System Verification and Validation Plan for SubLiMat

Uriel Garcilazo Cruz

April 1, 2025

Revision History

Date	Version	Notes
Feb. 15th	1.0	First version

Contents

1 Symbols, Abbreviations, and Acronyms

The following tables are extracted from the program's SGS documentation (Garcilazo-Cruz, 2025). The tables are presented here for easy reference and cross-validation.

symbol	**unit**	**HGVS**
*qa	alignment quality	fundamental unit of alignment quality
bp	base pair	fundamental unit of genetic sequence length
Kb	kilobase unit	one thousand base pairs

symbol	**unit**	**description**
F	–	Comparative alignment between two sequences
g	qa	penalty associated with a gap in the alignment of two sequences
N_A	bp	Adenine nitrogenous base, element of the purine family of nucleotides with an amine group in Carbon 6 of its pyrimidine ring
N_C	bp	Cytosine nitrogenous base, element of the pyrimidine family of nucleotides with a no methylated carbons making part of its pyrimidine ring
N_G	bp	Guanine nitrogenous base, element of the purine family of nucleotides with an amine group on Carbon 2 and a carbonyl group on Carbon 6 of its pyrimidine ring
N_T	bp	Tymine nitrogenous base, element of the pyrimidine family of nucleotides with a methyl group in Carbon 5 of its pyrimidine ring
Q_{AB}	qa	A collection of base pairs representing a genetic sequence to be compared with another sequence SEQ_A
S	qa	Substitution matrix used to score the alignment of two sequences
SEQ_A	Kb	A collection of base pairs representing a genetic sequence to be compared with another sequence SEQ_B
SEQ_B	Kb	A collection of base pairs representing a genetic sequence to be compared with another sequence SEQ_A

SNP	bp	single nucleotide polymorphism; variation in a single base pair in DNA sequence
T_I	qa	A transition occurrying between nucleotides of the same nitrogenous base families
T_V	qa	A transition occurrying between nucleotides of different nitrogenous base families

This document serves as a systematic point of reference for the SubLiMat software verification and validation process, with the goal of thoroughly documenting the verification and validation goals of the scientific program. The reader should refer to the SRS documentation for a detailed description of the software's theoretical framework, requirements and functionality. The document begins with the General information section 2, followed by a verification plan and a description of the system tests **??**. The document concludes with a unit test description **??**

2 General Information

2.1 Summary

This document reviews the verification and discusses the validation for the Substitution Matrix Benchmarking with Pariwise Sequence alignment (SubLiMat) software. SubLiMat is a program that describes the quality of genetic sequence alignments across different substitution matrices.

2.2 Objectives

The primary objective of the presented documentation is to provide a framework that enables a comparative and structured analysis on the confidence of the SubLiMat software correctness, while also exposing potential boundaries in the software's usability, and demonstrating examples of adequate usability, helping build expectations and giving awareness of potential caveats. Some objectives are out of the scope of the program's verification and validation analysis, at least in the documentation's first version, due to the limited amount of resources available for the project, including but not restricted to hardware and logistic constraints. The objectives outside of the scope of this documentation include a comprehensive testing of the software's validity given the lack of domain experts available. The verification and validation document will not include a comprehensive comparison with other software, due to the fundamental difference in the software's approach to solving sequence alginments via heuristic algorithms, whereas SubLiMat assumes the identification of the global optimum. Lastly, the verification of large datasets, including genomic-sized data, are out of the scope of the current testing protocol, as is is assumed the software works with coding genetic sequences

commonly found in molecular biology.

2.3 Challenge Level and Extras

The scope of verification and validation for SubLiMat is considered to be basic, as the software is a scientific program that does not require a high level of complexity in its verification and validation process, and there's a clearly defined set of validation tools that serve as a ground-truth to test the performance and accuracy of the software.

Additional (extra) features tackled by the project include the implementation of a verification software, which is assumed to mutate a sequence SEQ_A from sequence SEQ_B to generate the real alignment and score, which can then be compared with the alignment and score generated by the SubLiMat software.

2.4 Relevant Documentation

The relevant documentation for SubLiMat includes the Software Requirements Specification Garcilazo-Cruz (2025), the Module Guide (MG) and the Problem Statement. These elements can be found in the GitHub repository for the project, and are available for consultation at any time.

The SRS document provides a detailed description of the software's requirements, including the functional and non-functional requirements, and the theoretical framework that supports the software's functionality. The MG document provides a detailed description of the software's architecture, including the modules that compose the software, and the relationships between them. The Problem Statement provides a brief overview of the software's purpose and functionality, and the motivation behind the software's development.

