hypre **Reference Manual**

— Version 2.10.1 —

Contents

1 Struct System Interface — *A structured-grid conceptual interface* 5
1.1 Struct Grids — 5
1.2 Struct Stencils — 8
1.3 Struct Matrices — 9
1.4 Struct Vectors — 16

2 SStruct System Interface — *A semi-structured-grid conceptual interface* 21
2.1 SStruct Grids — 21
2.2 SStruct Stencils — 28
2.3 SStruct Graphs — 30
2.4 SStruct Matrices — 34
2.5 SStruct Vectors — 41

3 IJ System Interface — *A linear-algebraic conceptual interface* 48
3.1 IJ Matrices — 48
3.2 IJ Vectors — 56

4 Struct Solvers — *Linear solvers for structured grids* 62
4.1 Struct Solvers — 62
4.2 Struct Jacobi Solver — 63
4.3 Struct PFMG Solver — 66
4.4 Struct SMG Solver — 73
4.5 Struct CycRed Solver — 78
4.6 Struct PCG Solver — 81
4.7 Struct GMRES Solver — 82
4.8 Struct FlexGMRES Solver — 83
4.9 Struct LGMRES Solver — 84
4.10 Struct BiCGSTAB Solver — 85
4.11 Struct Hybrid Solver — 85
4.12 Struct LOBPCG Eigensolver — 92

5 SStruct Solvers — *Linear solvers for semi-structured grids* 94
5.1 SStruct Solvers — 94
5.2 SStruct SysPFMG Solver — 95
5.3 SStruct Split Solver — 101
5.4 SStruct FAC Solver — 105
5.5 SStruct Maxwell Solver — 114
5.6 SStruct PCG Solver — 121
5.7 SStruct GMRES Solver — 122
5.8 SStruct FlexGMRES Solver — 123
5.9 SStruct LGMRES Solver — 124
5.10 SStruct BiCGSTAB Solver — 125
5.11 SStruct LOBPCG Eigensolver — 126

6 ParCSR Solvers — *Linear solvers for sparse matrix systems* 128
6.1 ParCSR Solvers — 129
6.2 ParCSR BoomerAMG Solver and Preconditioner — 129
6.3 ParCSR ParaSails Preconditioner — 163
6.4 ParCSR Euclid Preconditioner — 168
6.5 ParCSR Pilut Preconditioner — 173

	6.6	ParCSR AMS Solver and Preconditioner —	175
	6.7	ParCSR ADS Solver and Preconditioner —	186
	6.8	ParCSR PCG Solver —	193
	6.9	ParCSR GMRES Solver —	195
	6.10	ParCSR FlexGMRES Solver —	196
	6.11	ParCSR LGMRES Solver —	197
	6.12	ParCSR BiCGSTAB Solver —	197
	6.13	ParCSR Hybrid Solver —	198
	6.14	ParCSR LOBPCG Eigensolver —	215
7	**Krylov Solvers** — ***A basic interface for Krylov solvers***		216
	7.1	Krylov Solvers —	216
	7.2	PCG Solver —	217
	7.3	GMRES Solver —	225
	7.4	FlexGMRES Solver —	232
	7.5	LGMRES Solver —	238
	7.6	BiCGSTAB Solver —	244
	7.7	CGNR Solver —	249
8	**Eigensolvers** — ***A basic interface for eigensolvers***		253
	8.1	EigenSolvers —	253
	8.2	LOBPCG Eigensolver —	254
9	**Finite Element Interface** — ***A finite element-based conceptual interface***		259
	9.1	FEI Functions —	259
	9.2	FEI Solver Parameters —	269

1 Struct System Interface

Names

1.1 **Struct Grids** 5

1.2 **Struct Stencils** 8

1.3 **Struct Matrices** 9

1.4 **Struct Vectors** 16

This interface represents a structured-grid conceptual view of a linear system.

1.1 Struct Grids

Names

1.1.1 typedef struct hypre_StructGrid_struct ***HYPRE_StructGrid**
A grid object is constructed out of several "boxes", defined on a global abstract index space 6

1.1.2 int
HYPRE_StructGridCreate (MPI_Comm comm, int ndim, HYPRE_StructGrid* grid)
Create an `ndim`*-dimensional grid object* 6

1.1.3 int
HYPRE_StructGridDestroy (HYPRE_StructGrid grid)
Destroy a grid object. 6

1.1.4 int
HYPRE_StructGridSetExtents (HYPRE_StructGrid grid, int* ilower, int* iupper)
Set the extents for a box on the grid 7

1.1.5 int
HYPRE_StructGridAssemble (HYPRE_StructGrid grid)
Finalize the construction of the grid before using 7

1.1.6 int

HYPRE_StructGridSetPeriodic (HYPRE_StructGrid grid, int* periodic)
Set the periodicity for the grid. 7

1.1.7 int
HYPRE_StructGridSetNumGhost (HYPRE_StructGrid grid, int* num_ghost)
Set the ghost layer in the grid object 7

1.1.1

typedef struct hypre_StructGrid_struct ***HYPRE_StructGrid**

A grid object is constructed out of several "boxes", defined on a global abstract index space

1.1.2

int
HYPRE_StructGridCreate (MPI_Comm comm, int ndim, HYPRE_StructGrid* grid)

Create an `ndim`-dimensional grid object

1.1.3

int **HYPRE_StructGridDestroy** (HYPRE_StructGrid grid)

Destroy a grid object. An object should be explicitly destroyed using this destructor when the user's code no longer needs direct access to it. Once destroyed, the object must not be referenced again. Note that the object may not be deallocated at the completion of this call, since there may be internal package references to the object. The object will then be destroyed when all internal reference counts go to zero.

1.1.4

int
HYPRE_StructGridSetExtents (HYPRE_StructGrid grid, int* ilower, int* iupper)

Set the extents for a box on the grid

1.1.5

int **HYPRE_StructGridAssemble** (HYPRE_StructGrid grid)

Finalize the construction of the grid before using

1.1.6

int **HYPRE_StructGridSetPeriodic** (HYPRE_StructGrid grid, int* periodic)

Set the periodicity for the grid.

The argument `periodic` is an `ndim`-dimensional integer array that contains the periodicity for each dimension. A zero value for a dimension means non-periodic, while a nonzero value means periodic and contains the actual period. For example, periodicity in the first and third dimensions for a 10x11x12 grid is indicated by the array [10,0,12].

NOTE: Some of the solvers in hypre have power-of-two restrictions on the size of the periodic dimensions.

1.1.7

int
HYPRE_StructGridSetNumGhost (HYPRE_StructGrid grid, int* num_ghost)

Set the ghost layer in the grid object

1.2

Struct Stencils

Names

1.2.1 typedef struct hypre_StructStencil_struct ***HYPRE_StructStencil**
The stencil object 8

1.2.2 int
HYPRE_StructStencilCreate (int ndim, int size,
HYPRE_StructStencil* stencil)
Create a stencil object for the specified number of spatial dimensions and stencil entries 8

1.2.3 int
HYPRE_StructStencilDestroy (HYPRE_StructStencil stencil)
Destroy a stencil object 9

1.2.4 int
HYPRE_StructStencilSetElement (HYPRE_StructStencil stencil, int entry,
int* offset)
Set a stencil entry. 9

1.2.1

typedef struct hypre_StructStencil_struct ***HYPRE_StructStencil**

The stencil object

1.2.2

int
HYPRE_StructStencilCreate (int ndim, int size, HYPRE_StructStencil* stencil)

Create a stencil object for the specified number of spatial dimensions and stencil entries

1.2.3

int **HYPRE_StructStencilDestroy** (HYPRE_StructStencil stencil)

Destroy a stencil object

1.2.4

int
HYPRE_StructStencilSetElement (HYPRE_StructStencil stencil, int entry, int* offset)

Set a stencil entry.

NOTE: The name of this routine will eventually be changed to `HYPRE_StructStencilSetEntry`.

1.3 Struct Matrices

Names

1.3.1 typedef struct hypre_StructMatrix_struct ***HYPRE_StructMatrix**
The matrix object 11

1.3.2 int
HYPRE_StructMatrixCreate (MPI_Comm comm, HYPRE_StructGrid grid, HYPRE_StructStencil stencil, HYPRE_StructMatrix* matrix)
Create a matrix object 11

1.3.3 int
HYPRE_StructMatrixDestroy (HYPRE_StructMatrix matrix)
Destroy a matrix object 11

1.3.4 int
HYPRE_StructMatrixInitialize (HYPRE_StructMatrix matrix)
Prepare a matrix object for setting coefficient values 12

1.3.5 int
HYPRE_StructMatrixSetValues (HYPRE_StructMatrix matrix, int* index, int nentries, int* entries, HYPRE_Complex* values)
Set matrix coefficients index by index. 12

1.3.6 int

HYPRE_StructMatrixAddToValues (HYPRE_StructMatrix matrix, int* index, int nentries, int* entries, HYPRE_Complex* values)
Add to matrix coefficients index by index. 12

1.3.7 int
HYPRE_StructMatrixSetConstantValues (HYPRE_StructMatrix matrix, int nentries, int* entries, HYPRE_Complex* values)
Set matrix coefficients which are constant over the grid. 12

1.3.8 int
HYPRE_StructMatrixAddToConstantValues (HYPRE_StructMatrix matrix, int nentries, int* entries, HYPRE_Complex* values)
Add to matrix coefficients which are constant over the grid. 13

1.3.9 int
HYPRE_StructMatrixSetBoxValues (HYPRE_StructMatrix matrix, int* ilower, int* iupper, int nentries, int* entries, HYPRE_Complex* values)
Set matrix coefficients a box at a time. 13

1.3.10 int
HYPRE_StructMatrixAddToBoxValues (HYPRE_StructMatrix matrix, int* ilower, int* iupper, int nentries, int* entries, HYPRE_Complex* values)
Add to matrix coefficients a box at a time. 13

1.3.11 int
HYPRE_StructMatrixAssemble (HYPRE_StructMatrix matrix)
Finalize the construction of the matrix before using 14

1.3.12 int
HYPRE_StructMatrixGetValues (HYPRE_StructMatrix matrix, int* index, int nentries, int* entries, HYPRE_Complex* values)
Get matrix coefficients index by index. 14

1.3.13 int
HYPRE_StructMatrixGetBoxValues (HYPRE_StructMatrix matrix, int* ilower, int* iupper, int nentries, int* entries, HYPRE_Complex* values)
Get matrix coefficients a box at a time. 14

1.3.14 int
HYPRE_StructMatrixSetSymmetric (HYPRE_StructMatrix matrix, int symmetric)
Define symmetry properties of the matrix. 14

1.3.15 int
HYPRE_StructMatrixSetConstantEntries (HYPRE_StructMatrix matrix, int nentries, int* entries)
Specify which stencil entries are constant over the grid. 15

1.3.16 int

HYPRE_StructMatrixSetNumGhost (HYPRE_StructMatrix matrix, int* num_ghost)
Set the ghost layer in the matrix 15

1.3.17 int
HYPRE_StructMatrixPrint (const char* filename, HYPRE_StructMatrix matrix, int all)
Print the matrix to file. 15

1.3.18 int
HYPRE_StructMatrixMatvec (HYPRE_Complex alpha, HYPRE_StructMatrix A, HYPRE_StructVector x, HYPRE_Complex beta, HYPRE_StructVector y)
Matvec operator. 16

1.3.1

typedef struct hypre_StructMatrix_struct ***HYPRE_StructMatrix**

The matrix object

1.3.2

int
HYPRE_StructMatrixCreate (MPI_Comm comm, HYPRE_StructGrid grid, HYPRE_StructStencil stencil, HYPRE_StructMatrix* matrix)

Create a matrix object

1.3.3

int **HYPRE_StructMatrixDestroy** (HYPRE_StructMatrix matrix)

Destroy a matrix object

1.3.4

int **HYPRE_StructMatrixInitialize** (HYPRE_StructMatrix matrix)

Prepare a matrix object for setting coefficient values

1.3.5

int
HYPRE_StructMatrixSetValues (HYPRE_StructMatrix matrix, int* index, int nentries, int* entries, HYPRE_Complex* values)

Set matrix coefficients index by index. The `values` array is of length `nentries`.

NOTE: For better efficiency, use HYPRE_StructMatrixSetBoxValues to set coefficients a box at a time.

1.3.6

int
HYPRE_StructMatrixAddToValues (HYPRE_StructMatrix matrix, int* index, int nentries, int* entries, HYPRE_Complex* values)

Add to matrix coefficients index by index. The `values` array is of length `nentries`.

NOTE: For better efficiency, use HYPRE_StructMatrixAddToBoxValues to set coefficients a box at a time.

1.3.7

int
HYPRE_StructMatrixSetConstantValues (HYPRE_StructMatrix matrix, int nentries, int* entries, HYPRE_Complex* values)

Set matrix coefficients which are constant over the grid. The `values` array is of length `nentries`.

1.3.8

int
HYPRE_StructMatrixAddToConstantValues (HYPRE_StructMatrix matrix, int nentries, int* entries, HYPRE_Complex* values)

Add to matrix coefficients which are constant over the grid. The `values` array is of length `nentries`.

1.3.9

int
HYPRE_StructMatrixSetBoxValues (HYPRE_StructMatrix matrix, int* ilower, int* iupper, int nentries, int* entries, HYPRE_Complex* values)

Set matrix coefficients a box at a time. The data in `values` is ordered as follows:

```
m = 0;
for (k = ilower[2]; k <= iupper[2]; k++)
   for (j = ilower[1]; j <= iupper[1]; j++)
      for (i = ilower[0]; i <= iupper[0]; i++)
         for (entry = 0; entry < nentries; entry++)
         {
            values[m] = ...;
            m++;
         }
```

1.3.10

int
HYPRE_StructMatrixAddToBoxValues (HYPRE_StructMatrix matrix, int* ilower, int* iupper, int nentries, int* entries, HYPRE_Complex* values)

Add to matrix coefficients a box at a time. The data in `values` is ordered as in HYPRE_StructMatrixSetBoxValues.

1.3.11

int **HYPRE_StructMatrixAssemble** (HYPRE_StructMatrix matrix)

Finalize the construction of the matrix before using

1.3.12

int
HYPRE_StructMatrixGetValues (HYPRE_StructMatrix matrix, int* index, int nentries, int* entries, HYPRE_Complex* values)

Get matrix coefficients index by index. The `values` array is of length `nentries`.

NOTE: For better efficiency, use HYPRE_StructMatrixGetBoxValues to get coefficients a box at a time.

1.3.13

int
HYPRE_StructMatrixGetBoxValues (HYPRE_StructMatrix matrix, int* ilower, int* iupper, int nentries, int* entries, HYPRE_Complex* values)

Get matrix coefficients a box at a time. The data in `values` is ordered as in HYPRE_StructMatrixSetBoxValues.

1.3.14

int
HYPRE_StructMatrixSetSymmetric (HYPRE_StructMatrix matrix, int symmetric)

Define symmetry properties of the matrix. By default, matrices are assumed to be nonsymmetric. Significant storage savings can be made if the matrix is symmetric.

1.3.15

int
HYPRE_StructMatrixSetConstantEntries (HYPRE_StructMatrix matrix, int nentries, int* entries)

Specify which stencil entries are constant over the grid. Declaring entries to be "constant over the grid" yields significant memory savings because the value for each declared entry will only be stored once. However, not all solvers are able to utilize this feature.

Presently supported:

- no entries constant (this function need not be called)
- all entries constant
- all but the diagonal entry constant

1.3.16

int
HYPRE_StructMatrixSetNumGhost (HYPRE_StructMatrix matrix, int* num_ghost)

Set the ghost layer in the matrix

1.3.17

int
HYPRE_StructMatrixPrint (const char* filename, HYPRE_StructMatrix matrix, int all)

Print the matrix to file. This is mainly for debugging purposes.

1.3.18

int
HYPRE_StructMatrixMatvec (HYPRE_Complex alpha, HYPRE_StructMatrix A, HYPRE_StructVector x, HYPRE_Complex beta, HYPRE_StructVector y)

Matvec operator. This operation is $y = \alpha A x + \beta y$. Note that you can do a simple matrix-vector multiply by setting $\alpha = 1$ and $\beta = 0$.

1.4

Struct Vectors

Names

1.4.1 typedef struct hypre_StructVector_struct ***HYPRE_StructVector**
The vector object 17

1.4.2 int
HYPRE_StructVectorCreate (MPI_Comm comm, HYPRE_StructGrid grid, HYPRE_StructVector* vector)
Create a vector object 17

1.4.3 int
HYPRE_StructVectorDestroy (HYPRE_StructVector vector)
Destroy a vector object 17

1.4.4 int
HYPRE_StructVectorInitialize (HYPRE_StructVector vector)
Prepare a vector object for setting coefficient values 18

1.4.5 int
HYPRE_StructVectorSetValues (HYPRE_StructVector vector, int* index, HYPRE_Complex value)
Set vector coefficients index by index. 18

1.4.6 int
HYPRE_StructVectorAddToValues (HYPRE_StructVector vector, int* index, HYPRE_Complex value)
Add to vector coefficients index by index. 18

1.4.7 int
HYPRE_StructVectorSetBoxValues (HYPRE_StructVector vector, int* ilower, int* iupper, HYPRE_Complex* values)
Set vector coefficients a box at a time. 18

1.4.8 int
HYPRE_StructVectorAddToBoxValues (HYPRE_StructVector vector, int* ilower, int* iupper, HYPRE_Complex* values)
Add to vector coefficients a box at a time. 19

1.4.9 int

HYPRE_StructVectorAssemble (HYPRE_StructVector vector)
Finalize the construction of the vector before using 19

1.4.10 int
HYPRE_StructVectorGetValues (HYPRE_StructVector vector, int* index, HYPRE_Complex* value)
Get vector coefficients index by index. 19

1.4.11 int
HYPRE_StructVectorGetBoxValues (HYPRE_StructVector vector, int* ilower, int* iupper, HYPRE_Complex* values)
Get vector coefficients a box at a time. 20

1.4.12 int
HYPRE_StructVectorPrint (const char* filename, HYPRE_StructVector vector, int all)
Print the vector to file. 20

1.4.1

typedef struct hypre_StructVector_struct ***HYPRE_StructVector**

The vector object

1.4.2

int
HYPRE_StructVectorCreate (MPI_Comm comm, HYPRE_StructGrid grid, HYPRE_StructVector* vector)

Create a vector object

1.4.3

int **HYPRE_StructVectorDestroy** (HYPRE_StructVector vector)

Destroy a vector object

1.4.4

int **HYPRE_StructVectorInitialize** (HYPRE_StructVector vector)

Prepare a vector object for setting coefficient values

1.4.5

int
HYPRE_StructVectorSetValues (HYPRE_StructVector vector, int* index, HYPRE_Complex value)

Set vector coefficients index by index.

NOTE: For better efficiency, use HYPRE_StructVectorSetBoxValues to set coefficients a box at a time.

1.4.6

int
HYPRE_StructVectorAddToValues (HYPRE_StructVector vector, int* index, HYPRE_Complex value)

Add to vector coefficients index by index.

NOTE: For better efficiency, use HYPRE_StructVectorAddToBoxValues to set coefficients a box at a time.

1.4.7

int
HYPRE_StructVectorSetBoxValues (HYPRE_StructVector vector, int* ilower, int* iupper, HYPRE_Complex* values)

Set vector coefficients a box at a time. The data in `values` is ordered as follows:

```
m = 0;
for (k = ilower[2]; k <= iupper[2]; k++)
   for (j = ilower[1]; j <= iupper[1]; j++)
      for (i = ilower[0]; i <= iupper[0]; i++)
      {
         values[m] = ...;
         m++;
      }
```

1.4.8

int
HYPRE_StructVectorAddToBoxValues (HYPRE_StructVector vector, int* ilower, int* iupper, HYPRE_Complex* values)

Add to vector coefficients a box at a time. The data in `values` is ordered as in HYPRE_StructVectorSetBoxValues.

1.4.9

int **HYPRE_StructVectorAssemble** (HYPRE_StructVector vector)

Finalize the construction of the vector before using

1.4.10

int
HYPRE_StructVectorGetValues (HYPRE_StructVector vector, int* index, HYPRE_Complex* value)

Get vector coefficients index by index.

NOTE: For better efficiency, use HYPRE_StructVectorGetBoxValues to get coefficients a box at a time.

1.4.11

int
HYPRE_StructVectorGetBoxValues (HYPRE_StructVector vector, int* ilower, int* iupper, HYPRE_Complex* values)

Get vector coefficients a box at a time. The data in `values` is ordered as in HYPRE_StructVectorSetBoxValues.

1.4.12

int
HYPRE_StructVectorPrint (const char* filename, HYPRE_StructVector vector, int all)

Print the vector to file. This is mainly for debugging purposes.

2 SStruct System Interface

Names

2.1 **SStruct Grids** 21

2.2 **SStruct Stencils** 28

2.3 **SStruct Graphs** 30

2.4 **SStruct Matrices** 34

2.5 **SStruct Vectors** 41

This interface represents a semi-structured-grid conceptual view of a linear system.

