System Verification and Validation Plan for MPIR

Xunzhou (Joe) Ye

February 24, 2025

Revision History

Date	Version	Notes
24 February 2025	1.0	Initial draft

Contents

List of Tables

List of Figures

1 Symbols, Abbreviations, and Acronyms

symbol	description
CI	Continuous Integration
MG	Module Guide
MIS	Module Interface Specification
SRS	Software Requirement Specification
T	Test
VnV	Verification and Validation

This document provides an introductory blurb and roadmap of the Verification and Validation (VnV) plan

2 General Information

This section provides a brief description of the project background and introduces the objectives for the VnV plan.

2.1 Summary

MPIR is a sparse linear solver designed to solve large, sparse real matrices efficiently. It uses the General Minimal Residual (GMRES) method for internal matrix solves and iterative refinement techniques to improve both speed and accuracy. The software is intended for use in computational science, engineering, and numerical analysis applications. As a complete library suite, the software also includes example programs to demonstrate the solver interfaces and practical use cases of the solver.

2.2 Objectives

The primary objective of the Verification and Validation (VnV) plan is to ensure the correctness, accuracy, and efficiency of MPIR in solving sparse linear systems. The secondary objective is to verify usability and maintainability of the software for integration with other numerical libraries.

Usability testing for non-expert users is not prioritized is out of the scope of this VnV plan, as MPIR is only intended for domain-expert users. The solver is expected to use an external library for matrix factorization. The example programs will also depend on an external library for reading and writing sparse matrices in Matrix Market Exchange Format (*Matrix Market: File Formats* 2013).

2.3 Challenge Level and Extras

2.4 Relevant Documentation

See the Software Requirements Specification (*MPIR/docs/SRS.pdf at main · yex33/MPIR* 2025) for MPIR, which details the goals, requirements, assumptions, and theories of the software. The Module Guide (*MPIR/docs/Design/SoftArchitecture/MG.p*

at main · yex33/MPIR 2025) and Module Interface Specification (*MPIR/docs/Design/SoftDetailed at main · yex33/MPIR* 2025) document the design of MPIR.

3 Plan

This section details the plan for the verification of both the documents and the software for MPIR. The primary artifacts being verified are: SRS, software design, VnV Plan, and implementation.

3.1 Verification and Validation Team

The table below summarizes the VnV Team for the project:

Team Member	**Role**
Xunzhou Ye	Lead developer and tester
Qianlin Chen	"Domain expert", provides feedbacks on documents per course guidelines and document templates
Dr. Nedialkov	Primary stakeholder, oversees project direction and validates all documents

3.2 SRS Verification Plan

The SRS will be verified via feedbacks from the assigned domain expert and the project supervisor. A SRS checklist (Smith 2024b) will be used to guide the review process. After each major revision of the SRS, a meeting with the project supervisor shall be scheduled within two weeks of the revision date. Changes to the SRS shall be presented and discussed with the supervisor. All feedbacks from both the domain expert and the supervisor will be documented in the form of GitHub issues assigned to the project owner. The project owner is responsible to address and close these issues as they iterate and revise the SRS document. The key objective is to verify that the software requirements and the documentation are sound and coherent to the intended users and defined in the SRS.

3.3 Design Verification Plan

Since the software design is derived from a research study, the underlying algorithms and mathematical models are predefined. As a result, the scope for structural modifications is limited, and design decisions must align with the established research framework. The primary focus is on enhancing performance, ensuring software quality, and maintaining consistency with the research objectives.

The VnV team will conduct structured reviews of the design, leveraging their professional expertise to provide informed feedback. This evaluation will be guided by the MG and MIS checklists (Smith 2024a; Smith 2022a) to ensure that design principles are adhered to and best practices are followed. Similar to the SRS verification process, all feedback will be documented as GitHub issues to maintain transparency and traceability.

3.4 Verification and Validation Plan Verification Plan

The VnV checklist (Smith 2022b) will be used to review each iteration of the VnV Plan. The goal is to uncover any mistakes and reveal any coverage gaps through the supervision and review of the VnV team members. Once the project reaches a deliverable milestone, the VnV team will check whether the documented testing plans and verification processes have been accomplished and the requirements fulfilled. Feedbacks will again be documented as GitHub issues.