3 Plan

The verification and validation plan for SubLiMat is divided into several sections. First, there is a description of the verification and validation team, where the roles of each team member are defined. Next, there is a description of the SRS verification plan, which includes the approaches that will be used to verify the software's requirements (both functional and non-functional).

Then there is a description of the design verification plan and the implementation verification plan, which describe the decisions and procedures that will be made to deploy testing.

3.1 Verification and Validation Team

Name	Document	Role	Description
Dr. Spencer Smith	All	Instructor	Reviews documentation correctness and adherence to verification standards
Uriel Garcilazo Cruz	All	Author	Creator of the documentation, including SGS, MG1 and VnV
Junwei Lin	All	Domain Expert	Reviews documentation content and ensures the validity in the theory supported by each document

3.2 SRS Verification Plan

The documentation for SubLiMat will be reviewed by the team members, including the author, using the following recipe:

1. The author will review the documentation for correctness, adherence to verification and alignment to documentation standards.

2. A new issue will be created for each version of the documentation on GitHub, and shared with the collaborators Dr. Spencer Smith and Junwei Lin.

3. The domain expert will review the documentation for correctness and ensure the validity of the theory supported by each document.

4. The instructor will review the documentation for correctness and alignment with documentation standards.

5. Author will address the issues and make the necessary changes to the documentation.

3.3 Design Verification Plan

The design verification plan for SubLiMat will be reviewed by the team members, including the author, using a dynamic verification approach by the main author Uriel Garcilazo-Cruz, and assisted in the verification and test accuracy by the domain expert Junwei Lin. The instructor Dr. Spencer Smith will review the documentation to determine correctness and adherence to verification standards.

3.4 Verification and Validation Plan Verification Plan

The verification and validation plan for SubLiMat will be reviewed systematically through the following process:

1. The author (Uriel Garcilazo Cruz) will create and validate the initial VnV plan.
2. A new issue will be created on GitHub for each version of the VnV plan.
3. Domain expert (Junwei Lin) will review the documentation and provide feedback through GitHub issues
4. Dr. Spencer Smith will conduct the final review of the VnV plan as the instructor.
5. The author will address all issues and incorporate suggested changes.
6. All reviews will be guided by the VnV Checklist (VnV-Checklist.pdf).

The progress of this review process will be tracked through GitHub issues, allowing for transparent communication and documentation of all feedback and revisions.

3.5 Implementation Verification Plan

To verify SubLiMat's implementation, the team will follow the following steps:

1. Static verification:

- Author (Uriel Garcilazo-Cruz) will validate and create the first version of the code.
- Using pylint to check for code quality and adherence to coding standards in the programming language Python.

2. Dynamic verification:

   - The test cases indicated in section 5 will be used to verify the implementation of the software.
   - Using pytest to carry out the tests and verify the software's functionality.

3.6 Automated Testing and Verification Tools

The automated testing will take place using the pytest framework, which will be used to run the test cases in combination with continuous integration allowed by GitHub Actions. The test cases will be written in Python, and will be used to verify the software's functionality.

3.7 Software Validation Plan

A validation plan for SubLiMat is beyond the scope of this document, given the limited amount of domain experts available to perform such validation, and the time available to complete the project's verification.

4 System Tests

4.1 Tests for Functional Requirements

In agreement with the functional requirements stated in the SRS document Garcilazo-Cruz (2025), the following tests will be performed to verify the software's functionality. R1 amd R2 functional requirements are related to the SubLiMat inputs, and R3, R4 and R5 are related to the software's outputs.

4.1.1 Input Tests: Valid Inputs

The SRS document Garcilazo-Cruz (2025) refers to the functional requirement R1 as the main descriptor on the software's inputs, and R2 for the

construction of a comparative matrix used in Needleman and Wunsch (1970).

ID	Input (bp)		Output	
	SEQ_A	SEQ_B	valid?	Error Message
TC-SubLiMat-1-1	ATCG	ATCG	Y	NONE
TC-SubLiMat-1-2	" "	ATCG	N	Sequence length bigger than zero
TC-SubLiMat-1-3	ATCG	" "	N	Sequence length bigger than zero
TC-SubLiMat-1-4	" "	" "	N	Sequence length bigger than zero
TC-SubLiMat-1-5	ATCG	A_CG	Y	NONE
TC-SubLiMat-1-6	A	ATCGGG	Y	NONE
TC-SubLiMat-1-7	A	T	Y	NONE
TC-SubLiMat-1-8	RTGA	ATGA	N	Sequence contains non DNA symbol

Table 2: Test case inputs and outputs

1. Functional Input Tests - Properties of genetic sequence data (R1)

   Control: Automatic

   Initial State: Hyperparameter defined during runtime

   Input: Inputs from Table 2

   Output: Return an error message with a detailed description of the sequence(s) creating the issue, or a list of numbers representing the alignment score(s) for the pairwise sequence alignment(s) of the input sequences SEQ_A and SEQ_B.

