Module Interface Specification for MPIR

Xunzhou (Joe) Ye

April 16, 2025

1 Revision History

Date	Version	Notes
19 March 2025	1.0	Initial draft
16 April 2025	1.1	Refine according to feedbacks

2 Symbols, Abbreviations and Acronyms

See SRS Documentation at Ye 2025b

Contents

3 Introduction

The following document details the Module Interface Specifications for MPIR. It is intended to solve a sparse linear system with an iterative method in mixed-precisions.

Complementary documents include the System Requirement Specifications and Module Guide. The full documentation and implementation can be found at Ye 2025b and Ye 2025a.

4 Notation

The structure of the MIS for modules comes from Hoffman and Strooper 1995, with the addition that template modules have been adapted from Ghezzi, Jazayeri, and Mandrioli 2003. The mathematical notation comes from Chapter 3 of Hoffman and Strooper 1995. For instance, the symbol := is used for a multiple assignment statement and conditional rules follow the form $(c_1 \Rightarrow r_1 | c_2 \Rightarrow r_2 | ... | c_n \Rightarrow r_n)$.

The following table summarizes the primitive data types used by MPIR.

Data Type	**Notation**	**Description**
character	char	a single symbol or digit
integer	$\mathbb{Z}$	a number without a fractional component in (-∞, ∞)
natural number	$\mathbb{N}$	a number without a fractional component in [1, ∞)
real	$\mathbb{R}$	any number in (-∞, ∞)

The specification of MPIR uses some derived data types: sequences, strings, and tuples. Sequences are lists filled with elements of the same data type. Strings are sequences of characters. Tuples contain a list of values, potentially of different types. In addition, MPIR uses functions, which are defined by the data types of their inputs and outputs. Local functions are described by giving their type signature followed by their specification.

Variables of matrices or vectors are in math bold face. For any matrix $\mathbf{A}$ or vector $\mathbf{b}$, one with subscript $\mathbf{A}_i$ or $\mathbf{b}_i$ always means "the ith matrix/vector". $a_{i,j}$ or b_i is used to reference "the element at row i column j in matrix $\mathbf{A}$" or "the ith element in vector $\mathbf{b}$".

5 Module Decomposition

The following table is taken directly from the Module Guide document for this project.

Level 1	**Level 2**
Hardware-Hiding Module	–
Behaviour-Hiding Module	Floating Point Concepts Module Matrix Operations Module Factorization Module Iterative Solver Module
Software Decision Module	

Table 1: Module Hierarchy

6 MIS of Factorization Module

6.1 Module

qdldl[1]

6.2 Uses

None

6.3 Syntax

6.3.1 Exported Access Programs

Name	In	Out	Exceptions
QDLDL_etree	$\mathbf{A} : \mathbb{R}^{n \times n}$	$L_{\text{nz}} : \mathbb{N}, \mathbf{E} : \mathbb{R}^n$	NOT_UPPER
QDLDL_factor	$\mathbf{A} : \mathbb{R}^{n \times n}, L_{\text{nz}} : \mathbb{N}, \mathbf{E} : \mathbb{R}^n, u_f$	$\mathbf{L} : \mathbb{R}^{n \times n}, \mathbf{d} : \mathbb{R}^n$	FAC_FAILED
QDLDL_solve	$\mathbf{L} : \mathbb{R}^{n \times n}, \mathbf{d} : \mathbb{R}^n, \mathbf{b} : \mathbb{R}^n, u_w$	$\mathbf{x} : \mathbb{R}^n$	–

6.4 Semantics

6.4.1 State Variables

None

6.4.2 Assumptions

The (sparse) matrix used or returned by this module is stored in Compressed Sparse Column (CSC) format (*Compressed Sparse Column Format (CSC) — Scipy lecture notes* 2025).

6.4.3 Access Routine Semantics

QDLDL_etree($\mathbf{A}$):

- output: $\mathbf{E} :=$ elimination tree[2] for the factorization $\mathbf{A} = \mathbf{LDL}^\intercal$, $L_{\text{nz}} :=$ the number of non-zeros in the $\mathbf{L}$ factor.
- exception: $err :=$ (entries found in lower triangle $\implies$ NOT_UPPER)

[1] This module was originally implemented in prior research work (Shahrooz Derakhshan et al. 2023), and the corresponding source code is provided as part of this project. The specifications presented here describe the exported routines and their intended usage, without delving into the underlying mathematical models or state transitions.

