

# CAS 741 (Development of Scientific Computing Software)

Winter 2025

## MPIR Verification and Validation Plan

Xunzhou (Joe) Ye

Faculty of Engineering, McMaster University

February 14, 2025



# General Information

- 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.
- 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.

# Inputs

Variable	Description
<b>A</b>	$n \times n$ matrix
<b>b</b>	$n$ -vector
$u_f$	factorization precision
$u_w$	working precision
$u_r$	precision in which the residuals are computed

# VnV Objectives

**Primary** Ensure the correctness, accuracy, and efficiency of MPIR in solving sparse linear systems.

**Secondary** Verify usability and maintainability of the software for integration with other numerical libraries.

**Out of Scope** Usability testing for non-expert users is not prioritized. External libraries used for matrix factorization, reading and writing sparse matrices in Matrix Market Exchange Format (*Matrix Market: File Formats 2013*), unit testing, and performance benchmarking are assumed to be correct.

# VnV Team

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

# Functional System Testing: Correctness and Accuracy

- Define acceptance criteria in terms of machine epsilon, the smallest difference in value that the computer can tell apart at a given precision.
- The condition number  $\text{cond}(\mathbf{A})$  is an important factor when choosing inputs.

Floating-Point Format	Machine Epsilon ( $\epsilon_{\text{mach}}$ )
bfloat16	$3.91 \times 10^{-3}$
fp16 (IEEE 754 Half)	$4.88 \times 10^{-4}$
fp32 (IEEE 754 Single)	$1.19 \times 10^{-7}$
fp64 (IEEE 754 Double)	$2.22 \times 10^{-16}$
fp128 (IEEE 754 Quad)	$1.93 \times 10^{-34}$

# The Known Solution Vector Approach

Test against manufactured solutions:

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

Caveats:

- Large error does not necessarily mean the solver is incorrect. If  $\text{cond}(\mathbf{A})$  is large, the error is likely to be large as well.
- For well-conditioned matrices, e.g.  $\text{cond}(\mathbf{A}) < 10^4$ , the relative error is expected to be  $e \approx 10^{-12}$  in double precision ( $\epsilon_{\text{mach}} \approx 2.2 \times 10^{-16}$ )

# The Trusted Solver Comparison Approach

- MATLAB® vpa
- Other readily available sparse solvers

The relative error,  $e = \frac{\|\mathbf{x} - \mathbf{x}_{\text{ref}}\|_2}{\|\mathbf{x}_{\text{ref}}\|_2}$  should be within a reasonable multiple of  $\epsilon_{\text{mach}}$  across a set of test problems.



# Functional System Testing: the Plan

1. Start with numerically stable inputs (well conditioned  $\mathbf{A}$ ) without mixed-precisions. Verify accuracy with both the “known solution vector” and “trusted solver comparison” approach.
2. Hypothesize that the solver is able to solve systems with condition number up to  $COND\_MAX$ . Test robustness by inputting ill-conditioned matrices with  $\text{cond}(\mathbf{A})$  at different magnitudes. (Non-functional?)
3. Hypothesize that the solver is able to maintain accuracy when mixed-precision technique is applied. Test accuracy by inputting different combinations of  $u_f, u_w, u_r$ .

# Functional System Testing: an Example

test-id1 Control: Automatic

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

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

Output: **x** of size  $10\,000$  such that

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

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

$$\mathbf{b} = \mathbf{A}\mathbf{x}_{\text{ref}}$$

How test will be performed: Automatic

# Non-Functional System Testing: Performance

Choice of performance metric?

- Convergence rate?
- Computation time/Wall-clock time

# Potential Problems with Measuring Runtime

- CPU clock frequency, temperature; Caching
- OS scheduling, interrupts, context switching
- Hardware optimizations specifically for certain arithmetic/precision; CPU vs. GPU
- Inconsistency between runs, hard to reproduce results

# Established Tools for Benchmarking

*google/benchmark 2025*

- Repeats benchmark cases until a statistically stable result is obtained
- Automatic CPU frequency scaling adjustments

# Non-Functional System Testing: the Plan

1. Hypothesize that the use of mixed-precision technique brings consistent improvement in runtime. Given well-conditioned inputs, benchmark runtime by varying  $u_f, u_w, u_r$ .

# References



*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).

# Questions