   Test Case Derivation: Evaluates the software's ability to handle different types of input data that represent genetic sequences, including valid and invalid sequences, and sequences with different lengths. The system returns error messages when the physical constrains of the data are violated, in cases where the lenght of the sequence is zero, or when the sequence contains non-DNA symbols.

   How test will be performed: Automatically using the pytest framework.

2. Functional Input Tests - Input Tests: Valid matrices of F(R2)

ID	Input (bp)		Output	
	$SEQ^{n}_{A_k=1}$	$SEQ^{m}_{B_k=1}$	Valid?	Error Message
TC-SubLiMat-2-1	$n = 10$	$m = 10$	Y	NONE
TC-SubLiMat-2-2	$n = 100$	$m = 10$	Y	NONE
TC-SubLiMat-2-3	$n = 10$	$m = 100$	Y	NONE
TC-SubLiMat-2-4	$n = 500$	$m = 500$	Y	NONE
TC-SubLiMat-2-5	$n = 5,000$	$m = 5,000$	Y	NONE
TC-SubLiMat-2-6	$n \geq 5,001$	$m \geq 5,001$	N	Sequence length exceeds maximum

Table 3: Test cases for sequence length validation

Control: Automatic

Initial State: Parameter defined during runtime

Input: Valid inputs from Table 2 with cases taken from elements from Table 5

Output: Return a matrix with the alignment scores for the pairwise sequence alignment of the input sequences SEQ_A and SEQ_B.

Test Case Derivation: Evaluates the software's ability to construct a comparison matrix for the alignment of two genetic sequences and returns a null value of N/A for matrices of size greater than 5,000 elements.

How test will be performed: Automatically using the pytest framework.

3. Functional Input Tests - Functional Input Tests - Input Tests: Valid matrices of S (R3)

ID	Input Matrix	Expected Output	
	Dimensions & Content	Valid?	Error Message
TC-SubMat-3-1	$n \times n, n = 4$	Y	NONE
TC-SubMat-3-2	$n \times m, n = 4, m = 5$	N	Matrix must be square
TC-SubMat-3-3	$n \times n, n = 4$, non-numeric	N	Matrix entries must be numeric
TC-SubMat-3-4	$n \times n, n = 3$	N	Matrix must be 4x4 for nucleotides
TC-SubMat-3-5	$n \times n, n = 4$ asymmetric	N	Matrix must be symmetric

Table 4: Test cases for substitution matrix validation. A valid substitution matrix must be square with dimensions 4x4 for nucleotides, contain only numeric values, and be symmetric for reciprocal comparisons between elements.

Control: Automatic

Initial State: Hyperparameter defined by the program

Input: Inputs from Table 4

Output: Return a boolean value indicating the validity of the input matrix S.

Test Case Derivation: Evaluates the software's ability to validate the input substitution matrix S for the alignment of two genetic sequences, ensuring the matrix is square, contains only numeric values, and is symmetric.

How test will be performed: Automatically using the pytest framework.

4.1.2 Output Tests: Coherence in the alignment

1. Functional Output Tests - Coherence in the alignment (R4 & R5)

Control: Automatic

Initial State: Pending output conditional to valid inputs

Input: Set of calculated alignment scores given a matrix F

Output: An error message that appears when the maximum length of a sequence as input is reached, and nothing otherwise. SEQ_A and SEQ_B.

Test Case Derivation: Evaluates the software's ability to process alignment scores, and build a comparison matrix F in a reasonable amount of time.

How test will be performed: Automatically using the pytest framework.

4.2 Tests for Nonfunctional Requirements

As described in SubLiMat SRS document Garcilazo-Cruz (2025), there are five non-functional requirements that will be tested in the software. The non-functional requirements for accuracy will just reference the appropriate functional tests from above.