3.5 Implementation Verification Plan

Both automated and manual testing will be performed for this project. For automated static code analysis, linters will be integrated as part of the Continuous Integration (CI) pipeline via GitHub Actions. Details on the choice of tools and its use in the project will be discussed in Section 3.6 below. Plans and schemes for automated dynamic tests will be done at various levels of abstraction, including unit tests, system tests. Details are listed in Section 4 and 5. A detailed code walkthrough on core algorithms will be conducted with the project supervisor to ensure that the implementation align with the established research work that this project is based on.

3.6 Automated Testing and Verification Tools

The software is expected to be implemented in C++. The following are the language specific tools that will be used for automated testing and verifications:

- CMake (*CMake - Upgrade Your Software Build System* 2025) will be used to streamline the building and testing process of the software. CTest, which is part of CMake, will be used to generate code coverage reports for unit tests.
- doctest (*doctest/doctest* 2025) will be used as the unit testing framework for writing test cases.
- clang-format (*ClangFormat — Clang 21.0.0git documentation* 2025) will be used to format source codes based on a set of predefined rules specified in a configuration file. A Git Hook will be deployed to run a format check to ensure that all committed source codes are properly formatted. The goal of formatting codes is to improve code readability and consistency.
- clang-tidy (*Clang-Tidy — Extra Clang Tools 21.0.0git documentation* 2025) will be used as the linter for static code analysis. Committed codes should be free of linter errors and warnings to minimize the chance of having incorrect codes.
- Benchmark (*google/benchmark* 2025) will be used to evaluate the runtime performance of the solver as part of the non-functional verification process.

Apart from these language specific tools, the following general purposed tools are also used:

- GitHub Actions will be used as the CI pipeline for the project. Most of the language specific automated tools mentioned above will be integrated as part of CI to ensure that the verification process is reproducible in an isolated, remote environment. This verifies the verification process itself and further improves confidence of the verification process.

3.7 Software Validation Plan

A valid linear solver is one that correctly solves linear systems. In the context of this project, the validity of the software is primarily determined by its correctness—ensuring that the computed solutions satisfy the given equations within an acceptable tolerance. Since this software is part of a research study, its validity is also assessed by how well the implementation adheres to the established specifications and aligns with prior research work. To verify this, code walkthroughs of the core algorithms will be conducted with the project supervisor.

Beyond correctness, performance also plays a role in validation. The solver is designed to offer advantages over existing solutions, making performance benchmarking an essential validation step. To assess this, an automated performance benchmark will be conducted, generating empirical runtime data for different problem sizes. These results will be manually compared against established solvers to evaluate the solver's computational efficiency.

4 System Tests

This section outlines the tests that will be performed for MPIR to verify both the functional and non-functional requirements specified in the SRS (*MPIR/docs/SRS.pdf at main · yex33/MPIR* 2025). Input specifications and constraints are also listed in the SRS.

4.1 Tests for Functional Requirements

In this section, the system tests that will be conducted are described in detail. These tests will be used to verify the fulfillment of the functional requirements as listed in the SRS (*MPIR/docs/SRS.pdf at main · yex33/MPIR* 2025).

4.1.1 Matrix Inputs and Outputs

This section covers the requirement R3 of the SRS. This includes essentially a "driver" for the solver which loads sparse matrices from a text file in Matrix Market Exchange (.mtx) Format (*Matrix Market: File Formats* 2013) into memory, invokes the solver interfaces, and outputs the results returned

from the solver. The tests described below will verify that such "driver" is functional.

T1: matrix-io

Control: Automatic

Initial State: A hard coded matrix $\mathbf{A}$ of size 100×100 in the programming language of choice is instantiated.

