Simulating Turbulent Flow with Synthetic Eddy: System Verification and Validation Plan for SynthEddy

Phil Du (Software)
Nikita Holyev (Theory)

August 14, 2024

Revision History

Date	Version	Notes
2024-02-19	1.0	Initial plan after SRS
2024-03-22	1.1	Address issues from feedback
2024-04-15	1.2	Add unit tests

Contents

List of Tables

List of Figures

1 Symbols, Abbreviations, and Acronyms

Please refer to the Software Requirements Specification (SRS) of this project. Section 1 contains Symbols, Abbreviations, and Acronyms. If you are not familiar with fluids and CFD, Section 4.1.1 of the SRS contains a list of Terminology and Definitions.

The following symbols and operators are used in this document:

symbol	**description**
tol	Tolerance, a small value used when testing differences, see 6.1
std()	Standard deviation of multiple values.
div()	Divergence of a velocity field.

2 General Information

This document lays the plan for the verification and validation of the software SynthEddy. It provides an overview of what the tests aim to achieve, and then details the plan for each test. It also serves as a communication aid between the software development team and the domain experts on the theoretical side, so that it can be ensured that the correct aspect of the software is being tested with valid expectations.

2.1 Summary

The software being tested is a program that generates approximated turbulent flow field with synthetic eddies. It is intended to provided initial and boundary conditions (IC and BC) to CFD solvers for turbulent flow simulations, which would cut down simulation time and save computational resources. An illustration of the flow field that the software is intended to generate is shown in Figure 1.

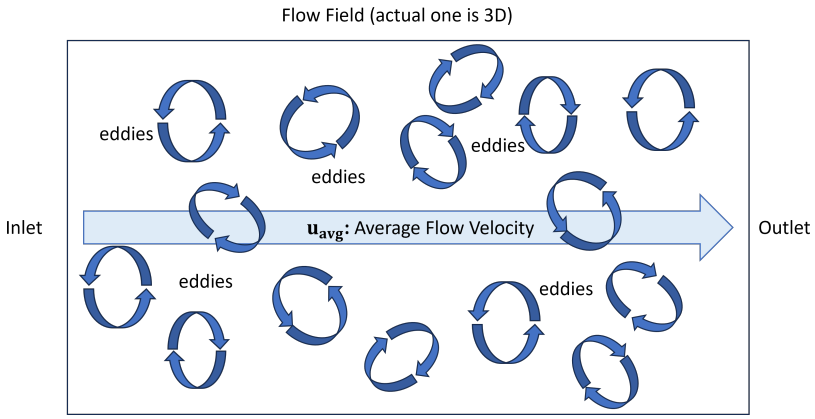

Figure 1: System Context

2.2 Objectives

The most crucial objective here is to build confidence that this software can correctly generate turbulent flow fields according to the synthetic eddy method proposed by Poletto et al. (2013). As no suitable pseudo-oracle

exists to allow for direct comparison of the results, the focus is to test for characteristics that a flow field generated with such method should exhibit. In the future with more theoretical background given, this will expand to making sure the generated flow fields are physically realistic in terms of turbulent properties.

Another objective is to demonstrate the this software can interface with CFD solvers and provide to IC and BC for turbulent flow simulations (future development).

Furthermore, maintainability, namely the ability for user to easily modify the software to fit their specific needs should also be demonstrated.

2.3 Relevant Documentation

- Software Requirements Specification (SRS) is the most important guideline for this VnV plan. It contains the requirements of the software, and the theoretical Modules of the synthetic eddy method.
- Design Documents will be used to guide the unit testing and the design verification. This include the Module Guide (MG) and the Module Interface Specification (MIS).

3 Plan

This section lays out how the VnV will be carried out in aspect and by whom. It also includes the tools that will be used for the VnV.

3.1 Verification and Validation Team

- Phil Du - Developer: Writing test scripts according to VnV and implementing automated tests with continuous integration (CI). Ensuring coding standards are respected with linters.
- Nikita Holyev - Theorist: Provide theoretical background support and potential usage cases for the software.
- Dr. Spencer Smith - Supervisor. Reviewing all documents.
- Cynthia Liu - Domain Expert: Reviewing all documents.

