Verification and Validation Report: MPIR

Xunzhou (Joe) Ye

April 16, 2025

1 Revision History

Date	Version	Notes
16 April 2025	1.0	Initial draft

2 Symbols, Abbreviations and Acronyms

symbol	description
T	Test

Contents

List of Tables

List of Figures

3 Functional Requirements Evaluation

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 (Ye 2025).

3.1 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 a "driver" is functional.

T1 matrix-io

Output: The elements of $\mathbf{A}$ matches exactly the one in the .mtx file. Result solution $\mathbf{x}$ is of size 100.

Result: Pass

3.2 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 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. The values of the manufactured reference solution $\mathbf{x}_{\text{ref}}$ in this section is uniformly distributed in the range of $[\min(a_{i,j}), \max(a_{i,j})]$. For the test cases in Sections 3.2, 3.3,

and below, the `bundle1` matrix (M. Lourakis 2006) from the Florida Sparse Matrix Collection (Davis and Hu 2011) will be used as the input matrix $\mathbf{A}$. This matrix has a size of $10\,581 \times 10\,581$ and $770\,811$ non-zeros. The estimated condition number is 1.3×10^4.

T2: generated-double-double

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

Result: Pass

T3: generated-single-double

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

Result: Pass

3.3 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 $\mathbf{Ax} = \mathbf{b}$. This process loosely follows the scheme below:

1. $\mathbf{x}_{\text{ref}} \leftarrow$ solution by an external solver
2. Solve $\mathbf{Ax} = \mathbf{b}$
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. For the test cases in this Section, MATLAB® will be used as the external reference solver.

T4: external-double-double

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

Result: Pass

T5: external-single-double

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

Result: Pass

4 Nonfunctional Requirements Evaluation

4.1 Accuracy

The accuracy of the solver is 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 functional tests T2, T3, T4, T5 are sufficient to verify the nonfunctional requirement NFR1 in the SRS with an accuracy metric of $\epsilon \approx 1 \times 10^{-10}$. Considering that the residual precision $u_r =$ `double`, the chosen $\epsilon \approx 1 \times 10^{-10}$ is reasonably close to the machine epsilon in `double` precise $\epsilon_{\text{mach}} \approx 1.1 \times 10^{-16}$.

4.2 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:

T6: nfr-use

Output/Result: Pass

The usability survey feedback was largely positive, highlighting the API's clarity and ease of integration, with a few suggestions for expanding documentation and use-case coverage. See Appendix A for a detailed report.

4.3 Maintainability

The maintainability test T7 was not executed due to the time constraints of the project and the nature of the proposed changes. Implementing support for an additional matrix storage format (CSR) is possible but still requires efforts to implement. While such a change would offer a meaningful scenario to assess maintainability, it is outside the defined scope of the current project,

which focuses on mixed-precision solver strategies for matrices in CSC format. The maintainability test is acknowledged but deferred as future work or as a candidate for post-project evaluation.

4.4 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:

T8: nfr-port

Output/Result: Pass, See CTest Github Action.

4.5 Performance

To fulfill the nonfunctional requirement for performance NFR5, a manual benchmark test will be conducted to evaluate the solver's runtime efficiency when using mixed-precision arithmetic. The runtime of the functional tests T2 and T3 will serve as the basis for comparing performance between two configurations:

1. Mixed-precision mode: factorization in single precision, internal solves and residual evaluations in double precision.

2. Double-precision mode: all computations in double precision.

Each test will be executed multiple times to account for variability and ensure statistically stable results. This is necessary because clock-time measurements on modern processors can be affected by factors such as CPU caching, thermal throttling, and OS-level scheduling.

T9: nfr-perf

Output/Result: The mixed-precision configuration consistently outperformed the full double-precision mode, achieving a 24.25% reduction in runtime on a representative sparse matrix. The results meet the nonfunctional performance requirement outlined in NFR5, confirming the efficiency advantage of the mixed-precision implementation. Test data is attached in Appendix B.