[2] An elimination tree is a directed tree that encodes the dependencies between columns of a sparse symmetric matrix during factorization. Each node in the tree corresponds to a column of the matrix, and the edges represent the flow of fill-ins (new non-zero entries) created during the elimination process.

QDLDL_factor(u_f)[3]($\mathbf{A}, L_{\text{nz}}, \mathbf{E}$):

- output: $\mathbf{L}, \mathbf{D} \coloneqq$ factors of $\mathbf{A}$ in u_f precision, where diagonal matrix $\mathbf{D}$ is simply represented by a vector $\mathbf{d}$ as there's only non-zero elements along the diagonal.
- exception: $err \coloneqq ((\exists\, i : \mathbf{d}_i = 0) \implies \text{FAC\_FAILED})$

QDLDL_solve(u_w)($\mathbf{L}, \mathbf{d}, \mathbf{b}$):

- output: solves $\mathbf{Ax} = \mathbf{LDL}^\intercal\mathbf{x} = \mathbf{b}$ in u_w precision, where $\mathbf{D}$ is reinterpreted from $\mathbf{d}$. Let $\mathbf{y} = \mathbf{DL}^\intercal\mathbf{x}$, we have $\mathbf{LDL}^\intercal\mathbf{x} = \mathbf{L}(\mathbf{DL}^\intercal\mathbf{x}) = \mathbf{Ly}$. Solve $\mathbf{Ly} = \mathbf{b}$ by back substitution (triangular solve). Solve $\mathbf{DL}^\intercal\mathbf{x}$ by multiplying $\mathbf{y}$ with $\frac{1}{d_i}$ followed by another triangular solve.

[3] A generic routine with type parameter u_f

7 MIS of Floating Point Concepts Module

7.1 Module

fp_concepts

7.2 Uses

None

7.3 Syntax

7.3.1 Exported Access Programs

Name	In	Out	Exceptions
FloatingPoint	tuple of $(T_1, T_2, \ldots, T_n), n \geq 1$	$out : \mathbb{B}$	–
PartialOrdered	tuple of $(T_1, T_2, \ldots, T_n), n \geq 1$	$out : \mathbb{B}$	–
Refinable	tuple of $(T_1, T_2, \ldots, T_n), n \geq 1$	$out : \mathbb{B}$	–

$(T_1, T_2, \ldots, T_n), n \geq 1$ is a variadic tuple of types, where T_n is some generic type, possible values include: `int`, `double`, `float`, …

7.4 Semantics

7.4.1 State Variables

None

7.4.2 Assumptions

None

7.4.3 Access Routine Semantics

FloatingPoint$(T_1, T_2, \ldots, T_n)$:

- output: $out := (\forall\, i : \mathbb{N} \,|\, 1 \leq i \leq n : T_i \text{ is a floating point type})$

PartialOrdered$(T_1, T_2, \ldots, T_n)$:

- output: $out := (\forall\, i : \mathbb{N} \,|\, 1 \leq i < n : \text{machine epsilon of } T_i > \text{machine epsilon of } T_{i+1})$

Refinable$(T_1, T_2, \ldots, T_n)$:

- output: $out := \text{FloatingPoint}(T_1, T_2, \ldots, T_n) \wedge \text{PartialOrdered}(T_1, T_2, \ldots, T_n)$

8 MIS of Matrix Operations Module

8.1 Module

ops

8.2 Uses

fp_concepts (Section 7)