- Kim Ying Wong - Secondary reviewer of SRS.
- Morteza Mirzaei - Secondary reviewer of VnV Plan.
- Nada Elmasry - Secondary reviewer of MG and MIS.
- Several volunteers as subjects for usability (future) and maintainability testing, see 4.2 for detailed procedures. They are recruited from class as well as potential users of the software.

3.2 SRS Verification Plan

The (SRS) is reviewed by Cynthia Liu, Kim Ying Wong and Dr. Spencer Smith to ensure that the requirements are clear with good traceability, and the models are represented to be easily understandable. The SRS Checklist can be used as reference. Feedback will be given as GitHub Issues, and the developer will update the document accordingly.

The developer will also go-through the models in the SRS with Nikita Holyev to ensure that they are mathematically correct and reflect the current direction of work in the theoretical side.

3.3 Design Verification Plan

The design documents, including the Module Guide (MG) and the Module Interface Specification (MIS), is reviewed by Cynthia Liu, Nada Elmasry and Dr. Spencer Smith to ensure that the design is unambiguous, inline with module design best-practices and consistent with the requirements in the SRS. The MG Checklist and MIS Checklist can be used as references. Feedback will be given as GitHub Issues, and the developer will update the documents accordingly.

Discussions will also be held with Nikita Holyev to identify potential usage cases that may require future proofing in the design.

3.4 Verification and Validation Plan Verification Plan

The VnV plan is reviewed by Cynthia Liu, Morteza Mirzaei and Dr. Spencer Smith to ensure the plan reflects requirements in the SRS with clear and specific test cases. The VnV Checklist can be used as reference. Feedback

will be given as GitHub Issues, and the developer will update the document accordingly.

3.5 Implementation Verification Plan

Successful implementation of the synthetic eddy method can be verified by automated tests detailed in Section 4.1. These tests will be run during development on local machines, and before merging to the main branch with GitHub Actions.

To ensure the quality of the codebase, the Source Code Checklist provided by Dr. Smith will be used as a guideline throughout the development process.

3.6 Automated Testing and Verification Tools

- Continuous Integration (CI): GitHub Actions to run automated tests upon pull request to main branch.
- System and Unit Tests: pytest.
- Code Coverage: pytest.
- Linters: flake8: ensure PEP8 coding standards are respected.

3.7 Software Validation Plan

Validation against real-life experiments is outside the scope of this project. The basis of SynthEddy is theoretical. The focus is to ensure that the software properly represent the theoretical proposals [SRS: TM1, TM2 and GD1].

4 System Test Description

The system tests are divided into two main categories: Functional and Nonfunctional Requirements. The Functional Requirements, other than the CFD interface, are tested with automated scripts, while the Nonfunctional Requirements are tested manually. The tests are traced to the requirements in the SRS.

4.1 Tests for Functional Requirements

These tests are traced to R1-R3 (Section 5.1) of the SRS. Only test for R3 is manual, as it involves interfacing with CFD software. The rest are automated with pytest and GitHub Actions.

Areas of testing are structured with respect to the list of Function Requirements.

4.1.1 Input Test

Check Query Point within Flow Field [R1]

1. test-query-coordinate

   Control: Automatic

   Initial State: Empty flow field with size (1m, 1m, 1m)

   Input: $\mathbf{x} = (2, 2, 2)$

   Output: Throw exception: Queried point outside of flow field!

   Test Case Derivation: Execution should be halted as results outside of the flow field are not meaningful.

   How test will be performed: Automated test script, GitHub Actions.

4.1.2 Eddy Generation Test

Generation of a Single Eddy [R2]

1. test-eddy-shape

   Control: Automatic

   Initial State: Empty flow field, zero $\mathbf{u_{avg}}$

   Input:

   - Profile: 1 eddy with predetermined center location
     $\mathbf{x} = (0, 0, 0)$ $\sigma = 1$ (length-scale, i.e. radius)
   - Query: Pairs of points relative to center:
     $\mathbf{x_1} = (0.5, 0.5, 0.5)$ $\mathbf{x_2} = (-0.5, -0.5, -0.5)$
     $\mathbf{x_3} = (2, 2, 2)$ $\mathbf{x_4} = (-2, -2, -2)$

   Output: Velocity vectors at each point: $\mathbf{u_1}$, $\mathbf{u_2}$, $\mathbf{u_3}$, $\mathbf{u_4}$
   Expected behavior:

   - $|\mathbf{u_1} - \mathbf{u_2}| < \text{tol}|\mathbf{u_1}|$ $|\mathbf{u_3}| = |\mathbf{u_4}| = 0$

   Test Case Derivation: Points should have equal and opposite velocities inside the eddy, and zero velocity outside. This is a result of the shape function cutting off the influence of each eddy beyond σ.

