

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

Phil Du (Software)
Nikita Holyev (Theory)

April 15, 2024

Revision History

Date	Version	Notes
2024-02-19	1.0	Initial Plan After SRS

Contents

1	Symbols, Abbreviations, and Acronyms	iv
2	General Information	1
2.1	Summary	1
2.2	Objectives	1
2.3	Relevant Documentation	2
3	Plan	2
3.1	Verification and Validation Team	2
3.2	SRS Verification Plan	3
3.3	Design Verification Plan	3
3.4	Verification and Validation Plan Verification Plan	3
3.5	Implementation Verification Plan	4
3.6	Automated Testing and Verification Tools	4
3.7	Software Validation Plan	4
4	System Test Description	5
4.1	Tests for Functional Requirements	5
4.1.1	Input Test	5
4.1.2	Eddy Generation Test	6
4.1.3	Field Genration Test	8
4.1.4	Divergence Test	9
4.1.5	CFD Interface	10
4.2	Tests for Nonfunctional Requirements	11
4.2.1	Accuracy	11
4.2.2	Maintainability	11
4.2.3	Performance	12
4.3	Traceability Between Test Cases and Requirements	13
5	Unit Test Description	13
5.1	Unit Testing Scope	13
5.2	Tests for Functional Requirements	14
5.2.1	Module 1	14
5.2.2	Module 2	15
5.3	Tests for Nonfunctional Requirements	15
5.3.1	Module ?	15

5.3.2	Module ?	16
5.4	Traceability Between Test Cases and Modules	16
6	Appendix	17
6.1	Symbolic Parameters	17

List of Tables

1	Traceability Between Test Cases and Requirements	13
---	--------------------------------------------------	----

List of Figures

1	System Context	1
2	Query Pair of Points	6
3	Eddy Orientation	7

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 provide 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.

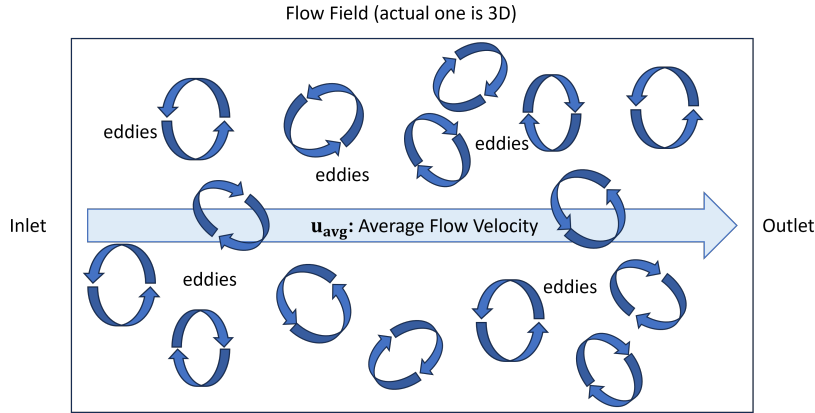


Figure 1: System Context

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 for automated tests.
- 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 Non-functional 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-R5 (Section 5.1) of the SRS [Du \(2024\)](#). With R5 being the only Function Requirement that needs to be tested manually, the rest will be automated.

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.

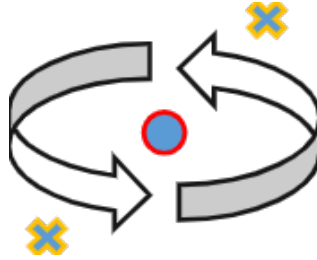


Figure 2: Query Pair of Points

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.

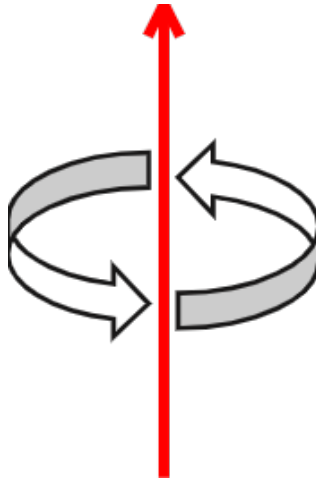


Figure 3: Eddy Orientation

4.1.3 Field Generation 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:

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

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 and NFR4 is already covered by functional test of CFD interface (4.1.5). NFR1 (accuracy), NFR3 (maintainability) 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 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		X	
Test 4.2.1				X				
Test 4.2.2						X		
Test 4.2.3								X

Table 1: Traceability Between Test Cases and Requirements

5 Unit Test Description

[This section should not be filled in until after the MIS (detailed design document) has been completed. —SS]

[Reference your MIS (detailed design document) and explain your overall philosophy for test case selection. —SS]

[To save space and time, it may be an option to provide less detail in this section. For the unit tests you can potentially layout your testing strategy here. That is, you can explain how tests will be selected for each module. For instance, your test building approach could be test cases for each access program, including one test for normal behaviour and as many tests as needed for edge cases. Rather than create the details of the input and output here, you could point to the unit testing code. For this to work, your code needs to be well-documented, with meaningful names for all of the tests. —SS]

5.1 Unit Testing Scope

[What modules are outside of the scope. If there are modules that are developed by someone else, then you would say here if you aren't planning on verifying them. There may also be modules that are part of your software,

but have a lower priority for verification than others. If this is the case, explain your rationale for the ranking of module importance. —SS]

5.2 Tests for Functional Requirements

[Most of the verification will be through automated unit testing. If appropriate specific modules can be verified by a non-testing based technique. That can also be documented in this section. —SS]

5.2.1 Module 1

[Include a blurb here to explain why the subsections below cover the module. References to the MIS would be good. You will want tests from a black box perspective and from a white box perspective. Explain to the reader how the tests were selected. —SS]

1. test-id1

Type: [Functional, Dynamic, Manual, Automatic, Static etc. Most will be automatic —SS]

Initial State:

Input:

Output: [The expected result for the given inputs —SS]

Test Case Derivation: [Justify the expected value given in the Output field —SS]

How test will be performed:

2. test-id2

Type: [Functional, Dynamic, Manual, Automatic, Static etc. Most will be automatic —SS]

Initial State:

Input:

Output: [The expected result for the given inputs —SS]

Test Case Derivation: [Justify the expected value given in the Output field —SS]

How test will be performed:

3. ...

5.2.2 Module 2

...

5.3 Tests for Nonfunctional Requirements

[If there is a module that needs to be independently assessed for performance, those test cases can go here. In some projects, planning for nonfunctional tests of units will not be that relevant. —SS]

[These tests may involve collecting performance data from previously mentioned functional tests. —SS]

5.3.1 Module ?

1. test-id1

Type: [Functional, Dynamic, Manual, Automatic, Static etc. Most will be automatic —SS]

Initial State:

Input/Condition:

Output/Result:

How test will be performed:

2. test-id2

Type: Functional, Dynamic, Manual, Static etc.

Initial State:

Input:

Output:

How test will be performed:

5.3.2 Module ?

...

5.4 Traceability Between Test Cases and Modules

[Provide evidence that all of the modules have been considered. —SS]

References

Phil Du. System requirements specification. <https://github.com/omltcat/turbulent-flow/blob/main/docs/SRS/SRS.pdf>, 2024.

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. $\text{tol} = 0.0001$
- Average velocity: along x-axis (1D flow), e.g. $\mathbf{u}_{\text{avg}} = (5, 0, 0)$