2.1 SStruct Grids

Names

2.1.1 typedef struct hypre_SStructGrid_struct ***HYPRE_SStructGrid**
A grid object is constructed out of several structured "parts" and an optional unstructured "part". 23

2.1.2 typedef int **HYPRE_SStructVariable**
An enumerated type that supports cell centered, node centered, face centered, and edge centered variables. 23

2.1.3 int
HYPRE_SStructGridCreate (MPI_Comm comm, int ndim, int nparts, HYPRE_SStructGrid* grid)
Create an `ndim`*-dimensional grid object with* `nparts` *structured parts* 24

2.1.4 int
HYPRE_SStructGridDestroy (HYPRE_SStructGrid grid)
Destroy a grid object. 24

2.1.5 int
HYPRE_SStructGridSetExtents (HYPRE_SStructGrid grid, int part, int* ilower, int* iupper)
Set the extents for a box on a structured part of the grid 24

2.1.6 int

HYPRE_SStructGridSetVariables (HYPRE_SStructGrid grid, int part, int nvars, HYPRE_SStructVariable* vartypes)
Describe the variables that live on a structured part of the grid 25

2.1.7 int
HYPRE_SStructGridAddVariables (HYPRE_SStructGrid grid, int part, int* index, int nvars, HYPRE_SStructVariable* vartypes)
Describe additional variables that live at a particular index. 25

2.1.8 int
HYPRE_SStructGridSetFEMOrdering (HYPRE_SStructGrid grid, int part, int* ordering)
Set the ordering of variables in a finite element problem. 25

2.1.9 int
HYPRE_SStructGridSetNeighborPart (HYPRE_SStructGrid grid, int part, int* ilower, int* iupper, int nbor_part, int* nbor_ilower, int* nbor_iupper, int* index_map, int* index_dir)
Describe how regions just outside of a part relate to other parts. 26

2.1.10 int
HYPRE_SStructGridSetSharedPart (HYPRE_SStructGrid grid, int part, int* ilower, int* iupper, int* offset, int shared_part, int* shared_ilower, int* shared_iupper, int* shared_offset, int* index_map, int* index_dir)
Describe how regions inside a part are shared with regions in other parts. 26

2.1.11 int
HYPRE_SStructGridAddUnstructuredPart (HYPRE_SStructGrid grid, int ilower, int iupper)
Add an unstructured part to the grid. 27

2.1.12 int
HYPRE_SStructGridAssemble (HYPRE_SStructGrid grid)
Finalize the construction of the grid before using 28

2.1.13 int
HYPRE_SStructGridSetPeriodic (HYPRE_SStructGrid grid, int part, int* periodic)
Set the periodicity on a particular part. 28

2.1.14 int
HYPRE_SStructGridSetNumGhost (HYPRE_SStructGrid grid, int* num_ghost)
Setting ghost in the sgrids 28

2.1.1

typedef struct hypre_SStructGrid_struct ***HYPRE_SStructGrid**

A grid object is constructed out of several structured "parts" and an optional unstructured "part". Each structured part has its own abstract index space.

2.1.2

typedef int **HYPRE_SStructVariable**

An enumerated type that supports cell centered, node centered, face centered, and edge centered variables. Face centered variables are split into x-face, y-face, and z-face variables, and edge centered variables are split into x-edge, y-edge, and z-edge variables. The edge centered variable types are only used in 3D. In 2D, edge centered variables are handled by the face centered types.

Variables are referenced relative to an abstract (cell centered) index in the following way:

- cell centered variables are aligned with the index;
- node centered variables are aligned with the cell corner at relative index (1/2, 1/2, 1/2);
- x-face, y-face, and z-face centered variables are aligned with the faces at relative indexes (1/2, 0, 0), (0, 1/2, 0), and (0, 0, 1/2), respectively;
- x-edge, y-edge, and z-edge centered variables are aligned with the edges at relative indexes (0, 1/2, 1/2), (1/2, 0, 1/2), and (1/2, 1/2, 0), respectively.

The supported identifiers are:

- `HYPRE_SSTRUCT_VARIABLE_CELL`
- `HYPRE_SSTRUCT_VARIABLE_NODE`
- `HYPRE_SSTRUCT_VARIABLE_XFACE`
- `HYPRE_SSTRUCT_VARIABLE_YFACE`
- `HYPRE_SSTRUCT_VARIABLE_ZFACE`
- `HYPRE_SSTRUCT_VARIABLE_XEDGE`
- `HYPRE_SSTRUCT_VARIABLE_YEDGE`
- `HYPRE_SSTRUCT_VARIABLE_ZEDGE`

NOTE: Although variables are referenced relative to a unique abstract cell-centered index, some variables are associated with multiple grid cells. For example, node centered variables in 3D are associated with 8 cells (away from boundaries). Although grid cells are distributed uniquely to different processes, variables may be owned by multiple processes because they may be associated with multiple cells.

2.1.3

int
HYPRE_SStructGridCreate (MPI_Comm comm, int ndim, int nparts, HYPRE_SStructGrid* grid)

Create an `ndim`-dimensional grid object with `nparts` structured parts

2.1.4

int **HYPRE_SStructGridDestroy** (HYPRE_SStructGrid grid)

Destroy a grid object. An object should be explicitly destroyed using this destructor when the user's code no longer needs direct access to it. Once destroyed, the object must not be referenced again. Note that the object may not be deallocated at the completion of this call, since there may be internal package references to the object. The object will then be destroyed when all internal reference counts go to zero.

2.1.5

int
HYPRE_SStructGridSetExtents (HYPRE_SStructGrid grid, int part, int* ilower, int* iupper)

Set the extents for a box on a structured part of the grid

2.1.6

int
HYPRE_SStructGridSetVariables (HYPRE_SStructGrid grid, int part, int nvars, HYPRE_SStructVariable* vartypes)

Describe the variables that live on a structured part of the grid

2.1.7

int
HYPRE_SStructGridAddVariables (HYPRE_SStructGrid grid, int part, int* index, int nvars, HYPRE_SStructVariable* vartypes)

Describe additional variables that live at a particular index. These variables are appended to the array of variables set in HYPRE_SStructGridSetVariables, and are referenced as such.

NOTE: This routine is not yet supported.

2.1.8

int
HYPRE_SStructGridSetFEMOrdering (HYPRE_SStructGrid grid, int part, int* ordering)

Set the ordering of variables in a finite element problem. This overrides the default ordering described below.

Array `ordering` is composed of blocks of size (1 + `ndim`). Each block indicates a specific variable in the element and the ordering of the blocks defines the ordering of the variables. A block contains a variable number followed by an offset direction relative to the element's center. For example, a block containing (2, 1, -1, 0) means variable 2 on the edge located in the (1, -1, 0) direction from the center of the element. Note that here variable 2 must be of type `ZEDGE` for this to make sense. The `ordering` array must account for all variables in the element. This routine can only be called after HYPRE_SStructGridSetVariables.

The default ordering for element variables (var, i, j, k) varies fastest in index i, followed by j, then k, then var. For example, if var 0, var 1, and var 2 are declared to be XFACE, YFACE, and NODE variables, respectively, then the default ordering (in 2D) would first list the two XFACE variables, then the two YFACE variables, then the four NODE variables as follows:

(0,-1,0), (0,1,0), (1,0,-1), (1,0,1), (2,-1,-1), (2,1,-1), (2,-1,1), (2,1,1)

2.1.9

int
HYPRE_SStructGridSetNeighborPart (HYPRE_SStructGrid grid, int part, int* ilower, int* iupper, int nbor_part, int* nbor_ilower, int* nbor_iupper, int* index_map, int* index_dir)

Describe how regions just outside of a part relate to other parts. This is done a box at a time.

Parts `part` and `nbor_part` must be different, except in the case where only cell-centered data is used.

Indexes should increase from `ilower` to `iupper`. It is not necessary that indexes increase from `nbor_ilower` to `nbor_iupper`.

The `index_map` describes the mapping of indexes 0, 1, and 2 on part `part` to the corresponding indexes on part `nbor_part`. For example, triple (1, 2, 0) means that indexes 0, 1, and 2 on part `part` map to indexes 1, 2, and 0 on part `nbor_part`, respectively.

The `index_dir` describes the direction of the mapping in `index_map`. For example, triple (1, 1, -1) means that for indexes 0 and 1, increasing values map to increasing values on `nbor_part`, while for index 2, decreasing values map to increasing values.

NOTE: All parts related to each other via this routine must have an identical list of variables and variable types. For example, if part 0 has only two variables on it, a cell centered variable and a node centered variable, and we declare part 1 to be a neighbor of part 0, then part 1 must also have only two variables on it, and they must be of type cell and node. In addition, variables associated with FACEs or EDGEs must be grouped together and listed in X, Y, Z order. This is to enable the code to correctly associate variables on one part with variables on its neighbor part when a coordinate transformation is specified. For example, an XFACE variable on one part may correspond to a YFACE variable on a neighbor part under a particular tranformation, and the code determines this association by assuming that the variable lists are as noted here.

2.1.10

int
HYPRE_SStructGridSetSharedPart (HYPRE_SStructGrid grid, int part, int* ilower, int* iupper, int* offset, int shared_part, int* shared_ilower, int* shared_iupper, int* shared_offset, int* index_map, int* index_dir)

Describe how regions inside a part are shared with regions in other parts.

Parts `part` and `shared_part` must be different.

Indexes should increase from `ilower` to `iupper`. It is not necessary that indexes increase from `shared_ilower` to `shared_iupper`. This is to maintain consistency with the `SetNeighborPart` function, which is also able to describe shared regions but in a more limited fashion.

The `offset` is a triple (in 3D) used to indicate the dimensionality of the shared set of data and its position with respect to the box extents `ilower` and `iupper` on part `part`. The dimensionality is given by the number of 0's in the triple, and the position is given by plus or minus 1's. For example: (0, 0, 0) indicates sharing of all data in the given box; (1, 0, 0) indicates sharing of data on the faces in the (1, 0, 0) direction; (1, 0, -1) indicates sharing of data on the edges in the (1, 0, -1) direction; and (1, -1, 1) indicates sharing of data on the nodes in the (1, -1, 1) direction. To ensure the dimensionality, it is required that for every nonzero entry, the corresponding extents of the box are the same. For example, if `offset` is (0, 1, 0), then (2, 1, 3) and (10, 1, 15) are valid box extents, whereas (2, 1, 3) and (10, 7, 15) are invalid (because 1 and 7 are not the same).

The `shared_offset` is used in the same way as `offset`, but with respect to the box extents `shared_ilower` and `shared_iupper` on part `shared_part`.

The `index_map` describes the mapping of indexes 0, 1, and 2 on part `part` to the corresponding indexes on part `shared_part`. For example, triple (1, 2, 0) means that indexes 0, 1, and 2 on part `part` map to indexes 1, 2, and 0 on part `shared_part`, respectively.

The `index_dir` describes the direction of the mapping in `index_map`. For example, triple (1, 1, -1) means that for indexes 0 and 1, increasing values map to increasing values on `shared_part`, while for index 2, decreasing values map to increasing values.

NOTE: All parts related to each other via this routine must have an identical list of variables and variable types. For example, if part 0 has only two variables on it, a cell centered variable and a node centered variable, and we declare part 1 to have shared regions with part 0, then part 1 must also have only two variables on it, and they must be of type cell and node. In addition, variables associated with FACEs or EDGEs must be grouped together and listed in X, Y, Z order. This is to enable the code to correctly associate variables on one part with variables on a shared part when a coordinate transformation is specified. For example, an XFACE variable on one part may correspond to a YFACE variable on a shared part under a particular tranformation, and the code determines this association by assuming that the variable lists are as noted here.

2.1.11

int
HYPRE_SStructGridAddUnstructuredPart (HYPRE_SStructGrid grid, int ilower, int iupper)

Add an unstructured part to the grid. The variables in the unstructured part of the grid are referenced by a global rank between 0 and the total number of unstructured variables minus one. Each process owns some unique consecutive range of variables, defined by `ilower` and `iupper`.

NOTE: This is just a placeholder. This part of the interface is not finished.

2.1.12

int **HYPRE_SStructGridAssemble** (HYPRE_SStructGrid grid)

Finalize the construction of the grid before using

2.1.13

int
HYPRE_SStructGridSetPeriodic (HYPRE_SStructGrid grid, int part, int*
periodic)

Set the periodicity on a particular part.

The argument `periodic` is an `ndim`-dimensional integer array that contains the periodicity for each dimension. A zero value for a dimension means non-periodic, while a nonzero value means periodic and contains the actual period. For example, periodicity in the first and third dimensions for a 10x11x12 part is indicated by the array [10,0,12].

NOTE: Some of the solvers in hypre have power-of-two restrictions on the size of the periodic dimensions.

2.1.14

int
HYPRE_SStructGridSetNumGhost (HYPRE_SStructGrid grid, int*
num_ghost)

Setting ghost in the sgrids

2.2

SStruct Stencils

Names

2.2.1 typedef struct hypre_SStructStencil_struct ***HYPRE_SStructStencil**

The stencil object 29

2.2.2 int
HYPRE_SStructStencilCreate (int ndim, int size, HYPRE_SStructStencil* stencil)
Create a stencil object for the specified number of spatial dimensions and stencil entries 29

2.2.3 int
HYPRE_SStructStencilDestroy (HYPRE_SStructStencil stencil)
Destroy a stencil object 29

2.2.4 int
HYPRE_SStructStencilSetEntry (HYPRE_SStructStencil stencil, int entry, int* offset, int var)
Set a stencil entry 30

2.2.1

typedef struct hypre_SStructStencil_struct ***HYPRE_SStructStencil**

The stencil object

2.2.2

int
HYPRE_SStructStencilCreate (int ndim, int size, HYPRE_SStructStencil* stencil)

Create a stencil object for the specified number of spatial dimensions and stencil entries

2.2.3

int **HYPRE_SStructStencilDestroy** (HYPRE_SStructStencil stencil)

Destroy a stencil object

2.2.4

int
HYPRE_SStructStencilSetEntry (HYPRE_SStructStencil stencil, int entry, int* offset, int var)

Set a stencil entry

2.3

SStruct Graphs

Names

2.3.1 typedef struct hypre_SStructGraph_struct ***HYPRE_SStructGraph**
The graph object is used to describe the nonzero structure of a matrix 31

2.3.2 int
HYPRE_SStructGraphCreate (MPI_Comm comm, HYPRE_SStructGrid grid, HYPRE_SStructGraph* graph)
Create a graph object 31

2.3.3 int
HYPRE_SStructGraphDestroy (HYPRE_SStructGraph graph)
Destroy a graph object 31

2.3.4 int
HYPRE_SStructGraphSetDomainGrid (HYPRE_SStructGraph graph, HYPRE_SStructGrid domain_grid)
Set the domain grid 32

2.3.5 int
HYPRE_SStructGraphSetStencil (HYPRE_SStructGraph graph, int part, int var, HYPRE_SStructStencil stencil)
Set the stencil for a variable on a structured part of the grid 32

2.3.6 int
HYPRE_SStructGraphSetFEM (HYPRE_SStructGraph graph, int part)
Indicate that an FEM approach will be used to set matrix values on this part 32

2.3.7 int
HYPRE_SStructGraphSetFEMSparsity (HYPRE_SStructGraph graph, int part, int nsparse, int* sparsity)
Set the finite element stiffness matrix sparsity. 32

2.3.8 int

HYPRE_SStructGraphAddEntries (HYPRE_SStructGraph graph, int part,
int* index, int var, int to_part,
int* to_index, int to_var)
Add a non-stencil graph entry at a particular index. 33

2.3.9 int
HYPRE_SStructGraphAssemble (HYPRE_SStructGraph graph)
Finalize the construction of the graph before using 33

2.3.10 int
HYPRE_SStructGraphSetObjectType (HYPRE_SStructGraph graph,
int type)
Set the storage type of the associated matrix object. 33

2.3.1

typedef struct hypre_SStructGraph_struct ***HYPRE_SStructGraph**

The graph object is used to describe the nonzero structure of a matrix

2.3.2

int
HYPRE_SStructGraphCreate (MPI_Comm comm, HYPRE_SStructGrid grid, HYPRE_SStructGraph* graph)

Create a graph object

2.3.3

int **HYPRE_SStructGraphDestroy** (HYPRE_SStructGraph graph)

Destroy a graph object

2.3.4

int
HYPRE_SStructGraphSetDomainGrid (HYPRE_SStructGraph graph, HYPRE_SStructGrid domain_grid)

Set the domain grid

2.3.5

int
HYPRE_SStructGraphSetStencil (HYPRE_SStructGraph graph, int part, int var, HYPRE_SStructStencil stencil)

Set the stencil for a variable on a structured part of the grid

2.3.6

int **HYPRE_SStructGraphSetFEM** (HYPRE_SStructGraph graph, int part)

Indicate that an FEM approach will be used to set matrix values on this part

2.3.7

int
HYPRE_SStructGraphSetFEMSparsity (HYPRE_SStructGraph graph, int part, int nsparse, int* sparsity)

Set the finite element stiffness matrix sparsity. This overrides the default full sparsity pattern described below.

Array `sparsity` contains `nsparse` row/column tuples (I,J) that indicate the nonzeroes of the local stiffness matrix. The layout of the values passed into the routine HYPRE_SStructMatrixAddFEMValues is determined here.

The default sparsity is full (each variable is coupled to all others), and the values passed into the routine HYPRE_SStructMatrixAddFEMValues are assumed to be by rows (that is, column indices vary fastest).

2.3.8

int
HYPRE_SStructGraphAddEntries (HYPRE_SStructGraph graph, int part, int* index, int var, int to_part, int* to_index, int to_var)

Add a non-stencil graph entry at a particular index. This graph entry is appended to the existing graph entries, and is referenced as such.

NOTE: Users are required to set graph entries on all processes that own the associated variables. This means that some data will be multiply defined.

2.3.9

int **HYPRE_SStructGraphAssemble** (HYPRE_SStructGraph graph)

Finalize the construction of the graph before using

2.3.10

int
HYPRE_SStructGraphSetObjectType (HYPRE_SStructGraph graph, int type)

Set the storage type of the associated matrix object. It is used before AddEntries and Assemble to compute the right ranks in the graph.

NOTE: This routine is only necessary for implementation reasons, and will eventually be removed.

See Also: HYPRE_SStructMatrixSetObjectType (→2.4.16, *page 40*)

2.4

SStruct Matrices

Names

2.4.1 typedef struct hypre_SStructMatrix_struct ***HYPRE_SStructMatrix**
The matrix object 36

2.4.2 int
HYPRE_SStructMatrixCreate (MPI_Comm comm,
HYPRE_SStructGraph graph,
HYPRE_SStructMatrix* matrix)
Create a matrix object 36

2.4.3 int
HYPRE_SStructMatrixDestroy (HYPRE_SStructMatrix matrix)
Destroy a matrix object 36

2.4.4 int
HYPRE_SStructMatrixInitialize (HYPRE_SStructMatrix matrix)
Prepare a matrix object for setting coefficient values 36

2.4.5 int
HYPRE_SStructMatrixSetValues (HYPRE_SStructMatrix matrix, int part,
int* index, int var, int nentries,
int* entries, HYPRE_Complex* values)
Set matrix coefficients index by index. 37

2.4.6 int
HYPRE_SStructMatrixAddToValues (HYPRE_SStructMatrix matrix,
int part, int* index, int var,
int nentries, int* entries,
HYPRE_Complex* values)
Add to matrix coefficients index by index. 37

2.4.7 int
HYPRE_SStructMatrixAddFEMValues (HYPRE_SStructMatrix matrix,
int part, int* index,
HYPRE_Complex* values)
Add finite element stiffness matrix coefficients index by index. 37

2.4.8 int
HYPRE_SStructMatrixGetValues (HYPRE_SStructMatrix matrix, int part,
int* index, int var, int nentries,
int* entries, HYPRE_Complex* values)
Get matrix coefficients index by index. 38

2.4.9 int
HYPRE_SStructMatrixGetFEMValues (HYPRE_SStructMatrix matrix,
int part, int* index,
HYPRE_Complex* values)
Get finite element stiffness matrix coefficients index by index. 38

2.4.10 int

HYPRE_SStructMatrixSetBoxValues (HYPRE_SStructMatrix matrix, int part, int* ilower, int* iupper, int var, int nentries, HYPRE_Inte entiies HYRRE_oompeex vvalees)
Set matrix coefficients a box at a time. 38

2.4.11 int
HYPRE_SStructMatrixAddToBoxValues (HYPRE_SStructMatrix matrix, int part, int* ilower, int* iupper, int var, int nentries, int* entries, HYPRE_Complex* values)
Add to matrix coefficients a box at a time. 39

2.4.12 int
HYPRE_SStructMatrixGetBoxValues (HYPRE_SStructMatrix matrix, int part, int* ilower, int* iupper, int var, int nentries, int* entries, HYPRE_Complex* values)
Get matrix coefficients a box at a time. 39

2.4.13 int
HYPRE_SStructMatrixAssemble (HYPRE_SStructMatrix matrix)
Finalize the construction of the matrix before using 40

2.4.14 int
HYPRE_SStructMatrixSetSymmetric (HYPRE_SStructMatrix matrix, int part, int var, int to_var, int symmetric)
Define symmetry properties for the stencil entries in the matrix. 40

2.4.15 int
HYPRE_SStructMatrixSetNSSymmetric (HYPRE_SStructMatrix matrix, int symmetric)
Define symmetry properties for all non-stencil matrix entries 40

2.4.16 int
HYPRE_SStructMatrixSetObjectType (HYPRE_SStructMatrix matrix, int type)
Set the storage type of the matrix object to be constructed. 40

2.4.17 int
HYPRE_SStructMatrixGetObject (HYPRE_SStructMatrix matrix, void** object)
Get a reference to the constructed matrix object. 41

2.4.18 int
HYPRE_SStructMatrixPrint (const char* filename, HYPRE_SStructMatrix matrix, int all)
Print the matrix to file. 41

2.4.1

typedef struct hypre_SStructMatrix_struct ***HYPRE_SStructMatrix**

The matrix object

2.4.2

int
HYPRE_SStructMatrixCreate (MPI_Comm comm, HYPRE_SStructGraph graph, HYPRE_SStructMatrix* matrix)

Create a matrix object

2.4.3

int **HYPRE_SStructMatrixDestroy** (HYPRE_SStructMatrix matrix)

Destroy a matrix object

2.4.4

int **HYPRE_SStructMatrixInitialize** (HYPRE_SStructMatrix matrix)

Prepare a matrix object for setting coefficient values

2.4.5

int
HYPRE_SStructMatrixSetValues (HYPRE_SStructMatrix matrix, int part, int* index, int var, int nentries, int* entries, HYPRE_Complex* values)

Set matrix coefficients index by index. The `values` array is of length `nentries`.