5 Comparison to Existing Implementation

A detailed comparison of the current implementation of the preconditioned GMRES algorithm against the previous prototype is provided in Appendix C. The improvements made to this core algorithm reflect a broader design philosophy adopted throughout the development of MPIR:

- A cleaner and safer C++ interface, with significantly fewer template parameters.
- Use of standard containers (`std::vector`) to eliminate manual memory management and reduce the risk of resource leaks.
- Decomposition of complex numerical routines into self-contained, reusable primitives, enhancing both readability and testability.
- A consistent type-safety mechanism using the `Refinable` concept, replacing scattered manual casts.
- Direct return of computed results, enabling more idiomatic error handling and seamless integration.

6 Unit Testing

All tests passing. See CTest Github Action.

7 Changes Due to Testing

Due to project time constraints and the scope of the current phase, no further implementation changes were made following this round of testing.

8 Automated Testing

Refer to Section 6.

9 Trace to Requirements

	R1	R2	R3	NFR1	NFR2	NFR3	NFR4	NFR5
T1			X					
T2	X	X		X				
T3	X	X		X				
T4	X	X		X				
T5	X	X		X				
T6					X			
T7						X		
T8							X	
T9								X

Table 1: Traceability matrix showing the connections between test cases and requirements

10 Trace to Modules

No specific test case is given in Section 6, but the unit testing strategy has been covered for each tested module.

11 Code Coverage Metrics

Code coverage testing was not performed in this project due to the following considerations:

- Given the limited development timeline, priority was given to functional correctness, numerical accuracy, and performance testing—areas more critical for a numerical solver in a research context.
- The primary objective of the project was to validate the correctness and performance of the underlying numerical methods (e.g., mixed-precision GMRES and iterative refinement), rather than achieving high test coverage metrics.

- Since the solver is structured around reusable, well-scoped numerical routines that are individually tested with known inputs and analytical results, the marginal benefit of computing line or branch coverage was minimal compared to the effort required to instrument and interpret those metrics.

References

Davis, Timothy A. and Yifan Hu (Dec. 7, 2011). "The university of Florida sparse matrix collection". In: *ACM Trans. Math. Softw.* 38.1, 1:1–1:25. ISSN: 0098-3500. DOI: 10.1145/2049662.2049663. URL: https://doi.org/10.1145/2049662.2049663 (visited on 04/11/2025).

M. Lourakis (2006). *Lourakis/bundle1*. SuiteSparse Matrix Collection. URL: https://sparse.tamu.edu/Lourakis/bundle1 (visited on 04/11/2025).

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

Ye, Xunzhou (2025). *SRS · yex33/MPIR*. GitHub. URL: https://github.com/yex33/MPIR/blob/main/docs/SRS/SRS.pdf.

Appendices

A Usability Survey Results

A.1 Summary

A usability evaluation was conducted on the solver's public API by three VnV team members with experience in numerical software development. Participants reviewed the API documentation, examined usage examples, and completed a structured survey comparing the solver's API to Eigen's sparse solver interface.

The feedback was largely positive, highlighting the API's clarity and ease of integration, with a few suggestions for expanding documentation and use-case coverage.

A.2 Participant Summary

Reviewer	Background
Reviewer A	MSc student, numerical linear algebra
Reviewer B	Senior undergrad, software engineering
Reviewer C	TA, scientific computing

A.3 Likert Scale Results

Question	Very Good	Good	Neutral	Poor
Q1. Intuitive function names	2	1	0	0
Q2. Clear documentation	2	1	0	0
Q3. Sufficient examples	1	2	0	0
Q4. Ease of setup	1	2	0	0
Q5. Quality of error messages	1	2	0	0
Q7. Overall usability	2	1	0	0
Q9. Recommend API	2	1	0	0

A.4 Open-Ended Feedback Highlights

- *Q6. Encountered difficulties:*
  - "It wasn't immediately clear how to switch between CSC and CSR modes."
  - "I expected a higher-level wrapper function that hides the solver config setup."
- *Q8. Suggestions for improvement:*
  - "Add more detailed documentation for return types and default configurations."
  - "Include an end-to-end integration example with a real-world sparse matrix."

