System Verification and Validation Plan for MPIR

Xunzhou (Joe) Ye

April 7, 2025

Revision History

Date	Version	Notes
24 February 2025	1.0	Initial draft

Contents

1 Symbols, Abbreviations, and Acronyms iv

2 General Information 1
- 2.1 Summary . . . 1
- 2.2 Objectives . . . 1
- 2.3 Challenge Level and Extras . . . 1
- 2.4 Relevant Documentation . . . 1

3 Plan 2
- 3.1 Verification and Validation Team . . . 2
- 3.2 SRS Verification Plan . . . 2
  - 3.2.1 Supervisor Walkthrough Meetings . . . 2
  - 3.2.2 Feedback Management . . . 3
- 3.3 Design Verification Plan . . . 3
- 3.4 Verification and Validation Plan Verification Plan . . . 4
- 3.5 Implementation Verification Plan . . . 4
- 3.6 Automated Testing and Verification Tools . . . 4
- 3.7 Software Validation Plan . . . 5

4 System Tests 6
- 4.1 Tests for Functional Requirements . . . 6
  - 4.1.1 Matrix Inputs and Outputs . . . 6
  - 4.1.2 Correctness Tests with Manufactured Solutions . . . 7
  - 4.1.3 Correctness Tests against Trusted Solvers . . . 8
- 4.2 Tests for Nonfunctional Requirements . . . 9
  - 4.2.1 Accuracy . . . 9
  - 4.2.2 Usability . . . 10
  - 4.2.3 Maintainability . . . 10
  - 4.2.4 Portability . . . 11
- 4.3 Traceability Between Test Cases and Requirements . . . 12

5 Unit Test Description 12

6 Appendix 14
- 6.1 Symbolic Parameters . . . 14
- 6.2 Usability Survey Questions . . . 14

List of Tables

1 Traceability matrix showing the connections between test cases and requirements 12

List of Figures

1 Symbols, Abbreviations, and Acronyms

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

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

The challenge level remains the same at the general level. The extra is expected to be conducting a usability test which will be discussed in Section 4.2.2.

2.4 Relevant Documentation

See the Software Requirements Specification (*SRS · yex33/MPIR* 2025) for MPIR, which details the goals, requirements, assumptions, and theories of the software. The Module Guide (*MG · yex33/MPIR* 2025) and Module Interface Specification (*MIS · 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. Smith	Course instructor, provides feedbacks on documents per course requirements
Dr. Nedialkov	Primary stakeholder, oversees project direction and validates all documents

3.2 SRS Verification Plan

The SRS will be verified through an iterative review process with the project supervisor. The objective is to ensure that the documented software requirements accurately reflect the project's goals, are technically sound, and meet quality standards for scientific software development.

3.2.1 Supervisor Walkthrough Meetings

After each major revision of the SRS, a walkthrough meeting will be scheduled within two weeks. The meeting will be led by the author(s) of the SRS and structured to maximize clarity, engagement, and feedback. To avoid the inefficiencies of a full line-by-line review, the walkthrough will instead:

- Begin with a brief overview of the SRS document structure and purpose, reaffirming the role of each section in capturing software requirements.
- Focus the discussion on key functional and non-functional requirements, underlying assumptions, and any newly introduced models or scope changes.
- Use figures, equations, and tables as prompts for discussion and clarification, ensuring their interpretation aligns with the supervisor's understanding.
- Be guided by open-ended questions posed by the author (e.g., "What are your thoughts on the maintainability test design?" or "How well does the portability requirement reflect our deployment targets?"), rather than binary yes/no queries.
- Encourage clarifications, suggestions, and identification of gaps or inconsistencies in the document.

3.2.2 Feedback Management

Feedback from the walkthrough will be documented as GitHub issues assigned to the project owner. Each issue will include a clear summary of the feedback point, context, and the proposed or required action. The project owner is responsible for addressing and closing these issues through successive iterations of 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 2024; 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 nonfunctional 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 nonfunctional requirements specified in the SRS (*SRS · 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 (*SRS · 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. $u_f = u_w = u_r = \texttt{double}$

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 $10\,000$, $u_f = u_w = u_r = \texttt{double}$

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

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 $e = \dfrac{\|\mathbf{x} - \mathbf{x}_{\text{ref}}\|_2}{\|\mathbf{x}_{\text{ref}}\|_2} < 1 \times 10^{-12}$

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

4.1.3 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. $e \leftarrow \dfrac{\|\mathbf{x} - \mathbf{x}_{\text{ref}}\|_2}{\|\mathbf{x}_{\text{ref}}\|_2}$

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 $\mathbf{A}\mathbf{x} = \mathbf{b}$ 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 $10\,000$, $u_f = u_w = u_r = \texttt{double}$

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 $\mathbf{A}\mathbf{x} = \mathbf{b}$ 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 $10\,000$, $u_f = \texttt{single}, u_w = u_r = \texttt{double}$

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

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 nonfunctional requirements as listed in the SRS (*SRS · yex33/MPIR* 2025).

4.2.1 Accuracy

The accuracy of the solver will be assessed by verifying that it converges to a solution within the user-defined tolerance ϵ. The level of accuracy required for computational science and engineering applications will be evaluated through the relative residual norm after convergence. The following tests will be performed to verify the nonfunctional requirement NFR1 in the SRS:

T6: nfr-acc

Type: Functional, Automated

Initial State: Similar to T2, matrix $\mathbf{A}$ is read from file and stored in memory. An manufactured expected exact solution $\mathbf{x}_{\text{ref}}$ is prepared.