NOTE: For better efficiency, use HYPRE_SStructMatrixSetBoxValues to set coefficients a box at a time.

NOTE: Users are required to set values on all processes that own the associated variables. This means that some data will be multiply defined.

NOTE: The entries in this routine must all be of the same type: either stencil or non-stencil, but not both. Also, if they are stencil entries, they must all represent couplings to the same variable type (there are no such restrictions for non-stencil entries).

2.4.6

int
HYPRE_SStructMatrixAddToValues (HYPRE_SStructMatrix matrix, int part, int* index, int var, int nentries, int* entries, HYPRE_Complex* values)

Add to matrix coefficients index by index. The `values` array is of length `nentries`.

NOTE: For better efficiency, use HYPRE_SStructMatrixAddToBoxValues to set coefficients a box at a time.

NOTE: Users are required to set values on all processes that own the associated variables. This means that some data will be multiply defined.

NOTE: The entries in this routine must all be of the same type: either stencil or non-stencil, but not both. Also, if they are stencil entries, they must all represent couplings to the same variable type.

2.4.7

int
HYPRE_SStructMatrixAddFEMValues (HYPRE_SStructMatrix matrix, int part, int* index, HYPRE_Complex* values)

Add finite element stiffness matrix coefficients index by index. The layout of the data in `values` is determined by the routines HYPRE_SStructGridSetFEMOrdering and HYPRE_SStructGraphSetFEMSparsity (→ 2.3.6, *page* 32).

2.4.8

int
HYPRE_SStructMatrixGetValues (HYPRE_SStructMatrix matrix, int part, int* index, int var, int nentries, int* entries, HYPRE_Complex* values)

Get matrix coefficients index by index. The `values` array is of length `nentries`.

NOTE: For better efficiency, use HYPRE_SStructMatrixGetBoxValues to get coefficients a box at a time.

NOTE: Users may get values on any process that owns the associated variables.

NOTE: The entries in this routine must all be of the same type: either stencil or non-stencil, but not both. Also, if they are stencil entries, they must all represent couplings to the same variable type (there are no such restrictions for non-stencil entries).

2.4.9

int
HYPRE_SStructMatrixGetFEMValues (HYPRE_SStructMatrix matrix, int part, int* index, HYPRE_Complex* values)

Get finite element stiffness matrix coefficients index by index. The layout of the data in `values` is determined by the routines HYPRE_SStructGridSetFEMOrdering and HYPRE_SStructGraphSetFEMSparsity (→ 2.3.6, *page* 32).

2.4.10

int
HYPRE_SStructMatrixSetBoxValues (HYPRE_SStructMatrix matrix, int part, int* ilower, int* iupper, int var, int nentries, HYPRE_Inte entiies HYRRE_oompeex vvalees)

Set matrix coefficients a box at a time. The data in `values` is ordered as follows:

```
m = 0;
for (k = ilower[2]; k <= iupper[2]; k++)
   for (j = ilower[1]; j <= iupper[1]; j++)
      for (i = ilower[0]; i <= iupper[0]; i++)
         for (entry = 0; entry < nentries; entry++)
```

```
{
   values[m] = ...;
   m++;
}
```

NOTE: Users are required to set values on all processes that own the associated variables. This means that some data will be multiply defined.

NOTE: The entries in this routine must all be of the same type: either stencil or non-stencil, but not both. Also, if they are stencil entries, they must all represent couplings to the same variable type (there are no such restrictions for non-stencil entries).

2.4.11

int
HYPRE_SStructMatrixAddToBoxValues (HYPRE_SStructMatrix matrix, int part, int* ilower, int* iupper, int var, int nentries, int* entries, HYPRE_Complex* values)

Add to matrix coefficients a box at a time. The data in `values` is ordered as in HYPRE_SStructMatrixSetBoxValues.

NOTE: Users are required to set values on all processes that own the associated variables. This means that some data will be multiply defined.

NOTE: The entries in this routine must all be of stencil type. Also, they must all represent couplings to the same variable type.

2.4.12

int
HYPRE_SStructMatrixGetBoxValues (HYPRE_SStructMatrix matrix, int part, int* ilower, int* iupper, int var, int nentries, int* entries, HYPRE_Complex* values)

Get matrix coefficients a box at a time. The data in `values` is ordered as in HYPRE_SStructMatrixSetBoxValues.

NOTE: Users may get values on any process that owns the associated variables.

NOTE: The entries in this routine must all be of stencil type. Also, they must all represent couplings to the same variable type.

2.4.13

int **HYPRE_SStructMatrixAssemble** (HYPRE_SStructMatrix matrix)

Finalize the construction of the matrix before using

2.4.14

int
HYPRE_SStructMatrixSetSymmetric (HYPRE_SStructMatrix matrix, int part, int var, int to_var, int symmetric)

Define symmetry properties for the stencil entries in the matrix. The boolean argument `symmetric` is applied to stencil entries on part `part` that couple variable `var` to variable `to_var`. A value of -1 may be used for `part`, `var`, or `to_var` to specify "all". For example, if `part` and `to_var` are set to -1, then the boolean is applied to stencil entries on all parts that couple variable `var` to all other variables.

By default, matrices are assumed to be nonsymmetric. Significant storage savings can be made if the matrix is symmetric.

2.4.15

int
HYPRE_SStructMatrixSetNSSymmetric (HYPRE_SStructMatrix matrix, int symmetric)

Define symmetry properties for all non-stencil matrix entries

2.4.16

int
HYPRE_SStructMatrixSetObjectType (HYPRE_SStructMatrix matrix, int type)

Set the storage type of the matrix object to be constructed. Currently, `type` can be either `HYPRE_SSTRUCT` (the default), `HYPRE_STRUCT`, or `HYPRE_PARCSR`.

See Also: HYPRE_SStructMatrixGetObject (→2.4.17, *page 41*)

2.4.17

int
HYPRE_SStructMatrixGetObject (HYPRE_SStructMatrix matrix, void** object)

Get a reference to the constructed matrix object.

See Also: HYPRE_SStructMatrixSetObjectType (→2.4.16, *page 40*)

2.4.18

int
HYPRE_SStructMatrixPrint (const char* filename, HYPRE_SStructMatrix matrix, int all)

Print the matrix to file. This is mainly for debugging purposes.

2.5

SStruct Vectors

Names

2.5.1 typedef struct hypre_SStructVector_struct ***HYPRE_SStructVector**
The vector object 43

2.5.2 int
HYPRE_SStructVectorCreate (MPI_Comm comm,
HYPRE_SStructGrid grid,
HYPRE_SStructVector* vector)
Create a vector object 43

2.5.3 int
HYPRE_SStructVectorDestroy (HYPRE_SStructVector vector)
Destroy a vector object 43

2.5.4 int

HYPRE_SStructVectorInitialize (HYPRE_SStructVector vector)
Prepare a vector object for setting coefficient values 44

2.5.5 int
HYPRE_SStructVectorSetValues (HYPRE_SStructVector vector, int part, int* index, int var, HYPRE_Complex* value)
Set vector coefficients index by index. 44

2.5.6 int
HYPRE_SStructVectorAddToValues (HYPRE_SStructVector vector, int part, int* index, int var, HYPRE_Complex* value)
Add to vector coefficients index by index. 44

2.5.7 int
HYPRE_SStructVectorAddFEMValues (HYPRE_SStructVector vector, int part, int* index, HYPRE_Complex* values)
Add finite element vector coefficients index by index. 44

2.5.8 int
HYPRE_SStructVectorGetValues (HYPRE_SStructVector vector, int part, int* index, int var, HYPRE_Complex* value)
Get vector coefficients index by index. 45

2.5.9 int
HYPRE_SStructVectorGetFEMValues (HYPRE_SStructVector vector, int part, int* index, HYPRE_Complex* values)
Get finite element vector coefficients index by index. 45

2.5.10 int
HYPRE_SStructVectorSetBoxValues (HYPRE_SStructVector vector, int part, int* ilower, int* iupper, int var, HYPRE_Complex* values)
Set vector coefficients a box at a time. 45

2.5.11 int
HYPRE_SStructVectorAddToBoxValues (HYPRE_SStructVector vector, int part, int* ilower, int* iupper, int var, HYPRE_Complex* values)
Add to vector coefficients a box at a time. 46

2.5.12 int
HYPRE_SStructVectorGetBoxValues (HYPRE_SStructVector vector, int part, int* ilower, int* iupper, int var, HYPRE_Complex* values)
Get vector coefficients a box at a time. 46

2.5.13 int
HYPRE_SStructVectorAssemble (HYPRE_SStructVector vector)
Finalize the construction of the vector before using 46

2.5.14 int

HYPRE_SStructVectorGather (HYPRE_SStructVector vector)
Gather vector data so that efficient `GetValues` *can be done.* 47

2.5.15 int
HYPRE_SStructVectorSetObjectType (HYPRE_SStructVector vector, int type)
Set the storage type of the vector object to be constructed. 47

2.5.16 int
HYPRE_SStructVectorGetObject (HYPRE_SStructVector vector, void** object)
Get a reference to the constructed vector object. 47

2.5.17 int
HYPRE_SStructVectorPrint (const char* filename, HYPRE_SStructVector vector, int all)
Print the vector to file. 47

2.5.1

typedef struct hypre_SStructVector_struct ***HYPRE_SStructVector**

The vector object

2.5.2

int
HYPRE_SStructVectorCreate (MPI_Comm comm, HYPRE_SStructGrid grid, HYPRE_SStructVector* vector)

Create a vector object

2.5.3

int **HYPRE_SStructVectorDestroy** (HYPRE_SStructVector vector)

Destroy a vector object

2.5.4

int **HYPRE_SStructVectorInitialize** (HYPRE_SStructVector vector)

Prepare a vector object for setting coefficient values

2.5.5

int
HYPRE_SStructVectorSetValues (HYPRE_SStructVector vector, int part, int* index, int var, HYPRE_Complex* value)

Set vector coefficients index by index.

NOTE: For better efficiency, use HYPRE_SStructVectorSetBoxValues to set coefficients a box at a time.

NOTE: Users are required to set values on all processes that own the associated variables. This means that some data will be multiply defined.

2.5.6

int
HYPRE_SStructVectorAddToValues (HYPRE_SStructVector vector, int part, int* index, int var, HYPRE_Complex* value)

Add to vector coefficients index by index.

NOTE: For better efficiency, use HYPRE_SStructVectorAddToBoxValues to set coefficients a box at a time.

NOTE: Users are required to set values on all processes that own the associated variables. This means that some data will be multiply defined.

2.5.7

int
HYPRE_SStructVectorAddFEMValues (HYPRE_SStructVector vector, int part, int* index, HYPRE_Complex* values)

Add finite element vector coefficients index by index. The layout of the data in `values` is determined by the routine HYPRE_SStructGridSetFEMOrdering.

2.5.8

int
HYPRE_SStructVectorGetValues (HYPRE_SStructVector vector, int part, int* index, int var, HYPRE_Complex* value)

Get vector coefficients index by index. Users must first call the routine HYPRE_SStructVectorGather to ensure that data owned by multiple processes is correct.

NOTE: For better efficiency, use HYPRE_SStructVectorGetBoxValues to get coefficients a box at a time.

NOTE: Users may only get values on processes that own the associated variables.

2.5.9

int
HYPRE_SStructVectorGetFEMValues (HYPRE_SStructVector vector, int part, int* index, HYPRE_Complex* values)

Get finite element vector coefficients index by index. The layout of the data in `values` is determined by the routine HYPRE_SStructGridSetFEMOrdering. Users must first call the routine HYPRE_SStructVectorGather to ensure that data owned by multiple processes is correct.

2.5.10

int
HYPRE_SStructVectorSetBoxValues (HYPRE_SStructVector vector, int part, int* ilower, int* iupper, int var, HYPRE_Complex* values)

Set vector coefficients a box at a time. The data in `values` is ordered as follows:

```
m = 0;
for (k = ilower[2]; k <= iupper[2]; k++)
   for (j = ilower[1]; j <= iupper[1]; j++)
      for (i = ilower[0]; i <= iupper[0]; i++)
```

```
{
   values[m] = ...;
   m++;
}
```

NOTE: Users are required to set values on all processes that own the associated variables. This means that some data will be multiply defined.

2.5.11

int
HYPRE_SStructVectorAddToBoxValues (HYPRE_SStructVector vector, int part, int* ilower, int* iupper, int var, HYPRE_Complex* values)

Add to vector coefficients a box at a time. The data in `values` is ordered as in HYPRE_SStructVectorSetBoxValues.

NOTE: Users are required to set values on all processes that own the associated variables. This means that some data will be multiply defined.

2.5.12

int
HYPRE_SStructVectorGetBoxValues (HYPRE_SStructVector vector, int part, int* ilower, int* iupper, int var, HYPRE_Complex* values)

Get vector coefficients a box at a time. The data in `values` is ordered as in HYPRE_SStructVectorSetBoxValues. Users must first call the routine HYPRE_SStructVectorGather to ensure that data owned by multiple processes is correct.

NOTE: Users may only get values on processes that own the associated variables.

2.5.13

int **HYPRE_SStructVectorAssemble** (HYPRE_SStructVector vector)

Finalize the construction of the vector before using

2.5.14

int **HYPRE_SStructVectorGather** (HYPRE_SStructVector vector)

Gather vector data so that efficient `GetValues` can be done. This routine must be called prior to calling `GetValues` to ensure that correct and consistent values are returned, especially for non cell-centered data that is shared between more than one processor.

2.5.15

int
HYPRE_SStructVectorSetObjectType (HYPRE_SStructVector vector, int type)

Set the storage type of the vector object to be constructed. Currently, `type` can be either `HYPRE_SSTRUCT` (the default), `HYPRE_STRUCT`, or `HYPRE_PARCSR`.

See Also: HYPRE_SStructVectorGetObject (→2.5.16, *page 47*)

2.5.16

int
HYPRE_SStructVectorGetObject (HYPRE_SStructVector vector, void** object)

Get a reference to the constructed vector object.

See Also: HYPRE_SStructVectorSetObjectType (→2.5.15, *page 47*)

2.5.17

int
HYPRE_SStructVectorPrint (const char* filename, HYPRE_SStructVector vector, int all)

Print the vector to file. This is mainly for debugging purposes.

3

IJ System Interface

Names

3.1 **IJ Matrices**
.... 48

3.2 **IJ Vectors**
.... 56

This interface represents a linear-algebraic conceptual view of a linear system. The 'I' and 'J' in the name are meant to be mnemonic for the traditional matrix notation A(I,J).

3.1

IJ Matrices

Names

3.1.1 typedef struct hypre_IJMatrix_struct ***HYPRE_IJMatrix**
The matrix object 50

3.1.2 int
HYPRE_IJMatrixCreate (MPI_Comm comm, int ilower, int iupper,
int jlower, int jupper, HYPRE_IJMatrix* matrix)
Create a matrix object. 50

3.1.3 int
HYPRE_IJMatrixDestroy (HYPRE_IJMatrix matrix)
Destroy a matrix object. 51

3.1.4 int
HYPRE_IJMatrixInitialize (HYPRE_IJMatrix matrix)
Prepare a matrix object for setting coefficient values. 51

3.1.5 int
HYPRE_IJMatrixSetValues (HYPRE_IJMatrix matrix, int nrows, int* ncols,
const int* rows, const int* cols,
const HYPRE_Complex* values)
Sets values for `nrows` *rows or partial rows of the matrix.* 51

3.1.6 int
HYPRE_IJMatrixAddToValues (HYPRE_IJMatrix matrix, int nrows,
int* ncols, const int* rows, const int* cols,
const HYPRE_Complex* values)
Adds to values for `nrows` *rows or partial rows of the matrix.* 52

3.1.7 int

HYPRE_IJMatrixAssemble (HYPRE_IJMatrix matrix)
Finalize the construction of the matrix before using 52

3.1.8 int
HYPRE_IJMatrixGetRowCounts (HYPRE_IJMatrix matrix, int nrows, int* rows, int* ncols)
Gets number of nonzeros elements for `nrows` *rows specified in* `rows` *and returns them in* `ncols`*, which needs to be allocated by the user* 52

3.1.9 int
HYPRE_IJMatrixGetValues (HYPRE_IJMatrix matrix, int nrows, int* ncols, int* rows, int* cols, HYPRE_Complex* values)
Gets values for `nrows` *rows or partial rows of the matrix.* 52

3.1.10 int
HYPRE_IJMatrixSetObjectType (HYPRE_IJMatrix matrix, int type)
Set the storage type of the matrix object to be constructed. 53

3.1.11 int
HYPRE_IJMatrixGetObjectType (HYPRE_IJMatrix matrix, int* type)
Get the storage type of the constructed matrix object 53

3.1.12 int
HYPRE_IJMatrixGetLocalRange (HYPRE_IJMatrix matrix, int* ilower, int* iupper, int* jlower, int* jupper)
Gets range of rows owned by this processor and range of column partitioning for this processor 53

3.1.13 int
HYPRE_IJMatrixGetObject (HYPRE_IJMatrix matrix, void** object)
Get a reference to the constructed matrix object. 53

3.1.14 int
HYPRE_IJMatrixSetRowSizes (HYPRE_IJMatrix matrix, const int* sizes)
(Optional) Set the max number of nonzeros to expect in each row. 54

3.1.15 int
HYPRE_IJMatrixSetDiagOffdSizes (HYPRE_IJMatrix matrix, const int* diag_sizes, const int* offdiag_sizes)
(Optional) Sets the exact number of nonzeros in each row of the diagonal and off-diagonal blocks. 54

3.1.16 int
HYPRE_IJMatrixSetMaxOffProcElmts (HYPRE_IJMatrix matrix, int max_off_proc_elmts)
(Optional) Sets the maximum number of elements that are expected to be set (or added) on other processors from this processor This routine can significantly improve the efficiency of matrix construction, and should always be utilized if possible. 54

3.1.17 int
HYPRE_IJMatrixSetPrintLevel (HYPRE_IJMatrix matrix, int print_level)
(Optional) Sets the print level, if the user wants to print error messages. 55

3.1.18 int

HYPRE_IJMatrixSetOMPFlag (HYPRE_IJMatrix matrix, int omp_flag)
(Optional) if set, will use a threaded version of HYPRE_IJMatrixSetValues and HYPRE_IJMatrixAddToValues. 55

3.1.19 int
HYPRE_IJMatrixRead (const char* filename, MPI_Comm comm, int type, HYPRE_IJMatrix* matrix)
Read the matrix from file. 55

3.1.20 int
HYPRE_IJMatrixPrint (HYPRE_IJMatrix matrix, const char* filename)
Print the matrix to file. 55

3.1.1

typedef struct hypre_IJMatrix_struct ***HYPRE_IJMatrix**

The matrix object

3.1.2

int
HYPRE_IJMatrixCreate (MPI_Comm comm, int ilower, int iupper, int jlower, int jupper, HYPRE_IJMatrix* matrix)

Create a matrix object. Each process owns some unique consecutive range of rows, indicated by the global row indices `ilower` and `iupper`. The row data is required to be such that the value of `ilower` on any process p be exactly one more than the value of `iupper` on process $p-1$. Note that the first row of the global matrix may start with any integer value. In particular, one may use zero- or one-based indexing.

For square matrices, `jlower` and `jupper` typically should match `ilower` and `iupper`, respectively. For rectangular matrices, `jlower` and `jupper` should define a partitioning of the columns. This partitioning must be used for any vector v that will be used in matrix-vector products with the rectangular matrix. The matrix data structure may use `jlower` and `jupper` to store the diagonal blocks (rectangular in general) of the matrix separately from the rest of the matrix.

Collective.

3.1.3

int **HYPRE_IJMatrixDestroy** (HYPRE_IJMatrix matrix)

Destroy a matrix object. An object should be explicitly destroyed using this destructor when the user's code no longer needs direct access to it. Once destroyed, the object must not be referenced again. Note that the object may not be deallocated at the completion of this call, since there may be internal package references to the object. The object will then be destroyed when all internal reference counts go to zero.

3.1.4

int **HYPRE_IJMatrixInitialize** (HYPRE_IJMatrix matrix)

Prepare a matrix object for setting coefficient values. This routine will also re-initialize an already assembled matrix, allowing users to modify coefficient values.