Input: matrix $\mathbf{A}$ in plain text .mtx format of size 100×100, a random vector $\mathbf{b}$ of size 100. $u_f = u_w = u_r =$ `double`

Output: The elements of $\mathbf{A}$ matches the hard-coded one. $\mathbf{x}$ of size 100

Test Case Derivation: N/A

How test will be performed: Automatic

4.1.2 Correctness Tests with Manufactured Solutions

This section covers one of the ways to verify the requirements R1 and R2 of the SRS. This includes tests on the accuracy of the yielded solution from the solver by manufacturing an exact solution $\mathbf{x}_{\text{ref}}$ to the problem $\mathbf{Ax} = \mathbf{b}$. This manufacturing process loosely follows the scheme below:

1. $\mathbf{x}_{\text{ref}} \leftarrow$ some random vector
2. $\mathbf{b} \leftarrow \mathbf{A}\mathbf{x}_{\text{ref}}$
3. Solve $\mathbf{Ax} = \mathbf{b}$
4. $e \leftarrow \dfrac{\|\mathbf{x} - \mathbf{x}_{\text{ref}}\|_2}{\|\mathbf{x}_{\text{ref}}\|_2}$

The relative error e will be used as the accuracy metric.

T2: generated-double-double

Control: Automatic

Initial State: matrix $\mathbf{A}$ is read from file and stored in memory. An expected exact solution $\mathbf{x}_{\text{ref}}$ is prepared.

Input: matrix $\mathbf{A}$ of size $10\,000 \times 10\,000$ with $\text{cond}(\mathbf{A}) \approx 1 \times 10^2$, $\mathbf{b}$ of size $10\,000$, $u_f = u_w = u_r =$ `double`

Output: $\mathbf{x}$ of size 10 000 such that $e = \frac{\|\mathbf{x} - \mathbf{x}_{\text{ref}}\|_2}{\|\mathbf{x}_{\text{ref}}\|_2} < 1 \times 10^{-12}$

Test Case Derivation: $\mathbf{x}_{\text{ref}}$ is randomly generated. $\mathbf{b} = \mathbf{A}\mathbf{x}_{\text{ref}}$

How test will be performed: Automatic

T3: generated-single-double

Control: Automatic

Initial State: matrix $\mathbf{A}$ is read from file and stored in memory. An expected exact solution $\mathbf{x}_{\text{ref}}$ is prepared.

Input: matrix $\mathbf{A}$ of size $10\,000 \times 10\,000$ with $\text{cond}(\mathbf{A}) \approx 1 \times 10^2$, $\mathbf{b}$ of size 10 000, $u_f = \texttt{single}, u_w = u_r = \texttt{double}$

Output: $\mathbf{x}$ of size 10 000 such that $e = \frac{\|\mathbf{x} - \mathbf{x}_{\text{ref}}\|_2}{\|\mathbf{x}_{\text{ref}}\|_2} < 1 \times 10^{-12}$

Test Case Derivation: $\mathbf{x}_{\text{ref}}$ is randomly generated. $\mathbf{b} = \mathbf{A}\mathbf{x}_{\text{ref}}$. Matrix factorization is done in single precision while working and residual computing are done in double precision.

How test will be performed: Automatic

4.1.3 Correctness Tests against Trusted Solvers

This section covers the other way to verify the requirements R1 and R2 of the SRS. This includes tests on the accuracy of the yielded solution from the solver by comparing it to an external, trusted solver to the problem $\mathbf{A}\mathbf{x} = \mathbf{b}$. This process loosely follows the scheme below:

1. $\mathbf{x}_{\text{ref}} \leftarrow$ solution by an external solver
2. Solve $\mathbf{A}\mathbf{x} = \mathbf{b}$
3. $e \leftarrow \frac{\|\mathbf{x} - \mathbf{x}_{\text{ref}}\|_2}{\|\mathbf{x}_{\text{ref}}\|_2}$

The relative error e will be used as the accuracy metric.

T4: external-double-double

Control: Automatic

Initial State: matrix $\mathbf{A}$ is read from file and stored in memory. The same problem $\mathbf{Ax} = \mathbf{b}$ is passed to the external solver. A reference solution $\mathbf{x}_{\text{ref}}$ is prepared.

Input: matrix $\mathbf{A}$ of size $10\,000 \times 10\,000$ with $\text{cond}(\mathbf{A}) \approx 1 \times 10^2$, $\mathbf{b}$ of size $10\,000$, $u_f = u_w = u_r =$ `double`

Output: $\mathbf{x}$ of size $10\,000$ such that $e = \dfrac{\|\mathbf{x} - \mathbf{x}_{\text{ref}}\|_2}{\|\mathbf{x}_{\text{ref}}\|_2} < 1 \times 10^{-12}$

Test Case Derivation: $\mathbf{x}_{\text{ref}}$ is randomly generated. $\mathbf{b} = \mathbf{A}\mathbf{x}_{\text{ref}}$

How test will be performed: Automatic

T5: external-single-double

Control: Automatic

Initial State: matrix $\mathbf{A}$ is read from file and stored in memory. The same problem $\mathbf{Ax} = \mathbf{b}$ is passed to the external solver. A reference solution $\mathbf{x}_{\text{ref}}$ is prepared.