8.3 Syntax

8.3.1 Exported Access Programs

Name	In	Out	Exceptions
MatrixMuliply((T, T_a, T_x) with Refinable(T_a, T_x, T))	$\mathbf{A} : \mathbb{R}^{n \times n}, \mathbf{x} : \mathbb{R}^n$	$\mathbf{res} : \mathbb{R}^n$	–
VectorAdd((T, T_a, T_b) with Refinable(T_a, T) $\wedge$ Refinable(T_b, T))	$\mathbf{a} : \mathbb{R}^n, \mathbf{b} : \mathbb{R}^n$	$\mathbf{res} : \mathbb{R}^n$	–
VectorSubtract((T, T_a, T_b) with Refinable(T_a, T) $\wedge$ Refinable(T_b, T))	$\mathbf{a} : \mathbb{R}^n, \mathbf{b} : \mathbb{R}^n$	$\mathbf{res} : \mathbb{R}^n$	–
VectorMultiply((T, T_a, T_b) with Refinable(T_a, T) $\wedge$ Refinable(T_b, T))	$\mathbf{a} : \mathbb{R}^n, \mathbf{b} : \mathbb{R}^n$	$\mathbf{res} : \mathbb{R}^n$	–
VectorScale((T, T_a, T_b) with Refinable(T_a, T) $\wedge$ Refinable(T_b, T))	$\mathbf{a} : \mathbb{R}^n, b : \mathbb{R}$	$\mathbf{res} : \mathbb{R}^n$	–
VectorDot((T, T_a, T_b) with Refinable(T_a, T) $\wedge$ Refinable(T_b, T))	$\mathbf{a} : \mathbb{R}^n, \mathbf{b} : \mathbb{R}^n$	$\mathbf{res} : \mathbb{R}^n$	–
Dnrm2((T, T_x) with Refinable(T_x, T))	$\mathbf{x} : \mathbb{R}^n$	$norm : \mathbb{R}$	–
InfNrm((T, T_x) with Refinable(T_x, T))	$\mathbf{x} : \mathbb{R}^n$	$norm : \mathbb{R}$	–

8.4 Semantics

8.4.1 State Variables

None

8.4.2 Assumptions

The (sparse) matrix used by this module is stored in CSC format.

8.4.3 Access Routine Semantics

MatrixMuliply((T, T_a, T_x) with Refinable(T_a, T_x, T))$(\mathbf{A}, \mathbf{x})$:

- output: $res_i \coloneqq \sum_{j=1}^{n} a_{i,j} x_j$ for $i = 1, 2, \ldots, n$, where $a_{i,j}$ is in precision T_a, x_j is in precision T_x, and res_i is in precision T.

VectorAdd((T, T_a, T_b) with Refinable$(T_a, T) \wedge$ Refinable(T_b, T))$(\mathbf{a}, \mathbf{b})$:

- output: $res_i \coloneqq a_i + b_i$ for $i = 1, 2, \ldots, n$, where a_i is in precision T_a, b_i is in precision T_b, and res_i is in precision T.

VectorSubtract((T, T_a, T_b) with Refinable$(T_a, T) \wedge$ Refinable(T_b, T))$(\mathbf{a}, \mathbf{b})$:

- output: $res_i \coloneqq a_i - b_i$ for $i = 1, 2, \ldots, n$, where a_i is in precision T_a, b_i is in precision T_b, and res_i is in precision T.

VectorMultiply((T, T_a, T_b) with Refinable$(T_a, T) \wedge$ Refinable(T_b, T))$(\mathbf{a}, \mathbf{b})$:

- output: $res_i \coloneqq a_i b_i$ for $i = 1, 2, \ldots, n$, where a_i is in precision T_a, b_i is in precision T_b, and res_i is in precision T.

VectorScale((T, T_a, T_b) with Refinable$(T_a, T) \wedge$ Refinable(T_b, T))$(\mathbf{a}, b)$:

- output: $res_i \coloneqq b a_i$ for $i = 1, 2, \ldots, n$, where a_i is in precision T_a, b is in precision T_b, and res_i is in precision T.

VectorDot((T, T_a, T_b) with Refinable$(T_a, T) \wedge$ Refinable(T_b, T))$(\mathbf{a}, \mathbf{b})$:

- output: $res \coloneqq \sum_{i=1}^{n} a_i b_i$, where a_i is in precision T_a, b_i is in precision T_b, and res is in precision T.