A.5 Summary Graphs

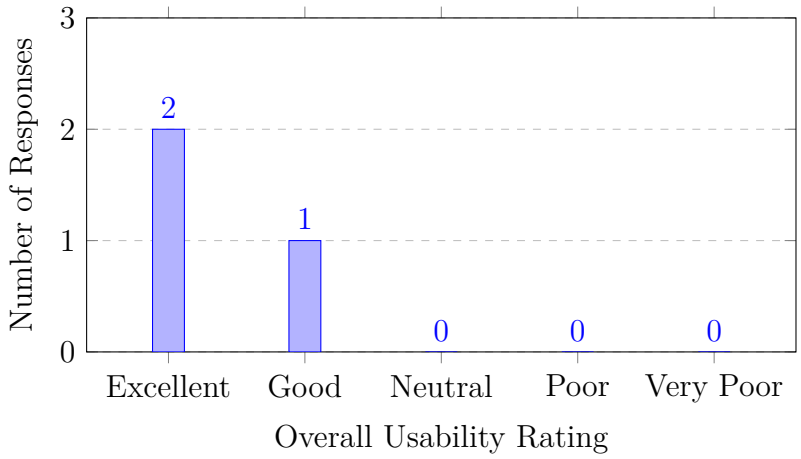

Figure 1: Overall Usability Rating

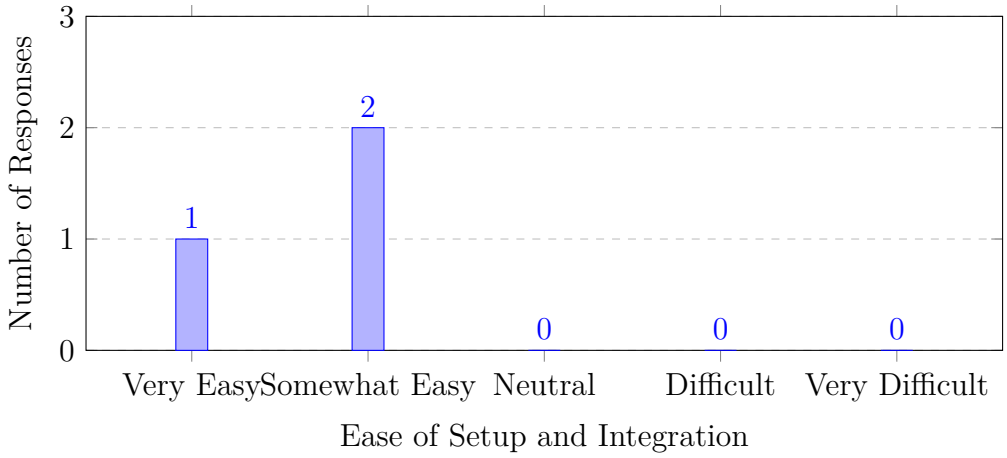

Figure 2: Ease of Setup and Integration

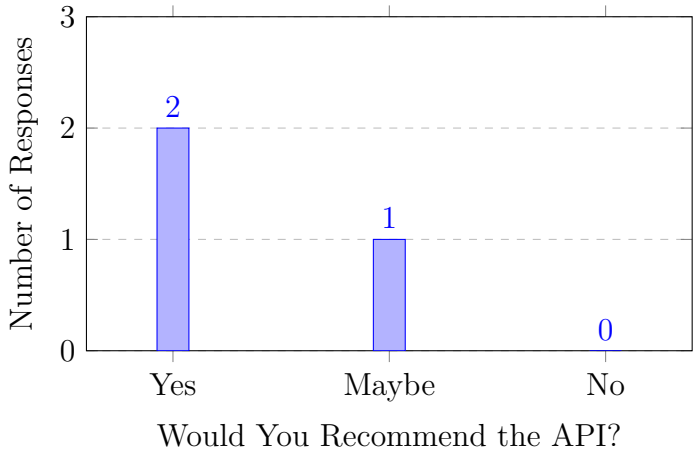

Figure 3: Recommendation Likelihood

A.6 Conclusion

The solver's API was rated as clear and usable, with minor improvements suggested for documentation and configurability. All reviewers would recommend it for use in larger numerical projects, provided additional examples and higher-level wrappers are included.