Input: matrix $\mathbf{A}$ of size $10\,000 \times 10\,000$ with $\text{cond}(\mathbf{A}) \approx 1 \times 10^2$, $\mathbf{b}$ of size $10\,000$, $u_f =$ `single`, $u_w = u_r =$ `double`

Output: $\mathbf{x}$ of size $10\,000$ such that $e = \dfrac{\|\mathbf{x} - \mathbf{x}_{\text{ref}}\|_2}{\|\mathbf{x}_{\text{ref}}\|_2} < 1 \times 10^{-12}$

Test Case Derivation: $\mathbf{x}_{\text{ref}}$ is randomly generated. $\mathbf{b} = \mathbf{A}\mathbf{x}_{\text{ref}}$. Matrix factorization is done in single precision while working and residual computing are done in double precision.

How test will be performed: Automatic

4.2 Tests for Nonfunctional Requirements

4.2.1 Area of Testing1

Title for Test

1. test-id1

Type: Functional, Dynamic, Manual, Static etc.

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:

4.2.2 Area of Testing2

...

4.3 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.

References

Clang-Tidy — Extra Clang Tools 21.0.0git documentation (2025). URL: https://clang.llvm.org/extra/clang-tidy/ (visited on 02/25/2025).

ClangFormat — Clang 21.0.0git documentation (2025). URL: https://clang.llvm.org/docs/ClangFormat.html (visited on 02/25/2025).

CMake - Upgrade Your Software Build System (2025). URL: https://cmake.org/ (visited on 02/25/2025).

doctest/doctest (Feb. 25, 2025). original-date: 2014-08-05T21:43:51Z. URL: https://github.com/doctest/doctest (visited on 02/25/2025).

google/benchmark (Feb. 14, 2025). original-date: 2013-12-12T00:10:48Z. URL: https://github.com/google/benchmark (visited on 02/14/2025).

Matrix Market: File Formats (Aug. 14, 2013). The Matrix Market. URL: https://math.nist.gov/MatrixMarket/formats.html (visited on 02/14/2025).

MPIR/docs/Design/SoftArchitecture/MG.pdf at main · yex33/MPIR (2025). GitHub. URL: https://github.com/yex33/MPIR/blob/main/docs/Design/SoftArchitecture/MG.pdf.

MPIR/docs/Design/SoftDetailedDes/MIS.pdf at main · yex33/MPIR (2025). GitHub. URL: https://github.com/yex33/MPIR/blob/main/docs/Design/SoftDetailedDes/MIS.pdf.

MPIR/docs/SRS.pdf at main · yex33/MPIR (2025). GitHub. URL: https://github.com/yex33/MPIR/blob/main/docs/SRS/SRS.pdf.

Smith, Spencer (Sept. 1, 2022a). *MPIR/docs/Checklists/MIS-Checklist.pdf at main · yex33/MPIR*. GitHub. URL: https://github.com/yex33/MPIR/blob/main/docs/Checklists/MIS-Checklist.pdf.

— (Sept. 1, 2022b). *MPIR/docs/Checklists/VnV-Checklist.pdf at main · yex33/MPIR*. GitHub. URL: https://github.com/yex33/MPIR/blob/main/docs/Checklists/VnV-Checklist.pdf.

— (Apr. 2, 2024a). *MPIR/docs/Checklists/MG-Checklist.pdf at main · yex33/MPIR*. GitHub. URL: https://github.com/yex33/MPIR/blob/main/docs/Checklists/MG-Checklist.pdf.

— (Feb. 26, 2024b). *MPIR/docs/Checklists/SRS-Checklist.pdf at main · yex33/MPIR*. GitHub. URL: https://github.com/yex33/MPIR/blob/main/docs/Checklists/SRS-Checklist.pdf.

6 Appendix

This is where you can place additional information.

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.