3.1.5

int
HYPRE_IJMatrixSetValues (HYPRE_IJMatrix matrix, int nrows, int* ncols, const int* rows, const int* cols, const HYPRE_Complex* values)

Sets values for `nrows` rows or partial rows of the matrix. The arrays `ncols` and `rows` are of dimension `nrows` and contain the number of columns in each row and the row indices, respectively. The array `cols` contains the column indices for each of the `rows`, and is ordered by rows. The data in the `values` array corresponds directly to the column entries in `cols`. Erases any previous values at the specified locations and replaces them with new ones, or, if there was no value there before, inserts a new one if set locally. Note that it is not possible to set values on other processors. If one tries to set a value from proc i on proc j, proc i will erase all previous occurrences of this value in its stack (including values generated with AddToValues), and treat it like a zero value. The actual value needs to be set on proc j.

Note that a threaded version (threaded over the number of rows) will be called if HYPRE_IJMatrixSetOMPFlag is set to a value != 0. This requires that rows[i] != rows[j] for i!= j and is only efficient if a large number of rows is set in one call to HYPRE_IJMatrixSetValues.

Not collective.

3.1.6

int
HYPRE_IJMatrixAddToValues (HYPRE_IJMatrix matrix, int nrows, int* ncols, const int* rows, const int* cols, const HYPRE_Complex* values)

Adds to values for `nrows` rows or partial rows of the matrix. Usage details are analogous to HYPRE_IJMatrixSetValues. Adds to any previous values at the specified locations, or, if there was no value there before, inserts a new one. AddToValues can be used to add to values on other processors.

Note that a threaded version (threaded over the number of rows) will be called if HYPRE_IJMatrixSetOMPFlag is set to a value != 0. This requires that rows[i] != rows[j] for i!= j and is only efficient if a large number of rows is added in one call to HYPRE_IJMatrixAddToValues.

Not collective.

3.1.7

int **HYPRE_IJMatrixAssemble** (HYPRE_IJMatrix matrix)

Finalize the construction of the matrix before using

3.1.8

int
HYPRE_IJMatrixGetRowCounts (HYPRE_IJMatrix matrix, int nrows, int* rows, int* ncols)

Gets number of nonzeros elements for `nrows` rows specified in `rows` and returns them in `ncols`, which needs to be allocated by the user

3.1.9

int
HYPRE_IJMatrixGetValues (HYPRE_IJMatrix matrix, int nrows, int* ncols, int* rows, int* cols, HYPRE_Complex* values)

Gets values for `nrows` rows or partial rows of the matrix. Usage details are analogous to HYPRE_IJMatrixSetValues.

3.1.10

int **HYPRE_IJMatrixSetObjectType** (HYPRE_IJMatrix matrix, int type)

Set the storage type of the matrix object to be constructed. Currently, `type` can only be `HYPRE_PARCSR`.

Not collective, but must be the same on all processes.

See Also: HYPRE_IJMatrixGetObject (→3.1.13, *page 53*)

3.1.11

int **HYPRE_IJMatrixGetObjectType** (HYPRE_IJMatrix matrix, int* type)

Get the storage type of the constructed matrix object

3.1.12

int
HYPRE_IJMatrixGetLocalRange (HYPRE_IJMatrix matrix, int* ilower, int* iupper, int* jlower, int* jupper)

Gets range of rows owned by this processor and range of column partitioning for this processor

3.1.13

int **HYPRE_IJMatrixGetObject** (HYPRE_IJMatrix matrix, void** object)

Get a reference to the constructed matrix object.

See Also: HYPRE_IJMatrixSetObjectType (→3.1.10, *page 53*)

3.1.14

int **HYPRE_IJMatrixSetRowSizes** (HYPRE_IJMatrix matrix, const int* sizes)

(Optional) Set the max number of nonzeros to expect in each row. The array `sizes` contains estimated sizes for each row on this process. This call can significantly improve the efficiency of matrix construction, and should always be utilized if possible.

Not collective.

3.1.15

int
HYPRE_IJMatrixSetDiagOffdSizes (HYPRE_IJMatrix matrix, const int* diag_sizes, const int* offdiag_sizes)

(Optional) Sets the exact number of nonzeros in each row of the diagonal and off-diagonal blocks. The diagonal block is the submatrix whose column numbers correspond to rows owned by this process, and the off-diagonal block is everything else. The arrays `diag_sizes` and `offdiag_sizes` contain estimated sizes for each row of the diagonal and off-diagonal blocks, respectively. This routine can significantly improve the efficiency of matrix construction, and should always be utilized if possible.

Not collective.

3.1.16

int
HYPRE_IJMatrixSetMaxOffProcElmts (HYPRE_IJMatrix matrix, int max_off_proc_elmts)

(Optional) Sets the maximum number of elements that are expected to be set (or added) on other processors from this processor This routine can significantly improve the efficiency of matrix construction, and should always be utilized if possible.

Not collective.

3.1.17

int **HYPRE_IJMatrixSetPrintLevel** (HYPRE_IJMatrix matrix, int print_level)

(Optional) Sets the print level, if the user wants to print error messages. The default is 0, i.e. no error messages are printed.

3.1.18

int **HYPRE_IJMatrixSetOMPFlag** (HYPRE_IJMatrix matrix, int omp_flag)

(Optional) if set, will use a threaded version of HYPRE_IJMatrixSetValues and HYPRE_IJMatrixAddToValues. This is only useful if a large number of rows is set or added to at once.

NOTE that the values in the rows array of HYPRE_IJMatrixSetValues or HYPRE_IJMatrixAddToValues must be different from each other !!!

This option is VERY inefficient if only a small number of rows is set or added at once and/or if reallocation of storage is required and/or if values are added to off processor values.

3.1.19

int
HYPRE_IJMatrixRead (const char* filename, MPI_Comm comm, int type, HYPRE_IJMatrix* matrix)

Read the matrix from file. This is mainly for debugging purposes.

3.1.20

int **HYPRE_IJMatrixPrint** (HYPRE_IJMatrix matrix, const char* filename)

Print the matrix to file. This is mainly for debugging purposes.

3.2

IJ Vectors

Names

3.2.1 typedef struct hypre_IJVector_struct ***HYPRE_IJVector**
The vector object 57

3.2.2 int
HYPRE_IJVectorCreate (MPI_Comm comm, int jlower, int jupper, HYPRE_IJVector* vector)
Create a vector object. 57

3.2.3 int
HYPRE_IJVectorDestroy (HYPRE_IJVector vector)
Destroy a vector object. 58

3.2.4 int
HYPRE_IJVectorInitialize (HYPRE_IJVector vector)
Prepare a vector object for setting coefficient values. 58

3.2.5 int
HYPRE_IJVectorSetMaxOffProcElmts (HYPRE_IJVector vector, int max_off_proc_elmts)
(Optional) Sets the maximum number of elements that are expected to be set (or added) on other processors from this processor This routine can significantly improve the efficiency of matrix construction, and should always be utilized if possible. 58

3.2.6 int
HYPRE_IJVectorSetValues (HYPRE_IJVector vector, int nvalues, const int* indices, const HYPRE_Complex* values)
Sets values in vector. 59

3.2.7 int
HYPRE_IJVectorAddToValues (HYPRE_IJVector vector, int nvalues, const int* indices, const HYPRE_Complex* values)
Adds to values in vector. 59

3.2.8 int
HYPRE_IJVectorAssemble (HYPRE_IJVector vector)
Finalize the construction of the vector before using 59

3.2.9 int
HYPRE_IJVectorGetValues (HYPRE_IJVector vector, int nvalues, const int* indices, HYPRE_Complex* values)
Gets values in vector. 59

3.2.10 int

HYPRE_IJVectorSetObjectType (HYPRE_IJVector vector, int type)
Set the storage type of the vector object to be constructed. 60

3.2.11 int
HYPRE_IJVectorGetObjectType (HYPRE_IJVector vector, int* type)
Get the storage type of the constructed vector object 60

3.2.12 int
HYPRE_IJVectorGetLocalRange (HYPRE_IJVector vector, int* jlower, int* jupper)
Returns range of the part of the vector owned by this processor 60

3.2.13 int
HYPRE_IJVectorGetObject (HYPRE_IJVector vector, void** object)
Get a reference to the constructed vector object. 60

3.2.14 int
HYPRE_IJVectorSetPrintLevel (HYPRE_IJVector vector, int print_level)
(Optional) Sets the print level, if the user wants to print error messages. 61

3.2.15 int
HYPRE_IJVectorRead (const char* filename, MPI_Comm comm, int type, HYPRE_IJVector* vector)
Read the vector from file. 61

3.2.16 int
HYPRE_IJVectorPrint (HYPRE_IJVector vector, const char* filename)
Print the vector to file. 61

3.2.1

typedef struct hypre_IJVector_struct ***HYPRE_IJVector**

The vector object

3.2.2

int
HYPRE_IJVectorCreate (MPI_Comm comm, int jlower, int jupper, HYPRE_IJVector* vector)

Create a vector object. Each process owns some unique consecutive range of vector unknowns, indicated by the global indices `jlower` and `jupper`. The data is required to be such that the value of `jlower` on any process p be exactly one more than the value of `jupper` on process $p-1$. Note that the first index of the global vector may start with any integer value. In particular, one may use zero- or one-based indexing.

Collective.

3.2.3

int **HYPRE_IJVectorDestroy** (HYPRE_IJVector vector)

Destroy a vector object. An object should be explicitly destroyed using this destructor when the user's code no longer needs direct access to it. Once destroyed, the object must not be referenced again. Note that the object may not be deallocated at the completion of this call, since there may be internal package references to the object. The object will then be destroyed when all internal reference counts go to zero.

3.2.4

int **HYPRE_IJVectorInitialize** (HYPRE_IJVector vector)

Prepare a vector object for setting coefficient values. This routine will also re-initialize an already assembled vector, allowing users to modify coefficient values.

3.2.5

int
HYPRE_IJVectorSetMaxOffProcElmts (HYPRE_IJVector vector, int max_off_proc_elmts)

(Optional) Sets the maximum number of elements that are expected to be set (or added) on other processors from this processor This routine can significantly improve the efficiency of matrix construction, and should always be utilized if possible.

Not collective.

3.2.6

int
HYPRE_IJVectorSetValues (HYPRE_IJVector vector, int nvalues, const int* indices, const HYPRE_Complex* values)

Sets values in vector. The arrays `values` and `indices` are of dimension `nvalues` and contain the vector values to be set and the corresponding global vector indices, respectively. Erases any previous values at the specified locations and replaces them with new ones. Note that it is not possible to set values on other processors. If one tries to set a value from proc i on proc j, proc i will erase all previous occurrences of this value in its stack (including values generated with AddToValues), and treat it like a zero value. The actual value needs to be set on proc j.

Not collective.

3.2.7

int
HYPRE_IJVectorAddToValues (HYPRE_IJVector vector, int nvalues, const int* indices, const HYPRE_Complex* values)

Adds to values in vector. Usage details are analogous to HYPRE_IJVectorSetValues. Adds to any previous values at the specified locations, or, if there was no value there before, inserts a new one. AddToValues can be used to add to values on other processors.

Not collective.

3.2.8

int **HYPRE_IJVectorAssemble** (HYPRE_IJVector vector)

Finalize the construction of the vector before using

3.2.9

int
HYPRE_IJVectorGetValues (HYPRE_IJVector vector, int nvalues, const int* indices, HYPRE_Complex* values)

Gets values in vector. Usage details are analogous to HYPRE_IJVectorSetValues.

Not collective.

3.2.10

int **HYPRE_IJVectorSetObjectType** (HYPRE_IJVector vector, int type)

Set the storage type of the vector object to be constructed. Currently, `type` can only be `HYPRE_PARCSR`.

Not collective, but must be the same on all processes.

See Also: HYPRE_IJVectorGetObject (→3.2.13, *page 60*)

3.2.11

int **HYPRE_IJVectorGetObjectType** (HYPRE_IJVector vector, int* type)

Get the storage type of the constructed vector object

3.2.12

int
HYPRE_IJVectorGetLocalRange (HYPRE_IJVector vector, int* jlower, int* jupper)

Returns range of the part of the vector owned by this processor

3.2.13

int **HYPRE_IJVectorGetObject** (HYPRE_IJVector vector, void** object)

Get a reference to the constructed vector object.

See Also: HYPRE_IJVectorSetObjectType (→3.2.10, *page 60*)

3.2.14

int **HYPRE_IJVectorSetPrintLevel** (HYPRE_IJVector vector, int print_level)

(Optional) Sets the print level, if the user wants to print error messages. The default is 0, i.e. no error messages are printed.

3.2.15

int
HYPRE_IJVectorRead (const char* filename, MPI_Comm comm, int type, HYPRE_IJVector* vector)

Read the vector from file. This is mainly for debugging purposes.

3.2.16

int **HYPRE_IJVectorPrint** (HYPRE_IJVector vector, const char* filename)

Print the vector to file. This is mainly for debugging purposes.

4 Struct Solvers

Names

4.1 **Struct Solvers** 62

4.2 **Struct Jacobi Solver** 63

4.3 **Struct PFMG Solver** 66

4.4 **Struct SMG Solver** 73

4.5 **Struct CycRed Solver** 78

4.6 **Struct PCG Solver** 81

4.7 **Struct GMRES Solver** 82

4.8 **Struct FlexGMRES Solver** 83

4.9 **Struct LGMRES Solver** 84

4.10 **Struct BiCGSTAB Solver** 85

4.11 **Struct Hybrid Solver** 85

4.12 **Struct LOBPCG Eigensolver** 92

These solvers use matrix/vector storage schemes that are tailored to structured grid problems.

4.1 Struct Solvers

Names

4.1.1 typedef struct hypre_StructSolver_struct ***HYPRE_StructSolver**
The solver object 63

4.1.1

typedef struct hypre_StructSolver_struct ***HYPRE_StructSolver**

The solver object

4.2

Struct Jacobi Solver

Names

4.2.1 int
HYPRE_StructJacobiCreate (MPI_Comm comm,
HYPRE_StructSolver* solver)
Create a solver object 64

4.2.2 int
HYPRE_StructJacobiDestroy (HYPRE_StructSolver solver)
Destroy a solver object. 64

4.2.3 int
HYPRE_StructJacobiSetup (HYPRE_StructSolver solver,
HYPRE_StructMatrix A,
HYPRE_StructVector b,
HYPRE_StructVector x)
Prepare to solve the system. 64

4.2.4 int
HYPRE_StructJacobiSolve (HYPRE_StructSolver solver,
HYPRE_StructMatrix A,
HYPRE_StructVector b,
HYPRE_StructVector x)
Solve the system 65

4.2.5 int
HYPRE_StructJacobiSetTol (HYPRE_StructSolver solver, HYPRE_Real tol)
(Optional) Set the convergence tolerance 65

4.2.6 int
HYPRE_StructJacobiSetMaxIter (HYPRE_StructSolver solver, int max_iter)
(Optional) Set maximum number of iterations 65

4.2.7 int
HYPRE_StructJacobiSetZeroGuess (HYPRE_StructSolver solver)
(Optional) Use a zero initial guess. 65

4.2.8 int

HYPRE_StructJacobiSetNonZeroGuess (HYPRE_StructSolver solver)
(Optional) Use a nonzero initial guess. 66

4.2.9 int
HYPRE_StructJacobiGetNumIterations (HYPRE_StructSolver solver, int* num_iterations)
Return the number of iterations taken 66

4.2.10 int
HYPRE_StructJacobiGetFinalRelativeResidualNorm (HYPRE_StructSolver solver, HYPRE_Real* norm)
Return the norm of the final relative residual 66

4.2.1

int
HYPRE_StructJacobiCreate (MPI_Comm comm, HYPRE_StructSolver* solver)

Create a solver object

4.2.2

int **HYPRE_StructJacobiDestroy** (HYPRE_StructSolver solver)

Destroy a solver object. An object should be explicitly destroyed using this destructor when the user's code no longer needs direct access to it. Once destroyed, the object must not be referenced again. Note that the object may not be deallocated at the completion of this call, since there may be internal package references to the object. The object will then be destroyed when all internal reference counts go to zero.

4.2.3

int
HYPRE_StructJacobiSetup (HYPRE_StructSolver solver, HYPRE_StructMatrix A, HYPRE_StructVector b, HYPRE_StructVector x)

Prepare to solve the system. The coefficient data in `b` and `x` is ignored here, but information about the layout of the data may be used.

4.2.4

int
HYPRE_StructJacobiSolve (HYPRE_StructSolver solver, HYPRE_StructMatrix A, HYPRE_StructVector b, HYPRE_StructVector x)

Solve the system

4.2.5

int
HYPRE_StructJacobiSetTol (HYPRE_StructSolver solver, HYPRE_Real tol)

(Optional) Set the convergence tolerance

4.2.6

int
HYPRE_StructJacobiSetMaxIter (HYPRE_StructSolver solver, int max_iter)

(Optional) Set maximum number of iterations

4.2.7

int **HYPRE_StructJacobiSetZeroGuess** (HYPRE_StructSolver solver)

(Optional) Use a zero initial guess. This allows the solver to cut corners in the case where a zero initial guess is needed (e.g., for preconditioning) to reduce compuational cost.

4.2.8

int **HYPRE_StructJacobiSetNonZeroGuess** (HYPRE_StructSolver solver)

(Optional) Use a nonzero initial guess. This is the default behavior, but this routine allows the user to switch back after using `SetZeroGuess`.

4.2.9

int
HYPRE_StructJacobiGetNumIterations (HYPRE_StructSolver solver, int*
num_iterations)

Return the number of iterations taken

4.2.10

int
HYPRE_StructJacobiGetFinalRelativeResidualNorm (HYPRE_StructSolver
solver, HYPRE_Real* norm)

Return the norm of the final relative residual

4.3 Struct PFMG Solver

Names

4.3.1 int
HYPRE_StructPFMGCreate (MPI_Comm comm,
HYPRE_StructSolver* solver)
Create a solver object 68

4.3.2 int
HYPRE_StructPFMGDestroy (HYPRE_StructSolver solver)
Destroy a solver object 68

4.3.3 int

HYPRE_StructPFMGSetup (HYPRE_StructSolver solver,
HYPRE_StructMatrix A,
HYPRE_StructVector b,
HYPRE_StructVector x)
Prepare to solve the system. 69

4.3.4 int
HYPRE_StructPFMGSolve (HYPRE_StructSolver solver,
HYPRE_StructMatrix A,
HYPRE_StructVector b,
HYPRE_StructVector x)
Solve the system 69

4.3.5 int
HYPRE_StructPFMGSetTol (HYPRE_StructSolver solver, HYPRE_Real tol)
(Optional) Set the convergence tolerance 69

4.3.6 int
HYPRE_StructPFMGSetMaxIter (HYPRE_StructSolver solver,
int max_iter)
(Optional) Set maximum number of iterations 69

4.3.7 int
HYPRE_StructPFMGSetMaxLevels (HYPRE_StructSolver solver,
int max_levels)
(Optional) Set maximum number of multigrid grid levels 70

4.3.8 int
HYPRE_StructPFMGSetRelChange (HYPRE_StructSolver solver,
int rel_change)
(Optional) Additionally require that the relative difference in successive iterates be small 70

4.3.9 int
HYPRE_StructPFMGSetZeroGuess (HYPRE_StructSolver solver)
(Optional) Use a zero initial guess. 70

4.3.10 int
HYPRE_StructPFMGSetNonZeroGuess (HYPRE_StructSolver solver)
(Optional) Use a nonzero initial guess. 70

4.3.11 int
HYPRE_StructPFMGSetRelaxType (HYPRE_StructSolver solver,
int relax_type)
(Optional) Set relaxation type. 71

4.3.12 int
HYPRE_StructPFMGSetRAPType (HYPRE_StructSolver solver,
int rap_type)
(Optional) Set type of coarse-grid operator to use. 71

4.3.13 int
HYPRE_StructPFMGSetNumPreRelax (HYPRE_StructSolver solver,
int num_pre_relax)
(Optional) Set number of relaxation sweeps before coarse-grid correction . 71

4.3.14 int

HYPRE_StructPFMGSetNumPostRelax (HYPRE_StructSolver solver, int num_post_relax)
(Optional) Set number of relaxation sweeps after coarse-grid correction .. 72

4.3.15 int
HYPRE_StructPFMGSetSkipRelax (HYPRE_StructSolver solver, int skip_relax)
(Optional) Skip relaxation on certain grids for isotropic problems. 72

4.3.16 int
HYPRE_StructPFMGSetLogging (HYPRE_StructSolver solver, int logging)
(Optional) Set the amount of logging to do 72

4.3.17 int
HYPRE_StructPFMGSetPrintLevel (HYPRE_StructSolver solver, int print_level)
(Optional) Set the amount of printing to do to the screen 72

4.3.18 int
HYPRE_StructPFMGGetNumIterations (HYPRE_StructSolver solver, int* num_iterations)
Return the number of iterations taken 73

4.3.19 int
HYPRE_StructPFMGGetFinalRelativeResidualNorm (HYPRE_StructSolver solver, HYPRE_Real* norm)
Return the norm of the final relative residual 73

PFMG is a semicoarsening multigrid solver that uses pointwise relaxation. For periodic problems, users should try to set the grid size in periodic dimensions to be as close to a power-of-two as possible. That is, if the grid size in a periodic dimension is given by $N = 2^m * M$ where M is not a power-of-two, then M should be as small as possible. Large values of M will generally result in slower convergence rates.