4.2.1 Nonfunctional: Performance (NFR5 in Garcilazo-Cruz (2025))

Performance

1. Nonfunctional Tests: Performance

ID	Input (bp)		Expected Performance	
	$SEQ^n_{A_k=1}$	$SEQ^m_{B_k=1}$	Matrix Size	Est. Time (s)
TC-SubLiMat-4-1	$n = 10$	$m = 10$	100	< 0.1
TC-SubLiMat-4-2	$n = 100$	$m = 10$	$1,000$	< 1.0
TC-SubLiMat-4-3	$n = 10$	$m = 100$	$1,000$	< 1.0
TC-SubLiMat-4-4	$n = 500$	$m = 500$	$250,000$	~ 25
TC-SubLiMat-4-5	$n = 5,000$	$m = 5,000$	$25,000,000$	> 300
TC-SubLiMat-4-3	$n \geq 5,001$	$m \geq 5,001$	N/A	< 0.1

Table 5: Test cases for estimated time of completion given the size of the comparison matrix

Initial State: Pending output conditional to valid inputs

Input: Set of calculated alignment scores given a matrix F

Output: A float value representing the amount of time taken to calculate the alignment scores for the input sequences SEQ_A and SEQ_B.

Test Case Derivation: Evaluates the software's ability to execute, in a commercial computer, the alignment of two genetic sequences SEQ_A and SEQ_B.

How test will be performed: Automatically using the pytest framework.

4.2.2 Nonfunctional: Portability (NFR4 in Garcilazo-Cruz (2025))

Portability

1. Nonfunctional Tests: Portability

ID	Operating System	Expected Result
TC-SCEC-5-1	Windows	Pass
TC-SCEC-5-2	Linux	Pass
TC-SCEC-5-3	macOS	Pass

Table 6: Test cases for system compatibility across different platforms

Initial State: Fresh system installation with no prior software components

Input: Software installation package

Output: Successful running system over all platforms to verify portability of the software, and summarized in a table.

Test Case Derivation: Evaluates the software's ability to install and run consistently across different operating systems while maintaining functionality and user experience.

How test will be performed: Code developer (Uriel Garcilazo Cruz) will try to install and run whole software in different operating systems.

4.2.3 Nonfunctional: Usability (NFR2 in Garcilazo-Cruz (2025))

Usability

No.	Question	Answer
1.	How many cores does your computer have?	
2.	Are you familiar with the use of Python?	
3.	Are the manual specifications easy to follow?	
4.	Did you have Python installed in your machine before?	
5.	Did you encounter any issues running the file from terminal?	
6.	Did you encounter any issues loaading your genetic sequences?	
7.	Is the output table easy to understand?	

Table 7: TC-SCEC-5 - Usability test survey

1. Nonfunctional Tests: Usability

   Type: Manual

   Input: None

   Initial State: None

   Output: Survey with user responses

   Test Case Derivation: Evaluates how easy it is for the user to interact with the software, and how intuitive the software is to use.

   How test will be performed: An online survey will be created and shared with potential users of the software, including domain experts and, if possible, students in the field of molecular biology or collaborators that meet the user specifications found in the SRS document Garcilazo-Cruz (2025).

4.3 Traceability Between Test Cases and Requirements

	R1	R2	R3	R4/R5	NFR2	NFR4	NFR5
TC-SubLiMat-1	X						
TC-SubLiMat-2		X					
TC-SubMat-3			X				
TC-SubLiMat-4				X			X
TC-SCEC-5					X	X	

Table 8: Traceability between test cases and requirements

5 Unit Test Description

The program SubLiMat is composed of the following modules:

- **constants.py**: Ensures proper integration with arguments received from user and initializes constants and hyperparameters in the program
- **pairwise_alignment.py**: Responsible of constructing the comparison matrix F and implementing the Needleman-Wunsch algorithm
- **main.py**: Integrates calculation of alignment scores over all substitution matrices and handles formatting

5.1 Unit Testing Scope

The unit testing for SubLiMat will focus on the following modules:

- **pairwise_alignment.py**

The core of the algorithm is implemented in the **pairwise_alignment.py** module, which is responsible for constructing the comparison matrix F and implementing the Needleman-Wunsch algorithm. This module is the most critical part of the software, and the unit tests will focus on verifying the correctness of the alignment scores calculated by the algorithm.

The modules **constants.py** and **main.py** will not be tested in this document, as they are not critical to the software's functionality, serving in the initialization and calling of the algorithm, respectively.

5.2 Tests for Functional Requirements

All of the tests for the functional requirements are described in the section subsection 4.1, and

5.2.1 Module 1

5.3 Tests for Nonfunctional Requirements

Tests for nonfunctional requirements are beyond the scope of the unit testing for SubLiMat.

References

Uriel Garcilazo-Cruz. System requirements specification. https://github.com/UGarCil/UGarcil_capstone/tree/main/docs/SRS, 2025.

Saul B. Needleman and Christian D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3):443–453, 1970. ISSN 0022-2836. doi: https://doi.org/10.1016/0022-2836(70)90057-4. URL https://www.sciencedirect.com/science/article/pii/0022283670900574.