   How test will be performed: Automated test script, GitHub Actions.

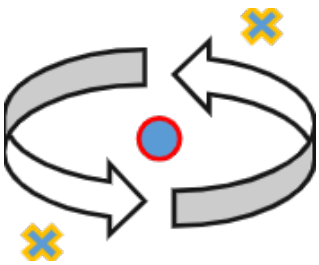

Figure 2: Query Pair of Points

2. test-eddy-std

Control: Automatic

Initial State: Empty flow field, zero $\mathbf{u_{avg}}$

Input:

- Profile: 1 eddy with predetermined orientation
  $\mathbf{x} = (0, 0, 0)$ $\sigma = 1$
  $\alpha = (0, 0, 1)$ (intensity = 1, orientation along z-axis)
- Query: Mesh grid of points covering the eddy

Output: Velocity vectors at each point $\mathbf{u}$
Expected behavior:

- $\text{std}(\mathbf{u_z}) < \text{tol}$

Test Case Derivation: Flow of an eddy should be around its orientation axis. There should not be velocity in the axial direction. Thus, velocity standard deviation along the eddy orientation should be zero (below tolerance).

How test will be performed: Automated test script, GitHub Actions.

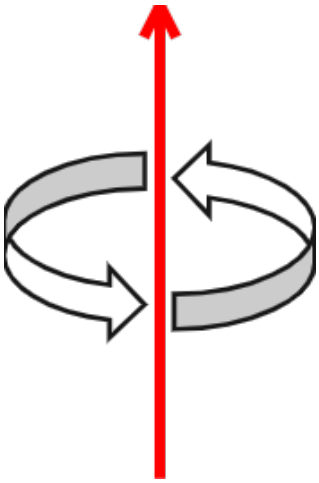

Figure 3: Eddy Orientation

4.1.3 Field Genration Test

Generation of Flow Field [R2]

1. test-field-mean

   Control: Automatic

   Initial State: Empty flow field, $\mathbf{u_{avg}}$

   Input:

   - Profile: Any number of eddies, random orientations
   - Query: Mesh grid of points of the entire flow field at any t

   Output: Velocity vectors at each point: $\mathbf{u}$
   Expected behavior:

   - $|\frac{\sum \mathbf{u}-\mathbf{u_{avg}}}{N_{grid\ \ points}}| < \text{tol}$

   Test Case Derivation: The vector mean of all velocity fluctuations caused by the eddies should be zero (below tolerance). For eddies completely within the field, this should be inherently true as velocity moves in circles. For eddies partially outside the field boundary, they should be wrapped around to the opposite side of the field to ensure this property.

   How test will be performed: Automated test script, GitHub Actions.

2. test-field-std-time

   Control: Automatic

   Initial State: Empty flow field, $\mathbf{u_{avg}}$

   Input:

   - Profile: Any number of eddies, random orientations
   - Query: Mesh grid of points of the entire flow field with $t_0 = 0$ and $t_1 = 10$.

   Output: Velocity vectors at each point: $\mathbf{u}$
   Expected behavior:

- $\text{std}(\mathbf{u})|_{t_0} - \text{std}(\mathbf{u})|_{t_1} < \text{tol}$

Test Case Derivation: Velocity standard deviation should stay the same regardless of time, as the eddies are only moved to different locations in the field.

How test will be performed: Automated test script, GitHub Actions.

3. test-field-std-intensity

Control: Automatic

Initial State: Empty flow field, $\mathbf{u_{avg}}$

Input:

- Profile: Any number of eddies, random orientations
  Increasing eddy intensity $|\alpha|$
- Query: Mesh grid of points of the entire flow field

Output: Velocity vectors at each point: $\mathbf{u}$
Expected behavior:

- if $|\alpha|_1 > |\alpha|_2$ then $\text{std}(\mathbf{u})_1 > \text{std}(\mathbf{u})_2$

Test Case Derivation: Velocity standard deviation should increase with eddy intensity $|\alpha|$, as the velocity fluctuations are stronger with higher intensity.