4.3.1

int
HYPRE_StructPFMGCreate (MPI_Comm comm, HYPRE_StructSolver* solver)

Create a solver object

4.3.2

int **HYPRE_StructPFMGDestroy** (HYPRE_StructSolver solver)

Destroy a solver object

4.3.3

int
HYPRE_StructPFMGSetup (HYPRE_StructSolver solver,
HYPRE_StructMatrix A, HYPRE_StructVector b, HYPRE_StructVector x)

Prepare to solve the system. The coefficient data in `b` and `x` is ignored here, but information about the layout of the data may be used.

4.3.4

int
HYPRE_StructPFMGSolve (HYPRE_StructSolver solver,
HYPRE_StructMatrix A, HYPRE_StructVector b, HYPRE_StructVector x)

Solve the system

4.3.5

int
HYPRE_StructPFMGSetTol (HYPRE_StructSolver solver, HYPRE_Real tol)

(Optional) Set the convergence tolerance

4.3.6

int
HYPRE_StructPFMGSetMaxIter (HYPRE_StructSolver solver, int max_iter)

(Optional) Set maximum number of iterations

4.3.7

int
HYPRE_StructPFMGSetMaxLevels (HYPRE_StructSolver solver, int max_levels)

(Optional) Set maximum number of multigrid grid levels

4.3.8

int
HYPRE_StructPFMGSetRelChange (HYPRE_StructSolver solver, int rel_change)

(Optional) Additionally require that the relative difference in successive iterates be small

4.3.9

int **HYPRE_StructPFMGSetZeroGuess** (HYPRE_StructSolver solver)

(Optional) Use a zero initial guess. This allows the solver to cut corners in the case where a zero initial guess is needed (e.g., for preconditioning) to reduce compuational cost.

4.3.10

int **HYPRE_StructPFMGSetNonZeroGuess** (HYPRE_StructSolver solver)

(Optional) Use a nonzero initial guess. This is the default behavior, but this routine allows the user to switch back after using `SetZeroGuess`.

4.3.11

int
HYPRE_StructPFMGSetRelaxType (HYPRE_StructSolver solver, int relax_type)

(Optional) Set relaxation type.

Current relaxation methods set by `relax_type` are:

0 & Jacobi –
1 & Weighted Jacobi (default) –
2 & Red/Black Gauss-Seidel (symmetric: RB pre-relaxation, BR post-relaxation) –
3 & Red/Black Gauss-Seidel (nonsymmetric: RB pre- and post-relaxation) –

4.3.12

int
HYPRE_StructPFMGSetRAPType (HYPRE_StructSolver solver, int rap_type)

(Optional) Set type of coarse-grid operator to use.

Current operators set by `rap_type` are:

0 – Galerkin (default)
1 – non-Galerkin 5-pt or 7-pt stencils

Both operators are constructed algebraically. The non-Galerkin option maintains a 5-pt stencil in 2D and a 7-pt stencil in 3D on all grid levels. The stencil coefficients are computed by averaging techniques.

4.3.13

int
HYPRE_StructPFMGSetNumPreRelax (HYPRE_StructSolver solver, int num_pre_relax)

(Optional) Set number of relaxation sweeps before coarse-grid correction

4.3.14

int
HYPRE_StructPFMGSetNumPostRelax (HYPRE_StructSolver solver, int num_post_relax)

(Optional) Set number of relaxation sweeps after coarse-grid correction

4.3.15

int
HYPRE_StructPFMGSetSkipRelax (HYPRE_StructSolver solver, int skip_relax)

(Optional) Skip relaxation on certain grids for isotropic problems. This can greatly improve efficiency by eliminating unnecessary relaxations when the underlying problem is isotropic.

4.3.16

int **HYPRE_StructPFMGSetLogging** (HYPRE_StructSolver solver, int logging)

(Optional) Set the amount of logging to do

4.3.17

int
HYPRE_StructPFMGSetPrintLevel (HYPRE_StructSolver solver, int print_level)

(Optional) Set the amount of printing to do to the screen

4.3.18

int
HYPRE_StructPFMGGetNumIterations (HYPRE_StructSolver solver, int* num_iterations)

Return the number of iterations taken

4.3.19

int
HYPRE_StructPFMGGetFinalRelativeResidualNorm (HYPRE_StructSolver solver, HYPRE_Real* norm)

Return the norm of the final relative residual

4.4

Struct SMG Solver

Names

4.4.1 int
HYPRE_StructSMGCreate (MPI_Comm comm, HYPRE_StructSolver* solver)
Create a solver object 75

4.4.2 int
HYPRE_StructSMGDestroy (HYPRE_StructSolver solver)
Destroy a solver object 75

4.4.3 int
HYPRE_StructSMGSetup (HYPRE_StructSolver solver, HYPRE_StructMatrix A, HYPRE_StructVector b, HYPRE_StructVector x)
Prepare to solve the system. 75

4.4.4 int
HYPRE_StructSMGSolve (HYPRE_StructSolver solver, HYPRE_StructMatrix A, HYPRE_StructVector b, HYPRE_StructVector x)
Solve the system 75

4.4.5 int

HYPRE_StructSMGSetTol (HYPRE_StructSolver solver, HYPRE_Real tol)
(Optional) Set the convergence tolerance 76

4.4.6 int
HYPRE_StructSMGSetMaxIter (HYPRE_StructSolver solver, int max_iter)
(Optional) Set maximum number of iterations 76

4.4.7 int
HYPRE_StructSMGSetRelChange (HYPRE_StructSolver solver, int rel_change)
(Optional) Additionally require that the relative difference in successive iterates be small 76

4.4.8 int
HYPRE_StructSMGSetZeroGuess (HYPRE_StructSolver solver)
(Optional) Use a zero initial guess. 76

4.4.9 int
HYPRE_StructSMGSetNonZeroGuess (HYPRE_StructSolver solver)
(Optional) Use a nonzero initial guess. 77

4.4.10 int
HYPRE_StructSMGSetNumPreRelax (HYPRE_StructSolver solver, int num_pre_relax)
(Optional) Set number of relaxation sweeps before coarse-grid correction . 77

4.4.11 int
HYPRE_StructSMGSetNumPostRelax (HYPRE_StructSolver solver, int num_post_relax)
(Optional) Set number of relaxation sweeps after coarse-grid correction .. 77

4.4.12 int
HYPRE_StructSMGSetLogging (HYPRE_StructSolver solver, int logging)
(Optional) Set the amount of logging to do 77

4.4.13 int
HYPRE_StructSMGSetPrintLevel (HYPRE_StructSolver solver, int print_level)
(Optional) Set the amount of printing to do to the screen 78

4.4.14 int
HYPRE_StructSMGGetNumIterations (HYPRE_StructSolver solver, int* num_iterations)
Return the number of iterations taken 78

4.4.15 int
HYPRE_StructSMGGetFinalRelativeResidualNorm (HYPRE_StructSolver solver, HYPRE_Real* norm)
Return the norm of the final relative residual 78

SMG is a semicoarsening multigrid solver that uses plane smoothing (in 3D). The plane smoother calls a 2D SMG algorithm with line smoothing, and the line smoother is cyclic reduction (1D SMG). For periodic problems, the grid size in periodic dimensions currently must be a power-of-two.

4.4.1

int
HYPRE_StructSMGCreate (MPI_Comm comm, HYPRE_StructSolver* solver)

Create a solver object

4.4.2

int **HYPRE_StructSMGDestroy** (HYPRE_StructSolver solver)

Destroy a solver object

4.4.3

int
HYPRE_StructSMGSetup (HYPRE_StructSolver solver, HYPRE_StructMatrix A, HYPRE_StructVector b, HYPRE_StructVector x)

Prepare to solve the system. The coefficient data in `b` and `x` is ignored here, but information about the layout of the data may be used.

4.4.4

int
HYPRE_StructSMGSolve (HYPRE_StructSolver solver, HYPRE_StructMatrix A, HYPRE_StructVector b, HYPRE_StructVector x)

Solve the system

4.4.5

int **HYPRE_StructSMGSetTol** (HYPRE_StructSolver solver, HYPRE_Real tol)

(Optional) Set the convergence tolerance

4.4.6

int **HYPRE_StructSMGSetMaxIter** (HYPRE_StructSolver solver, int max_iter)

(Optional) Set maximum number of iterations

4.4.7

int
HYPRE_StructSMGSetRelChange (HYPRE_StructSolver solver, int rel_change)

(Optional) Additionally require that the relative difference in successive iterates be small

4.4.8

int **HYPRE_StructSMGSetZeroGuess** (HYPRE_StructSolver solver)

(Optional) Use a zero initial guess. This allows the solver to cut corners in the case where a zero initial guess is needed (e.g., for preconditioning) to reduce compuational cost.

4.4.9

int **HYPRE_StructSMGSetNonZeroGuess** (HYPRE_StructSolver solver)

(Optional) Use a nonzero initial guess. This is the default behavior, but this routine allows the user to switch back after using `SetZeroGuess`.

4.4.10

int
HYPRE_StructSMGSetNumPreRelax (HYPRE_StructSolver solver, int num_pre_relax)

(Optional) Set number of relaxation sweeps before coarse-grid correction

4.4.11

int
HYPRE_StructSMGSetNumPostRelax (HYPRE_StructSolver solver, int num_post_relax)

(Optional) Set number of relaxation sweeps after coarse-grid correction

4.4.12

int **HYPRE_StructSMGSetLogging** (HYPRE_StructSolver solver, int logging)

(Optional) Set the amount of logging to do

4.4.13

int
HYPRE_StructSMGSetPrintLevel (HYPRE_StructSolver solver, int print_level)

(Optional) Set the amount of printing to do to the screen

4.4.14

int
HYPRE_StructSMGGetNumIterations (HYPRE_StructSolver solver, int* num_iterations)

Return the number of iterations taken

4.4.15

int
HYPRE_StructSMGGetFinalRelativeResidualNorm (HYPRE_StructSolver solver, HYPRE_Real* norm)

Return the norm of the final relative residual

4.5 Struct CycRed Solver

Names

4.5.1 int
HYPRE_StructCycRedCreate (MPI_Comm comm, HYPRE_StructSolver* solver)
Create a solver object 79

4.5.2 int
HYPRE_StructCycRedDestroy (HYPRE_StructSolver solver)
Destroy a solver object 79

4.5.3 int

HYPRE_StructCycRedSetup (HYPRE_StructSolver solver,
HYPRE_StructMatrix A,
HYPRE_StructVector b,
HYPRE_StructVector x)
Prepare to solve the system. 80

4.5.4 int
HYPRE_StructCycRedSolve (HYPRE_StructSolver solver,
HYPRE_StructMatrix A,
HYPRE_StructVector b,
HYPRE_StructVector x)
Solve the system 80

4.5.5 int
HYPRE_StructCycRedSetTDim (HYPRE_StructSolver solver, int tdim) 80

4.5.6 int
HYPRE_StructCycRedSetBase (HYPRE_StructSolver solver, int ndim,
int* base_index, int* base_stride)
(Optional) Set the base index and stride for the embedded 1D systems. . . . 80

CycRed is a cyclic reduction solver that simultaneously solves a collection of 1D tridiagonal systems embedded in a d-dimensional grid.

4.5.1

int
HYPRE_StructCycRedCreate (MPI_Comm comm, HYPRE_StructSolver* solver)

Create a solver object

4.5.2

int **HYPRE_StructCycRedDestroy** (HYPRE_StructSolver solver)

Destroy a solver object

4.5.3

int
HYPRE_StructCycRedSetup (HYPRE_StructSolver solver, HYPRE_StructMatrix A, HYPRE_StructVector b, HYPRE_StructVector x)

Prepare to solve the system. The coefficient data in `b` and `x` is ignored here, but information about the layout of the data may be used.

4.5.4

int
HYPRE_StructCycRedSolve (HYPRE_StructSolver solver, HYPRE_StructMatrix A, HYPRE_StructVector b, HYPRE_StructVector x)

Solve the system

4.5.5

int **HYPRE_StructCycRedSetTDim** (HYPRE_StructSolver solver, int tdim)

(Optional) Set the dimension number for the embedded 1D tridiagonal systems. The default is `tdim` = 0.

4.5.6

int
HYPRE_StructCycRedSetBase (HYPRE_StructSolver solver, int ndim, int* base_index, int* base_stride)

(Optional) Set the base index and stride for the embedded 1D systems. The stride must be equal one in the dimension corresponding to the 1D systems (see HYPRE_StructCycRedSetTDim).

4.6

Struct PCG Solver

Names

4.6.1 int
HYPRE_StructPCGCreate (MPI_Comm comm,
HYPRE_StructSolver* solver)
Create a solver object 81

4.6.2 int
HYPRE_StructPCGDestroy (HYPRE_StructSolver solver)
Destroy a solver object 81

4.6.3 int
HYPRE_StructDiagScaleSetup (HYPRE_StructSolver solver,
HYPRE_StructMatrix A,
HYPRE_StructVector y,
HYPRE_StructVector x)
Setup routine for diagonal preconditioning 82

4.6.4 int
HYPRE_StructDiagScale (HYPRE_StructSolver solver,
HYPRE_StructMatrix HA,
HYPRE_StructVector Hy,
HYPRE_StructVector Hx)
Solve routine for diagonal preconditioning 82

These routines should be used in conjunction with the generic interface in PCG Solver.

4.6.1

int
HYPRE_StructPCGCreate (MPI_Comm comm, HYPRE_StructSolver* solver)

Create a solver object

4.6.2

int **HYPRE_StructPCGDestroy** (HYPRE_StructSolver solver)

Destroy a solver object

4.6.3

int
HYPRE_StructDiagScaleSetup (HYPRE_StructSolver solver, HYPRE_StructMatrix A, HYPRE_StructVector y, HYPRE_StructVector x)

Setup routine for diagonal preconditioning

4.6.4

int
HYPRE_StructDiagScale (HYPRE_StructSolver solver, HYPRE_StructMatrix HA, HYPRE_StructVector Hy, HYPRE_StructVector Hx)

Solve routine for diagonal preconditioning

4.7

Struct GMRES Solver

Names

4.7.1 int
HYPRE_StructGMRESCreate (MPI_Comm comm, HYPRE_StructSolver* solver)
Create a solver object 82

4.7.2 int
HYPRE_StructGMRESDestroy (HYPRE_StructSolver solver)
Destroy a solver object 83

These routines should be used in conjunction with the generic interface in GMRES Solver.

4.7.1

int
HYPRE_StructGMRESCreate (MPI_Comm comm, HYPRE_StructSolver* solver)

Create a solver object

4.7.2

int **HYPRE_StructGMRESDestroy** (HYPRE_StructSolver solver)

Destroy a solver object

4.8

Struct FlexGMRES Solver

Names

4.8.1 int
HYPRE_StructFlexGMRESCreate (MPI_Comm comm,
HYPRE_StructSolver* solver)
Create a solver object 83
4.8.2 int
HYPRE_StructFlexGMRESDestroy (HYPRE_StructSolver solver)
Destroy a solver object 84

These routines should be used in conjunction with the generic interface in FlexGMRES Solver.

4.8.1

int
HYPRE_StructFlexGMRESCreate (MPI_Comm comm, HYPRE_StructSolver*
solver)

Create a solver object

4.8.2

int **HYPRE_StructFlexGMRESDestroy** (HYPRE_StructSolver solver)

Destroy a solver object

4.9

Struct LGMRES Solver

Names

4.9.1 int
HYPRE_StructLGMRESCreate (MPI_Comm comm, HYPRE_StructSolver* solver)
Create a solver object 84

4.9.2 int
HYPRE_StructLGMRESDestroy (HYPRE_StructSolver solver)
Destroy a solver object 84

These routines should be used in conjunction with the generic interface in LGMRES Solver.

4.9.1

int
HYPRE_StructLGMRESCreate (MPI_Comm comm, HYPRE_StructSolver* solver)

Create a solver object

4.9.2

int **HYPRE_StructLGMRESDestroy** (HYPRE_StructSolver solver)

Destroy a solver object

4.10 Struct BiCGSTAB Solver

Names

4.10.1 int
HYPRE_StructBiCGSTABCreate (MPI_Comm comm,
HYPRE_StructSolver* solver)
Create a solver object 85

4.10.2 int
HYPRE_StructBiCGSTABDestroy (HYPRE_StructSolver solver)
Destroy a solver object 85

These routines should be used in conjunction with the generic interface in BiCGSTAB Solver.

4.10.1

int
HYPRE_StructBiCGSTABCreate (MPI_Comm comm, HYPRE_StructSolver* solver)

Create a solver object

4.10.2

int **HYPRE_StructBiCGSTABDestroy** (HYPRE_StructSolver solver)

Destroy a solver object

4.11 Struct Hybrid Solver

Names

4.11.1 int

HYPRE_StructHybridCreate (MPI_Comm comm,
HYPRE_StructSolver* solver)
Create a solver object 87

4.11.2 int
HYPRE_StructHybridDestroy (HYPRE_StructSolver solver)
Destroy a solver object 88

4.11.3 int
HYPRE_StructHybridSetup (HYPRE_StructSolver solver,
HYPRE_StructMatrix A,
HYPRE_StructVector b,
HYPRE_StructVector x)
Prepare to solve the system. 88

4.11.4 int
HYPRE_StructHybridSolve (HYPRE_StructSolver solver,
HYPRE_StructMatrix A,
HYPRE_StructVector b,
HYPRE_StructVector x)
Solve the system 88

4.11.5 int
HYPRE_StructHybridSetTol (HYPRE_StructSolver solver, HYPRE_Real tol)
(Optional) Set the convergence tolerance 88

4.11.6 int
HYPRE_StructHybridSetConvergenceTol (HYPRE_StructSolver solver,
HYPRE_Real cf_tol)
(Optional) Set an accepted convergence tolerance for diagonal scaling (DS). 89

4.11.7 int
HYPRE_StructHybridSetDSCGMaxIter (HYPRE_StructSolver solver,
int ds_max_its)
(Optional) Set maximum number of iterations for diagonal scaling (DS). . 89

4.11.8 int
HYPRE_StructHybridSetPCGMaxIter (HYPRE_StructSolver solver,
int pre_max_its)
(Optional) Set maximum number of iterations for general preconditioner (PRE). 89

4.11.9 int
HYPRE_StructHybridSetTwoNorm (HYPRE_StructSolver solver,
int two_norm)
(Optional) Use the two-norm in stopping criteria 89

4.11.10 int
HYPRE_StructHybridSetRelChange (HYPRE_StructSolver solver,
int rel_change)
(Optional) Additionally require that the relative difference in successive iterates be small 90

4.11.11 int
HYPRE_StructHybridSetSolverType (HYPRE_StructSolver solver,
int solver_type)
(Optional) Set the type of Krylov solver to use. 90

4.11.12 int

HYPRE_StructHybridSetKDim (HYPRE_StructSolver solver, int k_dim)
(Optional) Set the maximum size of the Krylov space when using GMRES 90

4.11.13 int
HYPRE_StructHybridSetPrecond (HYPRE_StructSolver solver, HYPRE_PtrToStructSolverFcn precond, HYPRE_PtrToStructSolverFcn precond_setup, HYPRE_StructSolver precond_solver)
(Optional) Set the preconditioner to use 90

4.11.14 int
HYPRE_StructHybridSetLogging (HYPRE_StructSolver solver, int logging)
(Optional) Set the amount of logging to do 91

4.11.15 int
HYPRE_StructHybridSetPrintLevel (HYPRE_StructSolver solver, int print_level)
(Optional) Set the amount of printing to do to the screen 91

4.11.16 int
HYPRE_StructHybridGetNumIterations (HYPRE_StructSolver solver, int* num_its)
Return the number of iterations taken 91

4.11.17 int
HYPRE_StructHybridGetDSCGNumIterations (HYPRE_StructSolver solver, int* ds_num_its)
Return the number of diagonal scaling iterations taken 91

4.11.18 int
HYPRE_StructHybridGetPCGNumIterations (HYPRE_StructSolver solver, int* pre_num_its)
Return the number of general preconditioning iterations taken 92

4.11.19 int
HYPRE_StructHybridGetFinalRelativeResidualNorm (HYPRE_StructSolver solver, HYPRE_Real* norm)
Return the norm of the final relative residual 92

4.11.1

int
HYPRE_StructHybridCreate (MPI_Comm comm, HYPRE_StructSolver* solver)

Create a solver object

4.11.2

int **HYPRE_StructHybridDestroy** (HYPRE_StructSolver solver)

Destroy a solver object

4.11.3

int
HYPRE_StructHybridSetup (HYPRE_StructSolver solver,
HYPRE_StructMatrix A, HYPRE_StructVector b, HYPRE_StructVector x)

Prepare to solve the system. The coefficient data in `b` and `x` is ignored here, but information about the layout of the data may be used.

4.11.4

int
HYPRE_StructHybridSolve (HYPRE_StructSolver solver,
HYPRE_StructMatrix A, HYPRE_StructVector b, HYPRE_StructVector x)

Solve the system

4.11.5

int
HYPRE_StructHybridSetTol (HYPRE_StructSolver solver, HYPRE_Real tol)

(Optional) Set the convergence tolerance

4.11.6

int
HYPRE_StructHybridSetConvergenceTol (HYPRE_StructSolver solver, HYPRE_Real cf_tol)

(Optional) Set an accepted convergence tolerance for diagonal scaling (DS). The solver will switch preconditioners if the convergence of DS is slower than `cf_tol`.

