and training for science
  - “training and deployment of neural-network surrogate models for computational science”
  - Automatic parallelization of batch inference

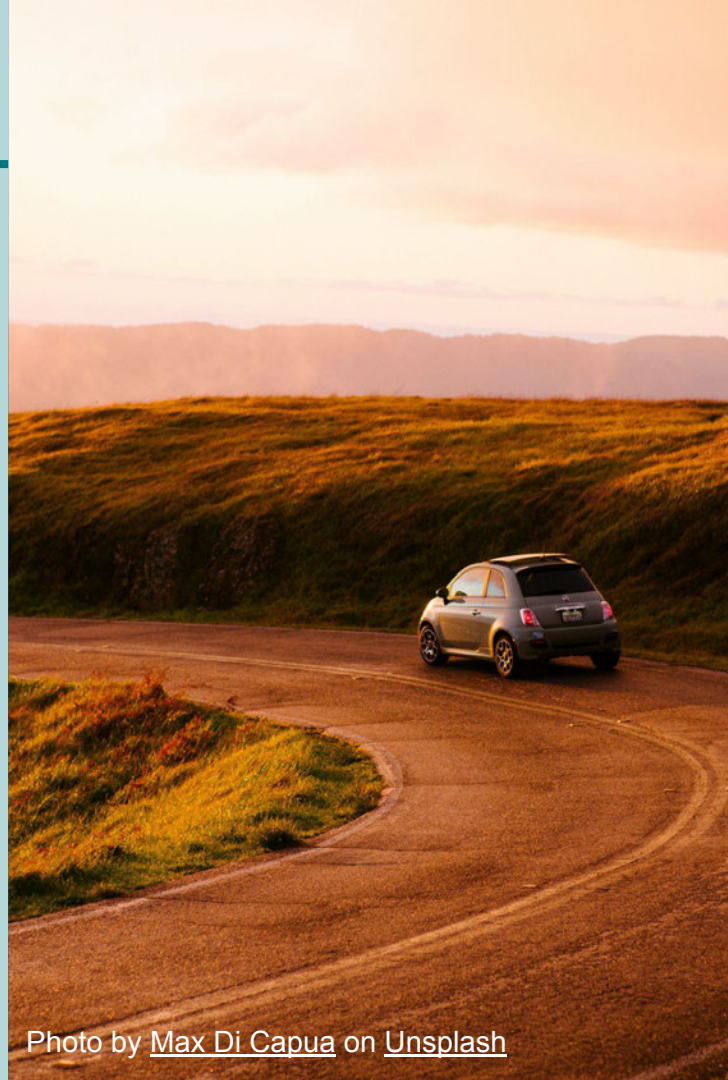


Photo by Max Di Capua on Unsplash

Fiats: Deep Learning with Fortran

- Assertions used for:
  - Validating input data is in correct format
    - Multiple ways of inputting data to construct the neural network
    - Need to validate the various ways
  - Validate the construction of the layers of the neural network
  - And more!