Input: Similar to T2, with an extra input for user-specified tolerance $\epsilon = 1 \times 10^{-12}$.

Output/Result: $\mathbf{x}$ of size $10\,000$ such that $e = \dfrac{\|\mathbf{x} - \mathbf{x}_{\text{ref}}\|_2}{\|\mathbf{x}_{\text{ref}}\|_2} < \epsilon = 1 \times 10^{-12}$. The computed solution's residual norm should be below the threshold ϵ. If convergence is not achieved after iterations, a warning should be issued.

How test will be performed: Automatic

4.2.2 Usability

The usability of the solver will be evaluated based on the clarity and accessibility of its public Application Programming Interface (API). The API should be self-contained, readable, and easy to integrate into other software as a dependency. Usability testing will reference the user characteristics section and include developer feedback. The following tests will be performed to verify the nonfunctional requirement NFR2 in the SRS:

T7: nfr-use

Type: Static, Review, Survey-Based

Initial State: The API documentation and usage examples are available. The API documentation of an external, established sparse linear solver is also presented as a comparable reference.

Input: VnV team members review the API and compare it against that of the external solver.

Output/Result: Feedback on clarity, ease of integration, pros and cons, and suggestions for improvements.

How test will be performed: A usability survey will be conducted (see Appendix 6.2). API walkthroughs and documentation inspections will be used to ensure that function calls and configurations are intuitive.

4.2.3 Maintainability

The effort required to modify the solver should be kept minimal. This will be verified by estimating the complexity of implementing likely changes and ensuring that modifications take less than a fraction of the original development time. The following tests will be performed to verify the nonfunctional requirement NFR3 in the SRS:

T8: nfr-mt

Type: Static, Code Review

Initial State: The codebase is available and structured.

Input: Introduce a minor algorithmic modification or API extension.

Output/Result: Measure the number of files and lines of code modified, as well as time taken.

How test will be performed: A code walkthrough will be conducted with team members to assess maintainability using the MG and MIS guidelines.

T9: nfr-format

Type: Static, Automated

Initial State: The source code is available in a repository with predefined formatting and linting rules.

Input: Run automated checks using clang-format for code style enforcement and clang-tidy for static analysis.

Output/Result: The code should conform to the predefined formatting rules, and no critical linting warnings or errors should remain unaddressed.

How test will be performed: As part of the CI pipeline, clang-format and clang-tidy will be executed on every commit. Any violations must be resolved before merging changes.

4.2.4 Portability

The solver should run on all actively maintained operating systems, including Windows 10, Windows 11, Linux, and MacOS. Compatibility testing will verify that all required functionalities work across different platforms. The following tests will be performed to verify the nonfunctional requirement NFR4 in the SRS:

T10: nfr-port

Type: Functional, Automated

Initial State: The solver is compiled and deployed on different machines with different operating systems.

Input: Run both the example programs and all automated functional tests described in Section 4.1.

Output/Result: The solver successfully compiles, executes test cases, and produces correct results.

How test will be performed: Automated CI pipelines will test the software across different platforms. Success will be determined by running the test suite in each environment and confirming consistent results.

4.3 Traceability Between Test Cases and Requirements

	R1	R2	R3	NFR1	NFR2	NFR3	NFR4
T1			X				
T2	X	X					
T3	X	X					
T4	X	X					
T5	X	X					
T6				X			
T7					X		
T8						X	
T9						X	
T10							X

Table 1: Traceability matrix showing the connections 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).

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

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

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

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

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

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

6.2 Usability Survey Questions

1. **Did you find the function names and parameters intuitive?**
   - ☐ Very intuitive
   - ☐ Somewhat intuitive
   - ☐ Neutral
   - ☐ Confusing
   - ☐ Very confusing
2. **Did the provided documentation clearly explain how to use the API?**
   - ☐ Yes, everything was clear
   - ☐ Mostly clear, but some aspects were confusing
   - ☐ Neutral
   - ☐ No, the documentation was unclear
   - ☐ No documentation was provided
3. **Were the example use cases sufficient to understand how to integrate the solver?**
   - ☐ Yes, they covered all necessary cases
   - ☐ Somewhat, but additional examples would be helpful
   - ☐ No, they were insufficient
4. **How easy was it to set up and call the solver in a basic use case?**
   - ☐ Very easy (Just a few function calls)
   - ☐ Somewhat easy (Needed minor adjustments)

- ☐ Neutral
- ☐ Difficult (Required significant effort)
- ☐ Very difficult (I could not get it working)

5. **Did the API provide useful error messages when incorrect inputs were provided?**
   - ☐ Yes, the error messages were informative
   - ☐ Somewhat, but could be improved
   - ☐ No, the error messages were vague or missing

6. **If you encountered any difficulties, what were they? (Open-ended)**

   ---

7. **How would you rate the overall usability of the API?**
   - ☐ Excellent
   - ☐ Good
   - ☐ Neutral
   - ☐ Poor
   - ☐ Very Poor

8. **What improvements would you suggest to enhance the usability of the API? (Open-ended)**

   ---

9. **Would you recommend this solver for integration into a larger numerical computing project?**
   - ☐ Yes
   - ☐ No
   - ☐ Maybe, if improvements are made