4.11.7

int
HYPRE_StructHybridSetDSCGMaxIter (HYPRE_StructSolver solver, int ds_max_its)

(Optional) Set maximum number of iterations for diagonal scaling (DS). The solver will switch preconditioners if DS reaches `ds_max_its`.

4.11.8

int
HYPRE_StructHybridSetPCGMaxIter (HYPRE_StructSolver solver, int pre_max_its)

(Optional) Set maximum number of iterations for general preconditioner (PRE). The solver will stop if PRE reaches `pre_max_its`.

4.11.9

int
HYPRE_StructHybridSetTwoNorm (HYPRE_StructSolver solver, int two_norm)

(Optional) Use the two-norm in stopping criteria

4.11.10

int
HYPRE_StructHybridSetRelChange (HYPRE_StructSolver solver, int rel_change)

(Optional) Additionally require that the relative difference in successive iterates be small

4.11.11

int
HYPRE_StructHybridSetSolverType (HYPRE_StructSolver solver, int solver_type)

(Optional) Set the type of Krylov solver to use.

Current krylov methods set by `solver_type` are:

0 – PCG (default)
1 – GMRES
2 – BiCGSTAB

4.11.12

int **HYPRE_StructHybridSetKDim** (HYPRE_StructSolver solver, int k_dim)

(Optional) Set the maximum size of the Krylov space when using GMRES

4.11.13

int
HYPRE_StructHybridSetPrecond (HYPRE_StructSolver solver, HYPRE_PtrToStructSolverFcn precond, HYPRE_PtrToStructSolverFcn precond_setup, HYPRE_StructSolver precond_solver)

(Optional) Set the preconditioner to use

4.11.14

int **HYPRE_StructHybridSetLogging** (HYPRE_StructSolver solver, int logging)

(Optional) Set the amount of logging to do

4.11.15

int
HYPRE_StructHybridSetPrintLevel (HYPRE_StructSolver solver, int print_level)

(Optional) Set the amount of printing to do to the screen

4.11.16

int
HYPRE_StructHybridGetNumIterations (HYPRE_StructSolver solver, int* num_its)

Return the number of iterations taken

4.11.17

int
HYPRE_StructHybridGetDSCGNumIterations (HYPRE_StructSolver solver, int* ds_num_its)

Return the number of diagonal scaling iterations taken

4.11.18

int
HYPRE_StructHybridGetPCGNumIterations (HYPRE_StructSolver solver, int* pre_num_its)

Return the number of general preconditioning iterations taken

4.11.19

int
HYPRE_StructHybridGetFinalRelativeResidualNorm
(HYPRE_StructSolver solver, HYPRE_Real* norm)

Return the norm of the final relative residual

4.12 Struct LOBPCG Eigensolver

Names

4.12.1 int
HYPRE_StructSetupInterpreter (mv_InterfaceInterpreter* i)
Load interface interpreter. 92

4.12.2 int
HYPRE_StructSetupMatvec (HYPRE_MatvecFunctions* mv)
Load Matvec interpreter with hypre_StructKrylov functions 93

These routines should be used in conjunction with the generic interface in LOBPCG Eigensolver.

4.12.1

int **HYPRE_StructSetupInterpreter** (mv_InterfaceInterpreter* i)

Load interface interpreter. Vector part loaded with hypre_StructKrylov functions and multivector part loaded with mv_TempMultiVector functions.

4.12.2

int **HYPRE_StructSetupMatvec** (HYPRE_MatvecFunctions* mv)

Load Matvec interpreter with hypre_StructKrylov functions

5 SStruct Solvers

Names

5.1 **SStruct Solvers** 94

5.2 **SStruct SysPFMG Solver** 95

5.3 **SStruct Split Solver** 101

5.4 **SStruct FAC Solver** 105

5.5 **SStruct Maxwell Solver** 114

5.6 **SStruct PCG Solver** 121

5.7 **SStruct GMRES Solver** 122

5.8 **SStruct FlexGMRES Solver** 123

5.9 **SStruct LGMRES Solver** 124

5.10 **SStruct BiCGSTAB Solver** 125

5.11 **SStruct LOBPCG Eigensolver** 126

These solvers use matrix/vector storage schemes that are taylored to semi-structured grid problems.

5.1 SStruct Solvers

Names

5.1.1 typedef struct hypre_SStructSolver_struct ***HYPRE_SStructSolver**
The solver object 95

5.1.1

typedef struct hypre_SStructSolver_struct ***HYPRE_SStructSolver**

The solver object

5.2

SStruct SysPFMG Solver

Names

5.2.1 int
HYPRE_SStructSysPFMGCreate (MPI_Comm comm,
HYPRE_SStructSolver* solver)
Create a solver object 97

5.2.2 int
HYPRE_SStructSysPFMGDestroy (HYPRE_SStructSolver solver)
Destroy a solver object. 97

5.2.3 int
HYPRE_SStructSysPFMGSetup (HYPRE_SStructSolver solver,
HYPRE_SStructMatrix A,
HYPRE_SStructVector b,
HYPRE_SStructVector x)
Prepare to solve the system. 97

5.2.4 int
HYPRE_SStructSysPFMGSolve (HYPRE_SStructSolver solver,
HYPRE_SStructMatrix A,
HYPRE_SStructVector b,
HYPRE_SStructVector x)
Solve the system 97

5.2.5 int
HYPRE_SStructSysPFMGSetTol (HYPRE_SStructSolver solver,
HYPRE_Real tol)
(Optional) Set the convergence tolerance 98

5.2.6 int
HYPRE_SStructSysPFMGSetMaxIter (HYPRE_SStructSolver solver,
int max_iter)
(Optional) Set maximum number of iterations 98

5.2.7 int
HYPRE_SStructSysPFMGSetRelChange (HYPRE_SStructSolver solver,
int rel_change)
(Optional) Additionally require that the relative difference in successive iterates be small 98

5.2.8 int

HYPRE_SStructSysPFMGSetZeroGuess (HYPRE_SStructSolver solver)
(Optional) Use a zero initial guess. 98

5.2.9 int
HYPRE_SStructSysPFMGSetNonZeroGuess (HYPRE_SStructSolver solver)
(Optional) Use a nonzero initial guess. 99

5.2.10 int
HYPRE_SStructSysPFMGSetRelaxType (HYPRE_SStructSolver solver, int relax_type)
(Optional) Set relaxation type. 99

5.2.11 int
HYPRE_SStructSysPFMGSetJacobiWeight (HYPRE_SStructSolver solver, HYPRE_Real weight)
(Optional) Set Jacobi Weight 99

5.2.12 int
HYPRE_SStructSysPFMGSetNumPreRelax (HYPRE_SStructSolver solver, int num_pre_relax)
(Optional) Set number of relaxation sweeps before coarse-grid correction . 99

5.2.13 int
HYPRE_SStructSysPFMGSetNumPostRelax (HYPRE_SStructSolver solver, int num_post_relax)
(Optional) Set number of relaxation sweeps after coarse-grid correction .. 100

5.2.14 int
HYPRE_SStructSysPFMGSetSkipRelax (HYPRE_SStructSolver solver, int skip_relax)
(Optional) Skip relaxation on certain grids for isotropic problems. 100

5.2.15 int
HYPRE_SStructSysPFMGSetLogging (HYPRE_SStructSolver solver, int logging)
(Optional) Set the amount of logging to do 100

5.2.16 int
HYPRE_SStructSysPFMGSetPrintLevel (HYPRE_SStructSolver solver, int print_level)
(Optional) Set the amount of printing to do to the screen 100

5.2.17 int
HYPRE_SStructSysPFMGGetNumIterations (HYPRE_SStructSolver solver, int* num_iterations)
Return the number of iterations taken 101

5.2.18 int
HYPRE_SStructSysPFMGGetFinalRelativeResidualNorm (HYPRE_SStructSolver solver, HYPRE_Real* norm)
Return the norm of the final relative residual 101

SysPFMG is a semicoarsening multigrid solver similar to PFMG, but for systems of PDEs. For periodic problems, users should try to set the grid size in periodic dimensions to be as close to a power-of-two as possible (for more details, see Struct PFMG Solver).

5.2.1

int
HYPRE_SStructSysPFMGCreate (MPI_Comm comm, HYPRE_SStructSolver* solver)

Create a solver object

5.2.2

int **HYPRE_SStructSysPFMGDestroy** (HYPRE_SStructSolver solver)

Destroy a solver object. An object should be explicitly destroyed using this destructor when the user's code no longer needs direct access to it. Once destroyed, the object must not be referenced again. Note that the object may not be deallocated at the completion of this call, since there may be internal package references to the object. The object will then be destroyed when all internal reference counts go to zero.

5.2.3

int
HYPRE_SStructSysPFMGSetup (HYPRE_SStructSolver solver, HYPRE_SStructMatrix A, HYPRE_SStructVector b, HYPRE_SStructVector x)

Prepare to solve the system. The coefficient data in `b` and `x` is ignored here, but information about the layout of the data may be used.

5.2.4

int
HYPRE_SStructSysPFMGSolve (HYPRE_SStructSolver solver, HYPRE_SStructMatrix A, HYPRE_SStructVector b, HYPRE_SStructVector x)

Solve the system

5.2.5

int
HYPRE_SStructSysPFMGSetTol (HYPRE_SStructSolver solver, HYPRE_Real tol)

(Optional) Set the convergence tolerance

5.2.6

int
HYPRE_SStructSysPFMGSetMaxIter (HYPRE_SStructSolver solver, int max_iter)

(Optional) Set maximum number of iterations

5.2.7

int
HYPRE_SStructSysPFMGSetRelChange (HYPRE_SStructSolver solver, int rel_change)

(Optional) Additionally require that the relative difference in successive iterates be small

5.2.8

int **HYPRE_SStructSysPFMGSetZeroGuess** (HYPRE_SStructSolver solver)

(Optional) Use a zero initial guess. This allows the solver to cut corners in the case where a zero initial guess is needed (e.g., for preconditioning) to reduce compuational cost.

5.2.9

int
HYPRE_SStructSysPFMGSetNonZeroGuess (HYPRE_SStructSolver solver)

(Optional) Use a nonzero initial guess. This is the default behavior, but this routine allows the user to switch back after using `SetZeroGuess`.

5.2.10

int
HYPRE_SStructSysPFMGSetRelaxType (HYPRE_SStructSolver solver, int relax_type)

(Optional) Set relaxation type.

Current relaxation methods set by `relax_type` are:

0 – Jacobi
1 – Weighted Jacobi (default)
2 – Red/Black Gauss-Seidel (symmetric: RB pre-relaxation, BR post-relaxation)

5.2.11

int
HYPRE_SStructSysPFMGSetJacobiWeight (HYPRE_SStructSolver solver, HYPRE_Real weight)

(Optional) Set Jacobi Weight

5.2.12

int
HYPRE_SStructSysPFMGSetNumPreRelax (HYPRE_SStructSolver solver, int num_pre_relax)

(Optional) Set number of relaxation sweeps before coarse-grid correction

5.2.13

int
HYPRE_SStructSysPFMGSetNumPostRelax (HYPRE_SStructSolver solver, int num_post_relax)

(Optional) Set number of relaxation sweeps after coarse-grid correction

5.2.14

int
HYPRE_SStructSysPFMGSetSkipRelax (HYPRE_SStructSolver solver, int skip_relax)

(Optional) Skip relaxation on certain grids for isotropic problems. This can greatly improve efficiency by eliminating unnecessary relaxations when the underlying problem is isotropic.

5.2.15

int
HYPRE_SStructSysPFMGSetLogging (HYPRE_SStructSolver solver, int logging)

(Optional) Set the amount of logging to do

5.2.16

int
HYPRE_SStructSysPFMGSetPrintLevel (HYPRE_SStructSolver solver, int print_level)

(Optional) Set the amount of printing to do to the screen

5.2.17

int
HYPRE_SStructSysPFMGGetNumIterations (HYPRE_SStructSolver solver, int* num_iterations)

Return the number of iterations taken

5.2.18

int
HYPRE_SStructSysPFMGGetFinalRelativeResidualNorm (HYPRE_SStructSolver solver, HYPRE_Real* norm)

Return the norm of the final relative residual

5.3

SStruct Split Solver

Names

5.3.1 int
HYPRE_SStructSplitCreate (MPI_Comm comm, HYPRE_SStructSolver* solver)
Create a solver object 102

5.3.2 int
HYPRE_SStructSplitDestroy (HYPRE_SStructSolver solver)
Destroy a solver object. 102

5.3.3 int
HYPRE_SStructSplitSetup (HYPRE_SStructSolver solver, HYPRE_SStructMatrix A, HYPRE_SStructVector b, HYPRE_SStructVector x)
Prepare to solve the system. 103

5.3.4 int
HYPRE_SStructSplitSolve (HYPRE_SStructSolver solver, HYPRE_SStructMatrix A, HYPRE_SStructVector b, HYPRE_SStructVector x)
Solve the system 103

5.3.5 int

HYPRE_SStructSplitSetTol (HYPRE_SStructSolver solver, HYPRE_Real tol)
(Optional) Set the convergence tolerance 103

5.3.6 int
HYPRE_SStructSplitSetMaxIter (HYPRE_SStructSolver solver, int max_iter)
(Optional) Set maximum number of iterations 103

5.3.7 int
HYPRE_SStructSplitSetZeroGuess (HYPRE_SStructSolver solver)
(Optional) Use a zero initial guess. 104

5.3.8 int
HYPRE_SStructSplitSetNonZeroGuess (HYPRE_SStructSolver solver)
(Optional) Use a nonzero initial guess. 104

5.3.9 int
HYPRE_SStructSplitSetStructSolver (HYPRE_SStructSolver solver, int ssolver)
(Optional) Set up the type of diagonal struct solver. 104

5.3.10 int
HYPRE_SStructSplitGetNumIterations (HYPRE_SStructSolver solver, int* num_iterations)
Return the number of iterations taken 104

5.3.11 int
HYPRE_SStructSplitGetFinalRelativeResidualNorm (HYPRE_SStructSolver solver, HYPRE_Real* norm)
Return the norm of the final relative residual 105

5.3.1

int
HYPRE_SStructSplitCreate (MPI_Comm comm, HYPRE_SStructSolver* solver)

Create a solver object

5.3.2

int **HYPRE_SStructSplitDestroy** (HYPRE_SStructSolver solver)

Destroy a solver object. An object should be explicitly destroyed using this destructor when the user's code no longer needs direct access to it. Once destroyed, the object must not be referenced again. Note that the object may not be deallocated at the completion of this call, since there may be internal package references to the object. The object will then be destroyed when all internal reference counts go to zero.

5.3.3

int
HYPRE_SStructSplitSetup (HYPRE_SStructSolver solver,
HYPRE_SStructMatrix A, HYPRE_SStructVector b, HYPRE_SStructVector x)

Prepare to solve the system. The coefficient data in `b` and `x` is ignored here, but information about the layout of the data may be used.

5.3.4

int
HYPRE_SStructSplitSolve (HYPRE_SStructSolver solver,
HYPRE_SStructMatrix A, HYPRE_SStructVector b, HYPRE_SStructVector x)

Solve the system

5.3.5

int
HYPRE_SStructSplitSetTol (HYPRE_SStructSolver solver, HYPRE_Real tol)

(Optional) Set the convergence tolerance

5.3.6

int
HYPRE_SStructSplitSetMaxIter (HYPRE_SStructSolver solver, int max_iter)

(Optional) Set maximum number of iterations

5.3.7

int **HYPRE_SStructSplitSetZeroGuess** (HYPRE_SStructSolver solver)

(Optional) Use a zero initial guess. This allows the solver to cut corners in the case where a zero initial guess is needed (e.g., for preconditioning) to reduce compuational cost.

5.3.8

int **HYPRE_SStructSplitSetNonZeroGuess** (HYPRE_SStructSolver solver)

(Optional) Use a nonzero initial guess. This is the default behavior, but this routine allows the user to switch back after using `SetZeroGuess`.

5.3.9

int
HYPRE_SStructSplitSetStructSolver (HYPRE_SStructSolver solver, int ssolver)

(Optional) Set up the type of diagonal struct solver. Either `ssolver` is set to `HYPRE_SMG` or `HYPRE_PFMG`.

5.3.10

int
HYPRE_SStructSplitGetNumIterations (HYPRE_SStructSolver solver, int* num_iterations)

Return the number of iterations taken

5.3.11

int
HYPRE_SStructSplitGetFinalRelativeResidualNorm
(HYPRE_SStructSolver solver, HYPRE_Real* norm)

Return the norm of the final relative residual

5.4

SStruct FAC Solver

Names

5.4.1 int
HYPRE_SStructFACCreate (MPI_Comm comm,
HYPRE_SStructSolver* solver)
Create a solver object 107

5.4.2 int
HYPRE_SStructFACDestroy2 (HYPRE_SStructSolver solver)
Destroy a solver object. 108

5.4.3 int
HYPRE_SStructFACAMR_RAP (HYPRE_SStructMatrix A,
int (*rfactors)[HYPRE_MAXDIM],
HYPRE_SStructMatrix* fac_A)
Re-distribute the composite matrix so that the amr hierachy is approximately nested. 108

5.4.4 int
HYPRE_SStructFACSetup2 (HYPRE_SStructSolver solver,
HYPRE_SStructMatrix A,
HYPRE_SStructVector b,
HYPRE_SStructVector x)
Set up the FAC solver structure 108

5.4.5 int
HYPRE_SStructFACSolve3 (HYPRE_SStructSolver solver,
HYPRE_SStructMatrix A,
HYPRE_SStructVector b,
HYPRE_SStructVector x)
Solve the system 108

5.4.6 int
HYPRE_SStructFACSetPLevels (HYPRE_SStructSolver solver, int nparts,
int* plevels)
Set up amr structure 109

5.4.7 int

HYPRE_SStructFACSetPRefinements (HYPRE_SStructSolver solver, int nparts, int (*rfactors)[HYPRE_MAXDIM])
Set up amr refinement factors 109

5.4.8 int
HYPRE_SStructFACZeroCFSten (HYPRE_SStructMatrix A, HYPRE_SStructGrid grid, int part, int rfactors[HYPRE_MAXDIM])
(Optional, but user must make sure that they do this function otherwise) Zero off the coarse level stencils reaching into a fine level grid 109

5.4.9 int
HYPRE_SStructFACZeroFCSten (HYPRE_SStructMatrix A, HYPRE_SStructGrid grid, int part)
(Optional, but user must make sure that they do this function otherwise) Zero off the fine level stencils reaching into a coarse level grid 109

5.4.10 int
HYPRE_SStructFACZeroAMRMatrixData (HYPRE_SStructMatrix A, int part_crse, int rfactors[HYPRE_MAXDIM])
(Optional, but user must make sure that they do this function otherwise) Places the identity in the coarse grid matrix underlying the fine patches. . 110

5.4.11 int
HYPRE_SStructFACZeroAMRVectorData (HYPRE_SStructVector b, int* plevels, int (*rfactors)[HYPRE_MAXDIM])
(Optional, but user must make sure that they do this function otherwise) Places zeros in the coarse grid vector underlying the fine patches. 110

5.4.12 int
HYPRE_SStructFACSetMaxLevels (HYPRE_SStructSolver solver, int max_levels)
(Optional) Set maximum number of FAC levels 110

5.4.13 int
HYPRE_SStructFACSetTol (HYPRE_SStructSolver solver, HYPRE_Real tol)
(Optional) Set the convergence tolerance 110

5.4.14 int
HYPRE_SStructFACSetMaxIter (HYPRE_SStructSolver solver, int max_iter)
(Optional) Set maximum number of iterations 111

5.4.15 int
HYPRE_SStructFACSetRelChange (HYPRE_SStructSolver solver, int rel_change)
(Optional) Additionally require that the relative difference in successive iterates be small 111

5.4.16 int
HYPRE_SStructFACSetZeroGuess (HYPRE_SStructSolver solver)
(Optional) Use a zero initial guess. 111

5.4.17 int

HYPRE_SStructFACSetNonZeroGuess (HYPRE_SStructSolver solver)
(Optional) Use a nonzero initial guess. 111

5.4.18 int
HYPRE_SStructFACSetRelaxType (HYPRE_SStructSolver solver, int relax_type)
(Optional) Set relaxation type. 112

5.4.19 int
HYPRE_SStructFACSetJacobiWeight (HYPRE_SStructSolver solver, HYPRE_Real weight)
(Optional) Set Jacobi weight if weighted Jacobi is used 112

5.4.20 int
HYPRE_SStructFACSetNumPreRelax (HYPRE_SStructSolver solver, int num_pre_relax)
(Optional) Set number of relaxation sweeps before coarse-grid correction . 112

5.4.21 int
HYPRE_SStructFACSetNumPostRelax (HYPRE_SStructSolver solver, int num_post_relax)
(Optional) Set number of relaxation sweeps after coarse-grid correction .. 112

5.4.22 int
HYPRE_SStructFACSetCoarseSolverType (HYPRE_SStructSolver solver, int csolver_type)
(Optional) Set coarsest solver type. 113

5.4.23 int
HYPRE_SStructFACSetLogging (HYPRE_SStructSolver solver, int logging)
(Optional) Set the amount of logging to do 113

5.4.24 int
HYPRE_SStructFACGetNumIterations (HYPRE_SStructSolver solver, int* num_iterations)
Return the number of iterations taken 113

5.4.25 int
HYPRE_SStructFACGetFinalRelativeResidualNorm (HYPRE_SStructSolver solver, HYPRE_Real* norm)
Return the norm of the final relative residual 113

5.4.1

int
HYPRE_SStructFACCreate (MPI_Comm comm, HYPRE_SStructSolver* solver)

Create a solver object

5.4.2

int **HYPRE_SStructFACDestroy2** (HYPRE_SStructSolver solver)

Destroy a solver object. An object should be explicitly destroyed using this destructor when the user's code no longer needs direct access to it. Once destroyed, the object must not be referenced again. Note that the object may not be deallocated at the completion of this call, since there may be internal package references to the object. The object will then be destroyed when all internal reference counts go to zero.