Dnrm2((T, T_x) with Refinable(T_x, T))$(\mathbf{x})$:

- output: $norm \coloneqq \sqrt{\sum_{i=1}^{n} x_i^2}$, where x_i is in precision T_x, and $norm$ is in precision T.

InfNrm((T, T_x) with Refinable(T_x, T))$(\mathbf{x})$:

- output: $norm \coloneqq \max_{1 \leq i \leq n} |x_i|$, where x_i is in precision T_x, and $norm$ is in precision T.

9 MIS of Iterative Solver Module

9.1 Module

GmresLDLIR(u_f, u_w, u_r) with Refinable(u_f, u_w, u_r)

9.2 Uses

qdldl (Section 6), fp_concepts (Section 7), ops (Section 8)

9.3 Syntax

9.3.1 Exported Access Programs

Name	In	Out	Exceptions
Compute	$\mathbf{A} : \mathbb{R}^{n\times n}$	–	–
Solve	$\mathbf{b} : \mathbb{R}^n$	$\mathbf{x} : \mathbb{R}^n$	–
SetMaxIRIterations	$n : \mathbb{N}$	–	–
SetMaxGmresIterations	$n : \mathbb{N}$	–	–
SetTolerance	$\epsilon : \mathbb{R}$	–	–

9.4 Semantics

9.4.1 State Variables

n_: $\mathbb{N}$, size of the matrix $\mathbf{A}$
Ap_: $\mathbb{N}^n$, column pointers of $\mathbf{A}$ (part of CSC format)
Ai_: $\mathbb{N}^{n_{\text{nz}}}$, row indices of $\mathbf{A}$ (part of CSC format), n_{nz} is the number of non-zeros
Ax_: $\mathbb{R}^{n_{\text{nz}}}$, non-zero values of $\mathbf{A}$ (part of CSC format), in precision u_w
Lp_: $\mathbb{N}^n$, column pointers of the $\mathbf{L}$ factor
Li_: $\mathbb{N}^{L_{\text{nz}}}$, row indices of the $\mathbf{L}$ factor
Lx_: $\mathbb{R}^{L_{\text{nz}}}$, non-zero values of the $\mathbf{L}$ factor in precision u_f
Dinv_: $\mathbb{R}^n$, non-zero values of the inverse of the $\mathbf{D}$ factor in precision u_f
ir_iter_: $\mathbb{N}$, maximum refinement iterations, default to 10
gmres_iter_: $\mathbb{N}$, maximum GMRES iterations, default to 10
tol_: $\mathbb{R}$, tolerance in precision u_r, default to 1×10^{-10}

9.4.2 Assumptions

The input matrix $\mathbf{A}$ is non-singular, symmetric quasi-definite, and is stored in CSC format.

9.4.3 Access Routine Semantics

SetMaxIRIterations(n):

- transition: ir_iter_ $\coloneqq n$

SetMaxGmresIterations(n):

- transition: gmres_iter_ $\coloneqq n$

SetTolerance(ϵ):

- transition: tol_ $\coloneqq \epsilon$

Compute($\mathbf{A}$):

- transition:
  n_ $\coloneqq$ size of $\mathbf{A}$,
  Ap_, Ai_, Ax_ $\coloneqq \mathbf{A}$,
  Lp_, Li_, Lx_, 1 / Dinv_ $\coloneqq$ QDLDL_factor(u_f)($\mathbf{A}, L_{\text{nz}}, \mathbf{E}$),
  where $L_{\text{nz}}, \mathbf{E} \coloneqq$ QDLDL_etree(u_f)($\mathbf{A}$)

Solve($\mathbf{b}$):

- output: solves $\mathbf{Ax} = \mathbf{b}$ in u_w precision following Algorithm 1.

9.4.4 Algorithms

These are copied from IM1 in the SRS (Ye 2025b).