How test will be performed: Automated test script, GitHub Actions.

4.1.4 Divergence Test

Divergence Free Condition [R2]

1. test-field-divergence

Control: Automatic

Initial State: Empty flow field, $\mathbf{u_{avg}}$

Input:

- Profile: Any number of eddies, random orientations

- Query: Mesh grid of points of the entire flow field

Output: Velocity vectors at each point: $\mathbf{u}$
Expected behavior:

- $\text{div}(\mathbf{u}) < \text{tol}$

Test Case Derivation: The flow field should be divergence-free (below tolerance). Divergence measures if a vector field satisfies the conservation of mass, i.e. nothing is created or destroyed. See numpy.gradient for more information regarding its calculation.

How test will be performed: Automated test script, GitHub Actions.

4.1.5 CFD Interface

Feed IC and BC to CFD Software [R3]

1. test-cfd-interface

   Type: Manual.

   Initial State: Empty flow field, $\mathbf{u_{avg}}$

   Input:

   - Profile: Any number of eddies, random orientations
   - Query: CFD software should query points in the flow field

   Output: Velocity vectors at each point: $\mathbf{u}$
   Expected behavior:

   - They should be feed into the CFD software

   Test Case Derivation: The CFD software should be able get the flow field generated by SynthEddy as IC and BC.

   How test will be performed: Manually perform a CFD simulation in conjecture with SynthEddy.

4.2 Tests for Nonfunctional Requirements

NFR2 is already covered by functional test of CFD interface (4.1.5). NFR1 (accuracy), NFR3 (maintainability) NFR4(portability) and NFR5 (performance) will be verified by tests detailed below.

4.2.1 Accuracy

Provide Realistic Profile Before Generation

Test case for this requirement will be added in the future given more theoretical background.

4.2.2 Maintainability

Modifying Shape Function

In terms of maintainability, users who looks to modify SynthEddy would most likely want to change the shape functions of the eddies to fit their specific needs. If the software is well structured and documented, they should be able to do so with ease, even without systematic programming knowledge.

The test subjects are not expected to be familiar with the theoretical background of the synthetic eddy method, as they will be given a valid mathematical equation of the shape function. They should have at least some basic programming exposures, such as using MATLAB or Python for simple calculations or data processing.

1. test-modify-shape-func

   Type: Static

   Initial State: Reviewer is provided with documentation and source code, and no other help.

   Input/Condition: Reviewer is given a mathematical equation for a new shape function.

   Output/Result: Reviewer should be able to modify the source code to implement the new shape function.

   How test will be performed: Performed by volunteers or potential users of the software. Time taken will be recorded. Check if the modified software still passes the automated functional tests.

4.2.3 Portability

Cross-platform Test

Since unit tests and system tests executed by GitHub Actions are performed on Ubuntu runners (see here), this already covers installing and running on Linux.

To test Windows and MacOS, an additional GitHub Action workflow is created to install and run a test of the Main Control Module on Windows and MacOS runners (see here). The details of this test file can be found in Section 5.1.1. The only reason for choosing this specific test is that it is fast to perform, as MacOS runners are billed 10 times as expensive as Linux runners.

4.2.4 Performance

Computing Time and Memory Usage

Test case in this section mimics real-life usage. The test is not meant to execute until finish as it would take too long. Instead, it is used to give a rough estimate of the computing time and memory usage.

1. test-performance

   Type: Manual

   Initial State: SynthEddy is installed and ready to run.

   Input:

   - Field: $20 \times 20 \times 20$ field with around 10 million eddies
   - Query: $1000 \times 1000 \times 1000$ meshgrid

   Output: Memory usage and estimated computing time

   How test will be performed: Developer will run the test on their local machine while monitor task manager and command line output. The program will give an estimate of the computing time as it computes the velocities in the field. The memory usage by the program reported by task manager will be taken when the program is running.