5.4.3

int
HYPRE_SStructFACAMR_RAP (HYPRE_SStructMatrix A, int (*rfactors)[HYPRE_MAXDIM], HYPRE_SStructMatrix* fac_A)

Re-distribute the composite matrix so that the amr hierachy is approximately nested. Coarse underlying operators are also formed.

5.4.4

int
HYPRE_SStructFACSetup2 (HYPRE_SStructSolver solver, HYPRE_SStructMatrix A, HYPRE_SStructVector b, HYPRE_SStructVector x)

Set up the FAC solver structure

5.4.5

int
HYPRE_SStructFACSolve3 (HYPRE_SStructSolver solver, HYPRE_SStructMatrix A, HYPRE_SStructVector b, HYPRE_SStructVector x)

Solve the system

5.4.6

int
HYPRE_SStructFACSetPLevels (HYPRE_SStructSolver solver, int nparts, int* plevels)

Set up amr structure

5.4.7

int
HYPRE_SStructFACSetPRefinements (HYPRE_SStructSolver solver, int nparts, int (*rfactors)[HYPRE_MAXDIM])

Set up amr refinement factors

5.4.8

int
HYPRE_SStructFACZeroCFSten (HYPRE_SStructMatrix A, HYPRE_SStructGrid grid, int part, int rfactors[HYPRE_MAXDIM])

(Optional, but user must make sure that they do this function otherwise) Zero off the coarse level stencils reaching into a fine level grid

5.4.9

int
HYPRE_SStructFACZeroFCSten (HYPRE_SStructMatrix A, HYPRE_SStructGrid grid, int part)

(Optional, but user must make sure that they do this function otherwise) Zero off the fine level stencils reaching into a coarse level grid

5.4.10

int
HYPRE_SStructFACZeroAMRMatrixData (HYPRE_SStructMatrix A, int part_crse, int rfactors[HYPRE_MAXDIM])

(Optional, but user must make sure that they do this function otherwise) Places the identity in the coarse grid matrix underlying the fine patches. Required between each pair of amr levels.

5.4.11

int
HYPRE_SStructFACZeroAMRVectorData (HYPRE_SStructVector b, int* plevels, int (*rfactors)[HYPRE_MAXDIM])

(Optional, but user must make sure that they do this function otherwise) Places zeros in the coarse grid vector underlying the fine patches. Required between each pair of amr levels.

5.4.12

int
HYPRE_SStructFACSetMaxLevels (HYPRE_SStructSolver solver, int max_levels)

(Optional) Set maximum number of FAC levels

5.4.13

int
HYPRE_SStructFACSetTol (HYPRE_SStructSolver solver, HYPRE_Real tol)

(Optional) Set the convergence tolerance

5.4.14

int
HYPRE_SStructFACSetMaxIter (HYPRE_SStructSolver solver, int max_iter)

(Optional) Set maximum number of iterations

5.4.15

int
HYPRE_SStructFACSetRelChange (HYPRE_SStructSolver solver, int rel_change)

(Optional) Additionally require that the relative difference in successive iterates be small

5.4.16

int **HYPRE_SStructFACSetZeroGuess** (HYPRE_SStructSolver solver)

(Optional) Use a zero initial guess. This allows the solver to cut corners in the case where a zero initial guess is needed (e.g., for preconditioning) to reduce compuational cost.

5.4.17

int **HYPRE_SStructFACSetNonZeroGuess** (HYPRE_SStructSolver solver)

(Optional) Use a nonzero initial guess. This is the default behavior, but this routine allows the user to switch back after using `SetZeroGuess`.

5.4.18

int
HYPRE_SStructFACSetRelaxType (HYPRE_SStructSolver solver, int relax_type)

(Optional) Set relaxation type. See HYPRE_SStructSysPFMGSetRelaxType for appropriate values of `relax_type`.

5.4.19

int
HYPRE_SStructFACSetJacobiWeight (HYPRE_SStructSolver solver, HYPRE_Real weight)

(Optional) Set Jacobi weight if weighted Jacobi is used

5.4.20

int
HYPRE_SStructFACSetNumPreRelax (HYPRE_SStructSolver solver, int num_pre_relax)

(Optional) Set number of relaxation sweeps before coarse-grid correction

5.4.21

int
HYPRE_SStructFACSetNumPostRelax (HYPRE_SStructSolver solver, int num_post_relax)

(Optional) Set number of relaxation sweeps after coarse-grid correction

5.4.22

int
HYPRE_SStructFACSetCoarseSolverType (HYPRE_SStructSolver solver, int csolver_type)

(Optional) Set coarsest solver type.

Current solver types set by `csolver_type` are:

1 – SysPFMG-PCG (default)
2 – SysPFMG

5.4.23

int **HYPRE_SStructFACSetLogging** (HYPRE_SStructSolver solver, int logging)

(Optional) Set the amount of logging to do

5.4.24

int
HYPRE_SStructFACGetNumIterations (HYPRE_SStructSolver solver, int* num_iterations)

Return the number of iterations taken

5.4.25

int
HYPRE_SStructFACGetFinalRelativeResidualNorm
(HYPRE_SStructSolver solver, HYPRE_Real* norm)

Return the norm of the final relative residual

5.5

SStruct Maxwell Solver

Names

5.5.1 int
HYPRE_SStructMaxwellCreate (MPI_Comm comm,
HYPRE_SStructSolver* solver)
Create a solver object 116

5.5.2 int
HYPRE_SStructMaxwellDestroy (HYPRE_SStructSolver solver)
Destroy a solver object. 116

5.5.3 int
HYPRE_SStructMaxwellSetup (HYPRE_SStructSolver solver,
HYPRE_SStructMatrix A,
HYPRE_SStructVector b,
HYPRE_SStructVector x)
Prepare to solve the system. 116

5.5.4 int
HYPRE_SStructMaxwellSolve (HYPRE_SStructSolver solver,
HYPRE_SStructMatrix A,
HYPRE_SStructVector b,
HYPRE_SStructVector x)
Solve the system. 117

5.5.5 int
HYPRE_SStructMaxwellSolve2 (HYPRE_SStructSolver solver,
HYPRE_SStructMatrix A,
HYPRE_SStructVector b,
HYPRE_SStructVector x)
Solve the system. 117

5.5.6 int
HYPRE_SStructMaxwellSetGrad (HYPRE_SStructSolver solver,
HYPRE_ParCSRMatrix T)
Sets the gradient operator in the Maxwell solver 117

5.5.7 int
HYPRE_SStructMaxwellSetRfactors (HYPRE_SStructSolver solver,
int rfactors[HYPRE_MAXDIM])
Sets the coarsening factor 117

5.5.8 int
HYPRE_SStructMaxwellPhysBdy (HYPRE_SStructGrid* grid_l,
int num_levels,
int rfactors[HYPRE_MAXDIM],
int*** BdryRanks_ptr,
int** BdryRanksCnt_ptr)
Finds the physical boundary row ranks on all levels 118

5.5.9 int

HYPRE_SStructMaxwellEliminateRowsCols (HYPRE_ParCSRMatrix parA, int nrows, int* rows)
Eliminates the rows and cols corresponding to the physical boundary in a parcsr matrix 118

5.5.10 int
HYPRE_SStructMaxwellZeroVector (HYPRE_ParVector b, int* rows, int nrows)
Zeros the rows corresponding to the physical boundary in a par vector 118

5.5.11 int
HYPRE_SStructMaxwellSetSetConstantCoef (HYPRE_SStructSolver solver, int flag)
(Optional) Set the constant coefficient flag- Nedelec interpolation used ... 118

5.5.12 int
HYPRE_SStructMaxwellGrad (HYPRE_SStructGrid grid, HYPRE_ParCSRMatrix* T)
(Optional) Creates a gradient matrix from the grid. 119

5.5.13 int
HYPRE_SStructMaxwellSetTol (HYPRE_SStructSolver solver, HYPRE_Real tol)
(Optional) Set the convergence tolerance 119

5.5.14 int
HYPRE_SStructMaxwellSetMaxIter (HYPRE_SStructSolver solver, int max_iter)
(Optional) Set maximum number of iterations 119

5.5.15 int
HYPRE_SStructMaxwellSetRelChange (HYPRE_SStructSolver solver, int rel_change)
(Optional) Additionally require that the relative difference in successive iterates be small 119

5.5.16 int
HYPRE_SStructMaxwellSetNumPreRelax (HYPRE_SStructSolver solver, int num_pre_relax)
(Optional) Set number of relaxation sweeps before coarse-grid correction . 120

5.5.17 int
HYPRE_SStructMaxwellSetNumPostRelax (HYPRE_SStructSolver solver, int num_post_relax)
(Optional) Set number of relaxation sweeps after coarse-grid correction .. 120

5.5.18 int
HYPRE_SStructMaxwellSetLogging (HYPRE_SStructSolver solver, int logging)
(Optional) Set the amount of logging to do 120

5.5.19 int
HYPRE_SStructMaxwellGetNumIterations (HYPRE_SStructSolver solver, int* num_iterations)
Return the number of iterations taken 120

5.5.20 int

HYPRE_SStructMaxwellGetFinalRelativeResidualNorm (HYPRE_SStructSolver solver, HYPRE_Real* norm)
Return the norm of the final relative residual 121

5.5.1

int
HYPRE_SStructMaxwellCreate (MPI_Comm comm, HYPRE_SStructSolver* solver)

Create a solver object

5.5.2

int **HYPRE_SStructMaxwellDestroy** (HYPRE_SStructSolver solver)

Destroy a solver object. An object should be explicitly destroyed using this destructor when the user's code no longer needs direct access to it. Once destroyed, the object must not be referenced again. Note that the object may not be deallocated at the completion of this call, since there may be internal package references to the object. The object will then be destroyed when all internal reference counts go to zero.

5.5.3

int
HYPRE_SStructMaxwellSetup (HYPRE_SStructSolver solver, HYPRE_SStructMatrix A, HYPRE_SStructVector b, HYPRE_SStructVector x)

Prepare to solve the system. The coefficient data in `b` and `x` is ignored here, but information about the layout of the data may be used.

5.5.4

int
HYPRE_SStructMaxwellSolve (HYPRE_SStructSolver solver, HYPRE_SStructMatrix A, HYPRE_SStructVector b, HYPRE_SStructVector x)

Solve the system. Full coupling of the augmented system used throughout the multigrid hierarchy.

5.5.5

int
HYPRE_SStructMaxwellSolve2 (HYPRE_SStructSolver solver, HYPRE_SStructMatrix A, HYPRE_SStructVector b, HYPRE_SStructVector x)

Solve the system. Full coupling of the augmented system used only on the finest level, i.e., the node and edge multigrid cycles are coupled only on the finest level.

5.5.6

int
HYPRE_SStructMaxwellSetGrad (HYPRE_SStructSolver solver, HYPRE_ParCSRMatrix T)

Sets the gradient operator in the Maxwell solver

5.5.7

int
HYPRE_SStructMaxwellSetRfactors (HYPRE_SStructSolver solver, int rfactors[HYPRE_MAXDIM])

Sets the coarsening factor

5.5.8

int
HYPRE_SStructMaxwellPhysBdy (HYPRE_SStructGrid* grid_l, int num_levels, int rfactors[HYPRE_MAXDIM], int*** BdryRanks_ptr, int** BdryRanksCnt_ptr)

Finds the physical boundary row ranks on all levels

5.5.9

int
HYPRE_SStructMaxwellEliminateRowsCols (HYPRE_ParCSRMatrix parA, int nrows, int* rows)

Eliminates the rows and cols corresponding to the physical boundary in a parcsr matrix

5.5.10

int
HYPRE_SStructMaxwellZeroVector (HYPRE_ParVector b, int* rows, int nrows)

Zeros the rows corresponding to the physical boundary in a par vector

5.5.11

int
HYPRE_SStructMaxwellSetSetConstantCoef (HYPRE_SStructSolver solver, int flag)

(Optional) Set the constant coefficient flag- Nedelec interpolation used

5.5.12

int
HYPRE_SStructMaxwellGrad (HYPRE_SStructGrid grid, HYPRE_ParCSRMatrix* T)

(Optional) Creates a gradient matrix from the grid. This presupposes a particular orientation of the edge elements.

5.5.13

int
HYPRE_SStructMaxwellSetTol (HYPRE_SStructSolver solver, HYPRE_Real tol)

(Optional) Set the convergence tolerance

5.5.14

int
HYPRE_SStructMaxwellSetMaxIter (HYPRE_SStructSolver solver, int max_iter)

(Optional) Set maximum number of iterations

5.5.15

int
HYPRE_SStructMaxwellSetRelChange (HYPRE_SStructSolver solver, int rel_change)

(Optional) Additionally require that the relative difference in successive iterates be small

5.5.16

int
HYPRE_SStructMaxwellSetNumPreRelax (HYPRE_SStructSolver solver, int num_pre_relax)

(Optional) Set number of relaxation sweeps before coarse-grid correction

5.5.17

int
HYPRE_SStructMaxwellSetNumPostRelax (HYPRE_SStructSolver solver, int num_post_relax)

(Optional) Set number of relaxation sweeps after coarse-grid correction

5.5.18

int
HYPRE_SStructMaxwellSetLogging (HYPRE_SStructSolver solver, int logging)

(Optional) Set the amount of logging to do

5.5.19

int
HYPRE_SStructMaxwellGetNumIterations (HYPRE_SStructSolver solver, int* num_iterations)

Return the number of iterations taken

5.5.20

int
HYPRE_SStructMaxwellGetFinalRelativeResidualNorm
(HYPRE_SStructSolver solver, HYPRE_Real* norm)

Return the norm of the final relative residual

5.6

SStruct PCG Solver

Names

5.6.1 int
HYPRE_SStructPCGCreate (MPI_Comm comm,
HYPRE_SStructSolver* solver)
Create a solver object 121

5.6.2 int
HYPRE_SStructPCGDestroy (HYPRE_SStructSolver solver)
Destroy a solver object. 122

5.6.3 int
HYPRE_SStructDiagScaleSetup (HYPRE_SStructSolver solver,
HYPRE_SStructMatrix A,
HYPRE_SStructVector y,
HYPRE_SStructVector x)
Setup routine for diagonal preconditioning 122

5.6.4 int
HYPRE_SStructDiagScale (HYPRE_SStructSolver solver,
HYPRE_SStructMatrix A,
HYPRE_SStructVector y,
HYPRE_SStructVector x)
Solve routine for diagonal preconditioning 122

These routines should be used in conjunction with the generic interface in PCG Solver.

5.6.1

int
HYPRE_SStructPCGCreate (MPI_Comm comm, HYPRE_SStructSolver*
solver)

Create a solver object

5.6.2

int **HYPRE_SStructPCGDestroy** (HYPRE_SStructSolver solver)

Destroy a solver object. An object should be explicitly destroyed using this destructor when the user's code no longer needs direct access to it. Once destroyed, the object must not be referenced again. Note that the object may not be deallocated at the completion of this call, since there may be internal package references to the object. The object will then be destroyed when all internal reference counts go to zero.

5.6.3

int
HYPRE_SStructDiagScaleSetup (HYPRE_SStructSolver solver,
HYPRE_SStructMatrix A, HYPRE_SStructVector y, HYPRE_SStructVector x)

Setup routine for diagonal preconditioning

5.6.4

int
HYPRE_SStructDiagScale (HYPRE_SStructSolver solver,
HYPRE_SStructMatrix A, HYPRE_SStructVector y, HYPRE_SStructVector x)

Solve routine for diagonal preconditioning

5.7

SStruct GMRES Solver

Names

5.7.1 int

HYPRE_SStructGMRESCreate (MPI_Comm comm, HYPRE_SStructSolver* solver)
Create a solver object 123

5.7.2 int **HYPRE_SStructGMRESDestroy** (HYPRE_SStructSolver solver)
Destroy a solver object. 123

These routines should be used in conjunction with the generic interface in GMRES Solver.

5.7.1

int
HYPRE_SStructGMRESCreate (MPI_Comm comm, HYPRE_SStructSolver* solver)

Create a solver object

5.7.2

int **HYPRE_SStructGMRESDestroy** (HYPRE_SStructSolver solver)

Destroy a solver object. An object should be explicitly destroyed using this destructor when the user's code no longer needs direct access to it. Once destroyed, the object must not be referenced again. Note that the object may not be deallocated at the completion of this call, since there may be internal package references to the object. The object will then be destroyed when all internal reference counts go to zero.

5.8 SStruct FlexGMRES Solver

Names

5.8.1 int **HYPRE_SStructFlexGMRESCreate** (MPI_Comm comm, HYPRE_SStructSolver* solver)
Create a solver object 124

5.8.2 int

HYPRE_SStructFlexGMRESDestroy (HYPRE_SStructSolver solver)
Destroy a solver object. 124

These routines should be used in conjunction with the generic interface in FlexGMRES Solver.

5.8.1

int
HYPRE_SStructFlexGMRESCreate (MPI_Comm comm,
HYPRE_SStructSolver* solver)

Create a solver object

5.8.2

int **HYPRE_SStructFlexGMRESDestroy** (HYPRE_SStructSolver solver)

Destroy a solver object. An object should be explicitly destroyed using this destructor when the user's code no longer needs direct access to it. Once destroyed, the object must not be referenced again. Note that the object may not be deallocated at the completion of this call, since there may be internal package references to the object. The object will then be destroyed when all internal reference counts go to zero.

5.9

SStruct LGMRES Solver

Names

5.9.1 int
HYPRE_SStructLGMRESCreate (MPI_Comm comm,
HYPRE_SStructSolver* solver)
Create a solver object 125

5.9.2 int
HYPRE_SStructLGMRESDestroy (HYPRE_SStructSolver solver)
Destroy a solver object. 125

These routines should be used in conjunction with the generic interface in LGMRES Solver.

5.9.1

int
HYPRE_SStructLGMRESCreate (MPI_Comm comm, HYPRE_SStructSolver* solver)

Create a solver object

5.9.2

int **HYPRE_SStructLGMRESDestroy** (HYPRE_SStructSolver solver)

Destroy a solver object. An object should be explicitly destroyed using this destructor when the user's code no longer needs direct access to it. Once destroyed, the object must not be referenced again. Note that the object may not be deallocated at the completion of this call, since there may be internal package references to the object. The object will then be destroyed when all internal reference counts go to zero.

5.10

SStruct BiCGSTAB Solver

Names

5.10.1 int
HYPRE_SStructBiCGSTABCreate (MPI_Comm comm, HYPRE_SStructSolver* solver)
Create a solver object 126

5.10.2 int
HYPRE_SStructBiCGSTABDestroy (HYPRE_SStructSolver solver)
Destroy a solver object. 126

These routines should be used in conjunction with the generic interface in BiCGSTAB Solver.

5.10.1

int
HYPRE_SStructBiCGSTABCreate (MPI_Comm comm,
HYPRE_SStructSolver* solver)

Create a solver object

5.10.2

int **HYPRE_SStructBiCGSTABDestroy** (HYPRE_SStructSolver solver)

Destroy a solver object. An object should be explicitly destroyed using this destructor when the user's code no longer needs direct access to it. Once destroyed, the object must not be referenced again. Note that the object may not be deallocated at the completion of this call, since there may be internal package references to the object. The object will then be destroyed when all internal reference counts go to zero.

5.11 SStruct LOBPCG Eigensolver

Names

5.11.1 int
HYPRE_SStructSetupInterpreter (mv_InterfaceInterpreter* i)
Load interface interpreter. 126

5.11.2 int
HYPRE_SStructSetupMatvec (HYPRE_MatvecFunctions* mv)
Load Matvec interpreter with hypre_SStructKrylov functions 127

These routines should be used in conjunction with the generic interface in LOBPCG Eigensolver.

5.11.1

int **HYPRE_SStructSetupInterpreter** (mv_InterfaceInterpreter* i)

Load interface interpreter. Vector part loaded with hypre_SStructKrylov functions and multivector part loaded with mv_TempMultiVector functions.

5.11.2

int **HYPRE_SStructSetupMatvec** (HYPRE_MatvecFunctions* mv)

Load Matvec interpreter with hypre_SStructKrylov functions

6

ParCSR Solvers

Names

6.1 **ParCSR Solvers** 129

6.2 **ParCSR BoomerAMG Solver and Preconditioner** 129

6.3 **ParCSR ParaSails Preconditioner** 163

6.4 **ParCSR Euclid Preconditioner** 168

6.5 **ParCSR Pilut Preconditioner** 173

6.6 **ParCSR AMS Solver and Preconditioner** 175

6.7 **ParCSR ADS Solver and Preconditioner** 186

6.8 **ParCSR PCG Solver** 193

6.9 **ParCSR GMRES Solver** 195

6.10 **ParCSR FlexGMRES Solver** 196

6.11 **ParCSR LGMRES Solver** 197

6.12 **ParCSR BiCGSTAB Solver** 197

6.13 **ParCSR Hybrid Solver** 198

6.14 **ParCSR LOBPCG Eigensolver** 215

These solvers use matrix/vector storage schemes that are taylored for general sparse matrix systems.