B Performance Test Results

B.1 Raw Runtime Data Table

Table 2: Runtime measurements for mixed-precision vs. double-precision (10 runs)

Run	Mixed Precision (ms)	Double Precision (ms)
1	11.8	15.4
2	11.7	15.5
3	11.6	15.4
4	11.9	15.3
5	11.8	15.6
6	11.7	15.5
7	11.6	15.4
8	11.7	15.6
9	11.8	15.5
10	11.6	15.5

B.2 Summary Statistics Table

Table 3: Summary of performance results

Metric	Mixed Precision	Double Precision
Average Runtime (ms)	11.72	15.47
Standard Deviation (ms)	0.11	0.09
Measured Gain	**24.25%**	–

C `PrecondGmres` Implementation Comparison

This section compares the new `PrecondGmres` implementation with the prior research prototype `gmres_cond` and `gmres_cond2`. The goal is to highlight improvements in modularity, safety, readability, and type handling. The comparison focuses on six major dimensions: interface design, memory management, core algorithm logic, type safety, maintainability, and return behavior.

Interface Design

Aspect	**Old Implementation**	**New Implementation**
Templating	Six template parameters (`ATVPREC`, `T`, `FACTPREC`, `RESPREC`, `WORKPREC`, `XPREC`)	Three template parameters constrained via `Refinable<UF, UW, UR>`
Parameter Style	Long list of raw pointers interleaved with outputs	Clean API with STL containers and logical parameter order
Preconditioner Input	Passed as separate pointer arrays	Encapsulated as class members
Callback for `A*v`	`std::function` passed per call	Replaced with internal `MatrixMultiply` + `QDLDL_solve` calls

Table 4: Interface Design Comparison

Memory Management

Aspect	Old Implementation	New Implementation
Memory Handling	Raw `new[]`/`delete[]` for all arrays	Automatic `std::vector` usage
Resource Safety	Manual cleanup at multiple return paths	RAII via containers ensures exception safety
Temporary Buffers	All manually allocated and deallocated	Constructed on-demand and destroyed automatically

Table 5: Memory and Resource Management

Algorithm Structure and Numerical Logic

Aspect	Old Implementation	New Implementation
Arnoldi Loop	Inline pointer math with duplicated logic	Modular functions (`VectorDot`, `VectorScale`, etc.)
Reorthogonalization	Implemented via nested loops manually	Same logic using modular vector operations
Givens Rotations	Explicit array indexing and logic inside loop	Decomposed into reusable transformations
Residual Computation	Custom residual + norm macros	Uses templated `Dnrm2` and subtraction helpers

Table 6: Numerical Logic and Algorithm Flow

Precision and Type Handling

Aspect	Old Implementation	New Implementation
Precision Control	Manual cast between six different types	Centralized via `Refinable<UF, UW, UR>`
Casting Overhead	Frequent `static_cast` at use sites	Type conversions localized and explicit
Precision Consistency	Developer responsible for type coherence	Enforced via compile-time constraints

Table 7: Precision and Type Safety

Maintainability and Readability

Aspect	Old Implementation	New Implementation
Function Length	500+ lines in one function	Structured into 50-line modular components
Naming	Many temporary or cryptic variable names	Clear variable names and helper functions
Readability	Difficult to trace control flow	Code reads like algorithm pseudocode
Extensibility	Difficult to adapt to new formats or solvers	Designed for modular extension

Table 8: Maintainability and Readability

Return and Error Handling

Aspect	Old Implementation	New Implementation
Return Method	Outputs written to raw pointer arguments	Returns computed vector directly
Error Handling	No explicit error path	Returns empty vector on failure (e.g., size mismatch)
Status Reporting	Total iterations via output reference	Embedded in solver's internal state (TBD for next version)

Table 9: Return Value and Error Handling

Summary

The new `PrecondGmres` implementation significantly improves over the previous research prototype in terms of code clarity, modularity, type safety, and maintainability. By adopting standard C++ constructs and a principled separation of responsibilities, the solver becomes easier to reason about, extend, and integrate. The use of constraints through the `Refinable` concept further ensures compile-time precision correctness, reducing the risk of errors in mixed-precision configurations.