Algorithm 1 GMRES-IR with $\mathbf{LDL}^\intercal$ factorization in MP

Perform $\mathbf{LDL}^\intercal$ factorization of $\mathbf{A}$ ▷ at u_f
Solve $\mathbf{LDL}^\intercal\mathbf{x}_0 = \mathbf{b}$ ▷ at u_f
for $i \leftarrow 0, 1, \ldots, n_{\text{iter}}$ and $\|r_i\|_2 \geq \epsilon$ **do**
    $r_i \leftarrow \mathbf{b} - \mathbf{A}\mathbf{x}_i$ ▷ at u_r
    Solve $(\mathbf{LDL}^\intercal)^{-1}\mathbf{A}\mathbf{d}_i = (\mathbf{LDL}^\intercal)^{-1}\mathbf{r_i}$ with GMRES (See Algorithm 2)
     where $\mathbf{M}^{-1} = (\mathbf{LDL}^\intercal)^{-1}$ ▷ at u_w
    $\mathbf{x}_{i+1} = \mathbf{x}_i + \mathbf{d}_i$ ▷ at u_w
end for

Algorithm 2 Restarted GMRES with left preconditioning

$\mathbf{A} \in \mathbb{R}^{n \times n}, \quad \mathbf{x}_0, \mathbf{b} \in \mathbb{R}^n, \quad \mathbf{M}^{-1} \approx \mathbf{A}^{-1}$

for $k \leftarrow 1, 2, \ldots,$ the kth restart **do**

$\mathbf{z}_k \leftarrow \mathbf{b} - \mathbf{A}\mathbf{x}_k$ ▷ Compute residual

$\mathbf{r}_k \leftarrow \mathbf{M}^{-1}\mathbf{z}_k$ ▷ Apply preconditioning

$\beta \leftarrow \|\mathbf{r}_k\|_2, \quad \mathbf{v}_1 = \mathbf{r}_k/\beta, \quad \mathbf{V}_1 \leftarrow [\mathbf{v}_1]$ ▷ Setup for Arnoldi process

Construct orthogonal basis of preconditioned Krylov subspace

$$\mathcal{K}_m(\mathbf{M}^{-1}\mathbf{A}, \mathbf{r}_k) = \operatorname{span}\{\mathbf{r}_k, \mathbf{M}^{-1}\mathbf{A}\mathbf{r}_k, \ldots, (\mathbf{M}^{-1}\mathbf{A})^{m-1}\mathbf{r}_k\}$$

The Arnoldi process gives

$$\mathbf{M}^{-1}\mathbf{A}\mathbf{V}_m = \mathbf{V}_{m+1}\bar{\mathbf{H}}_m$$

where $\mathbf{V}_m \in \mathbb{R}^{n \times m}$ contains orthonormal basis vectors for $\mathcal{K}_m$, $\mathbf{V}_{m+1} \in \mathbb{R}^{n \times (m+1)}$ extends $\mathbf{V}_m$ with one more vector, $\bar{\mathbf{H}}_m \in \mathbb{R}^{(m+1) \times m}$ is an upper Hessenberg matrix

Solve the least square problem

$$\min_{\mathbf{y}_m \in \mathbb{R}^m} \left\| \beta e_1 - \bar{\mathbf{H}}_m \mathbf{y}_m \right\|_2$$

where $e_1 \in \mathbb{R}^{m+1}$ is the first standard basis vector $e_1 = [1, 0, 0, \ldots, 0]^\intercal$

$\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{V}_m\mathbf{y}_m$ ▷ Add the correction

end for

References

Compressed Sparse Column Format (CSC) — Scipy lecture notes (2025). URL: https://scipy-lectures.org/advanced/scipy_sparse/csc_matrix.html (visited on 03/26/2025).

Ghezzi, Carlo, Mehdi Jazayeri, and Dino Mandrioli (2003). *Fundamentals of Software Engineering*. 2nd. Upper Saddle River, NJ, USA: Prentice Hall.

Hoffman, Daniel M. and Paul A. Strooper (1995). *Software Design, Automated Testing, and Maintenance: A Practical Approach*. New York, NY, USA: International Thomson Computer Press. URL: http://%20citeseer.ist.psu.edu/428727.html.

Shahrooz Derakhshan et al. (Dec. 27, 2023). *Using Mixed-Precision in the Linear Solver of Ipopt and in QDLDL*. 2. Hamilton, Ontario, Canada: McMaster University, p. 19.

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

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