4.3 Traceability Between Test Cases and Requirements

	R1	R2	R3	NFR1	NFR2	NFR3	NFR4	NFR5
Test 4.1.1	X							
Test 4.1.2		X						
Test 4.1.3		X						
Test 4.1.4		X						
Test 4.1.5			X		X			
Test 4.2.1				X				
Test 4.2.2						X		
Test 4.2.3							X	
Test 4.2.4								X

Table 1: Traceability Between Test Cases and Requirements

5 Unit Test Description

Unit tests are written for the modules described in the MG and MIS. For most modules, a dedicated test file is created, unless that module cannot function independently and can be sufficiently tested when other modules are tested.

Each module is tested for the expected use cases, edge cases and exceptions. These tests are heavily leveraged during development to ensure that changes in one module do not break the functionality of another.

All unit tests that are not designated as "slow" in the test file are run by GitHub Actions upon pull request to the main branch.

5.1 Unit Testing Scope

Unit tests are performed on all modules developed for SynthEddy, except:

- Eddy Module [MG: M6]: The inputs to this module need to be generated by the Flow Field Module. Thus, the Eddy Module will be tested when the Flow Field Module is tested.

- Visualization Module [MG: M10]: This module is only a placeholder at current stage. However, it is still covered by the test cases in the Query Interface Module.
- Utils: Not actually a module in the design, but several helper functions to reduce code repetition. These functions are tested when the modules that use them are tested.

5.1.1 Main Control Module [MG: M2]

Test for creating a new field and query an existing field, setting active shape functions and cutoff, as well as handling exceptions raised by lower-level modules. See test_main.py

1. test-main-new
2. test-main-query
3. test-main-new-exception
4. test-main-query-field-not-exist
5. test-main-query-shape-function-exceptions
6. test-main-query-cutoff
7. test-main-query-exception

5.1.2 Query Interface Module [MG: M3]

Test for query in meshgrid and points modes, as well as handling exceptions when the query is invalid. See test_query.py

1. test-query-meshgrid
2. test-query-points
3. test-query-meshgrid-exceptions
4. test-query-points-exceptions
5. test-query-mode-exceptions
6. test-query-plot-exceptions

5.1.3 Eddy Profile Module [MG: M4]

Test for loading valid and invalid eddy profiles (exceptions). See test_eddy_profile.py

1. test-eddy-profile
2. test-eddy-profile-invalid

5.1.4 Flow Field Module [MG: M5]

Test initializing field and calculating velocities with meshgrid. Eddy generation is also tested here. Exceptions during initialization and calculation are also tested. See test_flow_field.py

1. test-eddy-generation
2. test-flow-field
3. test-flow-field-wrap
4. test-flow-field-non-uniform-x-vel
5. test-flow-field-init-exceptions
6. test-flow-field-mesh-exceptions
7. test-flow-field-set-exceptions
8. test-flow-field-out-of-memory

5.1.5 Shape Function Module [MG: M7]

Test for calculating shape function values with different active shape functions and cutoffs, as well as exceptions when setting active shape function or cutoff. See test_shape_function.py

1. test-shape-function
2. test-shape-function-exceptions

5.1.6 File IO Module [MG: M8]

Test for reading, writing and deleting files within the program directory, as well as handling exceptions when the file is not found, unable to read/write or the expected format is not matched. See test_file_io.py

1. test-file-io
2. test-file-io-read-fail
3. test-file-io-read-fail-format
4. test-file-io-write-fail
5. test-file-io-clear-fail

5.2 Traceability Between Test Cases and Modules

	M2	M3	M4	M5	M6	M7	M8	M10
Test 5.1.1	X							
Test 5.1.2		X						X
Test 5.1.3			X					
Test 5.1.4				X	X			
Test 5.1.5						X		
Test 5.1.6							X	

Table 2: Traceability Between Unit Test Cases and Modules

References

Ruggero Poletto, T. Craft, and Alistair Revell. A new divergence free synthetic eddy method for the reproduction of inlet flow conditions for les. *Flow, Turbulence and Combustion*, 91:1–21, 10 2013. doi: 10.1007/s10494-013-9488-2.

6 Appendix

6.1 Symbolic Parameters

The definition of the test cases will call for SYMBOLIC_CONSTANTS. Their values are defined in this section for easy maintenance.

- Tolerance: some small value, e.g. tol = 0.0001
- Average velocity: along x-axis (1D flow), e.g. $\mathbf{u_{avg}} = (5, 0, 0)$