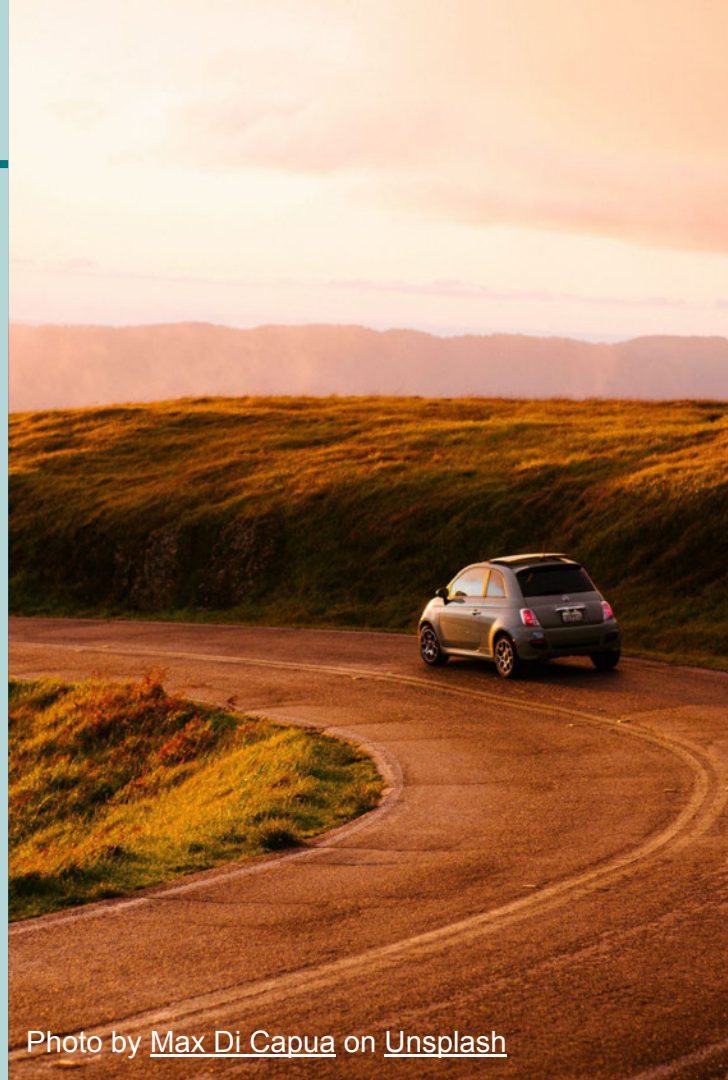


Photo by Max Di Capua on Unsplash

Assert Example with fiats

```
12  module procedure default_real_construct_layer
13
14    type(neuron_t), pointer ::  neuron
15    integer num_inputs, neurons_in_layer
16    character(len=:), allocatable :: line
17    logical hidden_layers, output_layer
18
19    line = adjustl(layer_lines(start)%string())
20    hidden_layers = line == '['
21    output_layer = line == '"output_layer": ['
22    call_assert_diagnose(hidden_layers .or. output_layer, "layer_s(default_real_construct_layer): layer start", line)
23
24    layer%neuron = neuron_t(layer_lines, start+1)
25    num_inputs = size(layer%neuron%weights())
26
27    neuron => layer%neuron
28    neurons_in_layer = 1
29    do
30      if (.not. neuron%next_allocated()) exit
31      neuron => neuron%next_pointer()
32      call_assert_describe(size(neuron%weights()) .ne. num_inputs, "layer_s(default_real_construct_layer): constant number of inputs")
33      neurons_in_layer = neurons_in_layer + 1
34    end do
35
36    line = trim(adjustl(layer_lines(start+4*neurons_in_layer+1)%string()))
37    call_assert_describe(line(1:1)==']', "layer_s(default_real_construct_layer): hidden layer end")
38
39    if (line(len(line):len(line)) == ",") layer%next = layer_t(layer_lines, start+4*neurons_in_layer+2)
40
41  end procedure
```

Fiats: Deep Learning with Fortran

- To learn more about Fiats, please attend Damian Rouson's talk "Cloud microphysics training and aerosol inference with the Fiats deep learning library"
- Date: Wednesday, April 9
- Time: 8:30 AM

Communities & Where to Find More Fortran

Grabbag of Fortran tools

- Codee - Static analysis tool for Fortran and C/C++
  - Code correctness
  - Modernization
  - Codee Youtube Channel
  - Codee training at NERSC
- fortran-linter
- rojff - Return of JSON for Fortran - Utility to support use of JSON files in Fortran source
- iso_varying_string - An implementation of the ISO_VARYING_STRING module as proposed for the ISO standard

Grabbag of Fortran tools

- FORD - (FORtran Documentation)
  - Automatic documentation generator for Fortran projects
  - Can build documentation locally, provides html files
  - Can deploy documentation in Github Actions
  - Example Ford Documentation
- Various tools to convert fixed form code to free form code
- A grabbag of more tools - Beliavsky/Fortran-Tools

Upcoming Fortran Features

- Upcoming features:
  - Generic programming (templates, etc)
    - Type-safe templates: requirements (strong concepts)
      - Must state properties of types
      - Compiler can provide error messages in template source code
  - Asynchrony: tasks, collectives
  - Standardized Fortran preprocessor
- International body - WG5
- Working group - INCITS US National body (informally known as J3)

Fortran Resources & Communities

- Stereotypes about Fortran include ideas of being antiquated
- Vibrant, engaged developer community
- For more Fortran questions or to engage and **share** with the Fortran community, visit:
  - Fortran Lang: fortran-lang.org
    - Tutorials, links to playgrounds, compilers info
    - Helpful language information and advice
    - Link to LFortran, a Fortran compiler and interpreter
  - Discourse: fortran-lang.discourse.group
  - Fortran Wiki: fortranwiki.org
- Fortran at LBNL: fortran.lbl.gov

Acknowledgements

- This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research

- As mentioned, Julienne was initially inspired by the testing framework Veggies
  - Veggies lead developer: Brad Richardson
  - Veggies repository: https://github.com/everythingfunctional/veggies

More Fiats and Fortran Fun

Talk: Cloud microphysics training and aerosol inference with the Fiats deep learning library

Presenter: Damian Rouson

Date: Wednesday, April 9

Time: 8:30 AM

Thank You

Questions?

Email: fortran@lbl.gov