6.1

ParCSR Solvers

Names

6.1.1 #define **HYPRE_SOLVER_STRUCT**
The solver object 129

6.1.1

#define **HYPRE_SOLVER_STRUCT**

The solver object

6.2

ParCSR BoomerAMG Solver and Preconditioner

Names

6.2.1 int
HYPRE_BoomerAMGCreate (HYPRE_Solver* solver)
Create a solver object 137

6.2.2 int
HYPRE_BoomerAMGDestroy (HYPRE_Solver solver)
Destroy a solver object 138

6.2.3 int
HYPRE_BoomerAMGSetup (HYPRE_Solver solver,
HYPRE_ParCSRMatrix A,
HYPRE_ParVector b, HYPRE_ParVector x)
Set up the BoomerAMG solver or preconditioner. 138

6.2.4 int
HYPRE_BoomerAMGSolve (HYPRE_Solver solver,
HYPRE_ParCSRMatrix A,
HYPRE_ParVector b, HYPRE_ParVector x)
Solve the system or apply AMG as a preconditioner. 138

6.2.5 int

HYPRE_BoomerAMGSolveT (HYPRE_Solver solver, HYPRE_ParCSRMatrix A, HYPRE_ParVector b, HYPRE_ParVector x)
Solve the transpose system $A^T x = b$ or apply AMG as a preconditioner to the transpose system 139

6.2.6 int
HYPRE_BoomerAMGGetResidual (HYPRE_Solver solver, HYPRE_ParVector* residual)
Returns the residual 139

6.2.7 int
HYPRE_BoomerAMGGetNumIterations (HYPRE_Solver solver, int* num_iterations)
Returns the number of iterations taken 139

6.2.8 int
HYPRE_BoomerAMGGetFinalRelativeResidualNorm (HYPRE_Solver solver, HYPRE_Real* rel_resid_norm)
Returns the norm of the final relative residual 140

6.2.9 int
HYPRE_BoomerAMGSetNumFunctions (HYPRE_Solver solver, int num_functions)
(Optional) Sets the size of the system of PDEs, if using the systems version. 140

6.2.10 int
HYPRE_BoomerAMGSetDofFunc (HYPRE_Solver solver, int* dof_func)
(Optional) Sets the mapping that assigns the function to each variable, if using the systems version. 140

6.2.11 int
HYPRE_BoomerAMGSetTol (HYPRE_Solver solver, HYPRE_Real tol)
(Optional) Set the convergence tolerance, if BoomerAMG is used as a solver. 140

6.2.12 int
HYPRE_BoomerAMGSetMaxIter (HYPRE_Solver solver, int max_iter)
(Optional) Sets maximum number of iterations, if BoomerAMG is used as a solver. 141

6.2.13 int
HYPRE_BoomerAMGSetMinIter (HYPRE_Solver solver, int min_iter)
(Optional) 141

6.2.14 int
HYPRE_BoomerAMGSetMaxCoarseSize (HYPRE_Solver solver, int max_coarse_size)
(Optional) Sets maximum size of coarsest grid. 141

6.2.15 int
HYPRE_BoomerAMGSetMinCoarseSize (HYPRE_Solver solver, int min_coarse_size)
(Optional) Sets minimum size of coarsest grid. 141

6.2.16 int

HYPRE_BoomerAMGSetMaxLevels (HYPRE_Solver solver, int max_levels)
(Optional) Sets maximum number of multigrid levels. 142

6.2.17 int
HYPRE_BoomerAMGSetStrongThreshold (HYPRE_Solver solver, HYPRE_Real strong_threshold)
(Optional) Sets AMG strength threshold. 142

6.2.18 int
HYPRE_BoomerAMGSetSCommPkgSwitch (HYPRE_Solver solver, HYPRE_Real S_commpkg_switch)
(Optional) Defines the largest strength threshold for which the strength matrix S uses the communication package of the operator A. 142

6.2.19 int
HYPRE_BoomerAMGSetMaxRowSum (HYPRE_Solver solver, HYPRE_Real max_row_sum)
(Optional) Sets a parameter to modify the definition of strength for diagonal dominant portions of the matrix. 142

6.2.20 int
HYPRE_BoomerAMGSetCoarsenType (HYPRE_Solver solver, int coarsen_type)
(Optional) Defines which parallel coarsening algorithm is used. 143

6.2.21 int
HYPRE_BoomerAMGSetNonGalerkinTol (HYPRE_Solver solver, HYPRE_Real nongalerkin_tol)
(Optional) Defines the non-Galerkin drop-tolerance for sparsifying coarse grid operators and thus reducing communication. 143

6.2.22 int
HYPRE_BoomerAMGSetLevelNonGalerkinTol (HYPRE_Solver solver, HYPRE_Real nongalerkin_tol, int level)
(Optional) Defines the level specific non-Galerkin drop-tolerances for sparsifying coarse grid operators and thus reducing communication. 144

6.2.23 int
HYPRE_BoomerAMGSetMeasureType (HYPRE_Solver solver, int measure_type)
(Optional) Defines whether local or global measures are used 144

6.2.24 int
HYPRE_BoomerAMGSetAggNumLevels (HYPRE_Solver solver, int agg_num_levels)
(Optional) Defines the number of levels of aggressive coarsening. 144

6.2.25 int
HYPRE_BoomerAMGSetNumPaths (HYPRE_Solver solver, int num_paths)
(Optional) Defines the degree of aggressive coarsening. 145

6.2.26 int
HYPRE_BoomerAMGSetCGCIts (HYPRE_Solver solver, int its)
(optional) Defines the number of pathes for CGC-coarsening 145

6.2.27 int

HYPRE_BoomerAMGSetNodal (HYPRE_Solver solver, int nodal)
(Optional) Sets whether to use the nodal systems coarsening. 145

6.2.28 int
HYPRE_BoomerAMGSetNodalDiag (HYPRE_Solver solver, int nodal_diag)
(Optional) Sets whether to give special treatment to diagonal elements in the nodal systems version. 145

6.2.29 int
HYPRE_BoomerAMGSetInterpType (HYPRE_Solver solver, int interp_type)
(Optional) Defines which parallel interpolation operator is used. 146

6.2.30 int
HYPRE_BoomerAMGSetTruncFactor (HYPRE_Solver solver, HYPRE_Real trunc_factor)
(Optional) Defines a truncation factor for the interpolation. 146

6.2.31 int
HYPRE_BoomerAMGSetPMaxElmts (HYPRE_Solver solver, int P_max_elmts)
(Optional) Defines the maximal number of elements per row for the interpolation. 147

6.2.32 int
HYPRE_BoomerAMGSetSepWeight (HYPRE_Solver solver, int sep_weight)
(Optional) Defines whether separation of weights is used when defining strength for standard interpolation or multipass interpolation. 147

6.2.33 int
HYPRE_BoomerAMGSetAggInterpType (HYPRE_Solver solver, int agg_interp_type)
(Optional) Defines the interpolation used on levels of aggressive coarsening The default is 4, ie. 147

6.2.34 int
HYPRE_BoomerAMGSetAggTruncFactor (HYPRE_Solver solver, HYPRE_Real agg_trunc_factor)
(Optional) Defines the truncation factor for the interpolation used for aggressive coarsening. 147

6.2.35 int
HYPRE_BoomerAMGSetAggP12TruncFactor (HYPRE_Solver solver, HYPRE_Real agg_P12_trunc_factor)
(Optional) Defines the truncation factor for the matrices P1 and P2 which are used to build 2-stage interpolation. 148

6.2.36 int
HYPRE_BoomerAMGSetAggPMaxElmts (HYPRE_Solver solver, int agg_P_max_elmts)
(Optional) Defines the maximal number of elements per row for the interpolation used for aggressive coarsening. 148

6.2.37 int

HYPRE_BoomerAMGSetAggP12MaxElmts (HYPRE_Solver solver, int agg_P12_max_elmts)
(Optional) Defines the maximal number of elements per row for the matrices P1 and P2 which are used to build 2-stage interpolation. 148

6.2.38 int
HYPRE_BoomerAMGSetInterpVectors (HYPRE_Solver solver, int num_vectors, HYPRE_ParVector* interp_vectors)
(Optional) Allows the user to incorporate additional vectors into the interpolation for systems AMG, eg. 149

6.2.39 int
HYPRE_BoomerAMGSetInterpVecVariant (HYPRE_Solver solver, int var)
(Optional) Defines the interpolation variant used for HYPRE_BoomerAMGSetInterpVectors:

1	*GM approach 1*
2	*GM approach 2 (to be preferred over 1)*
3	*LN approach*

.... 149

6.2.40 int
HYPRE_BoomerAMGSetInterpVecQMax (HYPRE_Solver solver, int q_max)
(Optional) Defines the maximal elements per row for Q, the additional columns added to the original interpolation matrix P, to reduce complexity. 149

6.2.41 int
HYPRE_BoomerAMGSetInterpVecAbsQTrunc (HYPRE_Solver solver, HYPRE_Real q_trunc)
(Optional) Defines a truncation factor for Q, the additional columns added to the original interpolation matrix P, to reduce complexity. 149

6.2.42 int
HYPRE_BoomerAMGSetGSMG (HYPRE_Solver solver, int gsmg)
(Optional) Specifies the use of GSMG - geometrically smooth coarsening and interpolation. 150

6.2.43 int
HYPRE_BoomerAMGSetNumSamples (HYPRE_Solver solver, int num_samples)
(Optional) Defines the number of sample vectors used in GSMG or LS interpolation 150

6.2.44 int
HYPRE_BoomerAMGSetCycleType (HYPRE_Solver solver, int cycle_type)
(Optional) Defines the type of cycle. 150

6.2.45 int
HYPRE_BoomerAMGSetAdditive (HYPRE_Solver solver, int addlvl)
(Optional) Defines use of an additive V(1, 1)-cycle using the classical additive method starting at level 'addlvl'. 150

6.2.46 int

HYPRE_BoomerAMGSetMultAdditive (HYPRE_Solver solver, int addlvl)
(Optional) Defines use of an additive V(1, 1)-cycle using the mult-additive method starting at level 'addlvl'. 151

6.2.47 int
HYPRE_BoomerAMGSetSimple (HYPRE_Solver solver, int addlvl)
(Optional) Defines use of an additive V(1, 1)-cycle using the simplified mult-additive method starting at level 'addlvl'. 151

6.2.48 int
HYPRE_BoomerAMGSetMultAddTruncFactor (HYPRE_Solver solver, HYPRE_Real add_trunc_factor)
(Optional) Defines the truncation factor for the smoothed interpolation used for mult-additive or simple method. 151

6.2.49 int
HYPRE_BoomerAMGSetMultAddPMaxElmts (HYPRE_Solver solver, int add_P_max_elmts)
(Optional) Defines the maximal number of elements per row for the smoothed interpolation used for mult-additive or simple method. 151

6.2.50 int
HYPRE_BoomerAMGSetSeqThreshold (HYPRE_Solver solver, int seq_threshold)
(Optional) Sets maximal size for agglomeration or redundant coarse grid solve. 152

6.2.51 int
HYPRE_BoomerAMGSetRedundant (HYPRE_Solver solver, int redundant)
(Optional) operates switch for redundancy. 152

6.2.52 int
HYPRE_BoomerAMGSetNumSweeps (HYPRE_Solver solver, int num_sweeps)
(Optional) Sets the number of sweeps. 152

6.2.53 int
HYPRE_BoomerAMGSetCycleNumSweeps (HYPRE_Solver solver, int num_sweeps, int k)
(Optional) Sets the number of sweeps at a specified cycle. 152

6.2.54 int
HYPRE_BoomerAMGSetGridRelaxType (HYPRE_Solver solver, int* grid_relax_type)
(Optional) Defines which smoother is used on the fine and coarse grid, the up and down cycle. 153

6.2.55 int
HYPRE_BoomerAMGSetRelaxType (HYPRE_Solver solver, int relax_type)
(Optional) Defines the smoother to be used. 153

6.2.56 int
HYPRE_BoomerAMGSetCycleRelaxType (HYPRE_Solver solver, int relax_type, int k)
(Optional) Defines the smoother at a given cycle. 153

6.2.57 int

HYPRE_BoomerAMGSetRelaxOrder (HYPRE_Solver solver,
int relax_order)
(Optional) Defines in which order the points are relaxed. 154

6.2.58 int
HYPRE_BoomerAMGSetRelaxWt (HYPRE_Solver solver,
HYPRE_Real relax_weight)
(Optional) Defines the relaxation weight for smoothed Jacobi and hybrid SOR on all levels. 154

6.2.59 int
HYPRE_BoomerAMGSetLevelRelaxWt (HYPRE_Solver solver,
HYPRE_Real relax_weight,
int level)
(Optional) Defines the relaxation weight for smoothed Jacobi and hybrid SOR on the user defined level. 155

6.2.60 int
HYPRE_BoomerAMGSetOmega (HYPRE_Solver solver,
HYPRE_Real* omega)
(Optional) Defines the outer relaxation weight for hybrid SOR. 155

6.2.61 int
HYPRE_BoomerAMGSetOuterWt (HYPRE_Solver solver,
HYPRE_Real omega)
(Optional) Defines the outer relaxation weight for hybrid SOR and SSOR on all levels. 155

6.2.62 int
HYPRE_BoomerAMGSetLevelOuterWt (HYPRE_Solver solver,
HYPRE_Real omega, int level)
(Optional) Defines the outer relaxation weight for hybrid SOR or SSOR on the user defined level. 156

6.2.63 int
HYPRE_BoomerAMGSetChebyOrder (HYPRE_Solver solver, int order)
(Optional) Defines the Order for Chebyshev smoother. 156

6.2.64 int
HYPRE_BoomerAMGSetChebyFraction (HYPRE_Solver solver,
HYPRE_Real ratio)
(Optional) Fraction of the spectrum to use for the Chebyshev smoother. . 156

6.2.65 int
HYPRE_BoomerAMGSetSmoothType (HYPRE_Solver solver,
int smooth_type)
(Optional) Enables the use of more complex smoothers. 156

6.2.66 int
HYPRE_BoomerAMGSetSmoothNumLevels (HYPRE_Solver solver,
int smooth_num_levels)
(Optional) Sets the number of levels for more complex smoothers. 157

6.2.67 int
HYPRE_BoomerAMGSetSmoothNumSweeps (HYPRE_Solver solver,
int smooth_num_sweeps)
(Optional) Sets the number of sweeps for more complex smoothers. 157

6.2.68 int

HYPRE_BoomerAMGSetVariant (HYPRE_Solver solver, int variant)
(Optional) Defines which variant of the Schwarz method is used. 157

6.2.69 int
HYPRE_BoomerAMGSetOverlap (HYPRE_Solver solver, int overlap)
(Optional) Defines the overlap for the Schwarz method. 158

6.2.70 int
HYPRE_BoomerAMGSetDomainType (HYPRE_Solver solver, int domain_type)
(Optional) Defines the type of domain used for the Schwarz method. 158

6.2.71 int
HYPRE_BoomerAMGSetSchwarzRlxWeight (HYPRE_Solver solver, HYPRE_Real schwarz_rlx_weight)
(Optional) Defines a smoothing parameter for the additive Schwarz method 158

6.2.72 int
HYPRE_BoomerAMGSetSchwarzUseNonSymm (HYPRE_Solver solver, int use_nonsymm)
(Optional) Indicates that the aggregates may not be SPD for the Schwarz method. 159

6.2.73 int
HYPRE_BoomerAMGSetSym (HYPRE_Solver solver, int sym)
(Optional) Defines symmetry for ParaSAILS. 159

6.2.74 int
HYPRE_BoomerAMGSetLevel (HYPRE_Solver solver, int level)
(Optional) Defines number of levels for ParaSAILS. 159

6.2.75 int
HYPRE_BoomerAMGSetThreshold (HYPRE_Solver solver, HYPRE_Real threshold)
(Optional) Defines threshold for ParaSAILS. 159

6.2.76 int
HYPRE_BoomerAMGSetFilter (HYPRE_Solver solver, HYPRE_Real filter)
(Optional) Defines filter for ParaSAILS. 160

6.2.77 int
HYPRE_BoomerAMGSetDropTol (HYPRE_Solver solver, HYPRE_Real drop_tol)
(Optional) Defines drop tolerance for PILUT. 160

6.2.78 int
HYPRE_BoomerAMGSetMaxNzPerRow (HYPRE_Solver solver, int max_nz_per_row)
(Optional) Defines maximal number of nonzeros for PILUT. 160

6.2.79 int
HYPRE_BoomerAMGSetEuclidFile (HYPRE_Solver solver, char* euclidfile)
(Optional) Defines name of an input file for Euclid parameters. 160

6.2.80 int
HYPRE_BoomerAMGSetEuLevel (HYPRE_Solver solver, int eu_level)
(Optional) Defines number of levels for ILU(k) in Euclid. 161

6.2.81 int

HYPRE_BoomerAMGSetEuSparseA (HYPRE_Solver solver, HYPRE_Real eu_sparse_A)
(Optional) Defines filter for ILU(k) for Euclid. 161

6.2.82 int
HYPRE_BoomerAMGSetEuBJ (HYPRE_Solver solver, int eu_bj)
(Optional) Defines use of block jacobi ILUT for Euclid. 161

6.2.83 int
HYPRE_BoomerAMGSetPrintLevel (HYPRE_Solver solver, int print_level)
(Optional) Requests automatic printing of setup and solve information. .. 161

6.2.84 int
HYPRE_BoomerAMGSetLogging (HYPRE_Solver solver, int logging)
(Optional) Requests additional computations for diagnostic and similar data to be logged by the user. 162

6.2.85 int
HYPRE_BoomerAMGSetDebugFlag (HYPRE_Solver solver, int debug_flag)
(Optional) 162

6.2.86 int
HYPRE_BoomerAMGInitGridRelaxation (int** num_grid_sweeps_ptr, int** grid_relax_type_ptr, int*** grid_relax_points_ptr, int coarsen_type, HYPRE_Real** relax_weights_ptr, int max_levels)
(Optional) This routine will be eliminated in the future 162

6.2.87 int
HYPRE_BoomerAMGSetRAP2 (HYPRE_Solver solver, int rap2)
(Optional) If rap2 not equal 0, the triple matrix product RAP is replaced by two matrix products 162

6.2.88 int
HYPRE_BoomerAMGSetKeepTranspose (HYPRE_Solver solver, int keepTranspose)
(Optional) If set to 1, the local interpolation transposes will be saved to use more efficient matvecs instead of matvecTs 163

Parallel unstructured algebraic multigrid solver and preconditioner

6.2.1

int **HYPRE_BoomerAMGCreate** (HYPRE_Solver* solver)

Create a solver object

6.2.2

int **HYPRE_BoomerAMGDestroy** (HYPRE_Solver solver)

Destroy a solver object

6.2.3

int
HYPRE_BoomerAMGSetup (HYPRE_Solver solver, HYPRE_ParCSRMatrix A, HYPRE_ParVector b, HYPRE_ParVector x)

Set up the BoomerAMG solver or preconditioner. If used as a preconditioner, this function should be passed to the iterative solver `SetPrecond` function.

Parameters:

`solver`	[IN] object to be set up.
`A`	[IN] ParCSR matrix used to construct the solver/preconditioner.
`b`	Ignored by this function.
`x`	Ignored by this function.

6.2.4

int
HYPRE_BoomerAMGSolve (HYPRE_Solver solver, HYPRE_ParCSRMatrix A, HYPRE_ParVector b, HYPRE_ParVector x)

Solve the system or apply AMG as a preconditioner. If used as a preconditioner, this function should be passed to the iterative solver `SetPrecond` function.

Parameters:

`solver`	[IN] solver or preconditioner object to be applied.
`A`	[IN] ParCSR matrix, matrix of the linear system to be solved
`b`	[IN] right hand side of the linear system to be solved
`x`	[OUT] approximated solution of the linear system to be solved

6.2.5

int
HYPRE_BoomerAMGSolveT (HYPRE_Solver solver, HYPRE_ParCSRMatrix A, HYPRE_ParVector b, HYPRE_ParVector x)

Solve the transpose system $A^T x = b$ or apply AMG as a preconditioner to the transpose system . Note that this function should only be used when preconditioning CGNR with BoomerAMG. It can only be used with Jacobi smoothing (relax_type 0 or 7) and without CF smoothing, i.e relax_order needs to be set to 0. If used as a preconditioner, this function should be passed to the iterative solver `SetPrecond` function.

Parameters:

`solver`	[IN] solver or preconditioner object to be applied.
`A`	[IN] ParCSR matrix
`b`	[IN] right hand side of the linear system to be solved
`x`	[OUT] approximated solution of the linear system to be solved

6.2.6

int
HYPRE_BoomerAMGGetResidual (HYPRE_Solver solver, HYPRE_ParVector* residual)

Returns the residual

6.2.7

int
HYPRE_BoomerAMGGetNumIterations (HYPRE_Solver solver, int* num_iterations)

Returns the number of iterations taken

6.2.8

int
HYPRE_BoomerAMGGetFinalRelativeResidualNorm (HYPRE_Solver solver, HYPRE_Real* rel_resid_norm)

Returns the norm of the final relative residual

6.2.9

int
HYPRE_BoomerAMGSetNumFunctions (HYPRE_Solver solver, int num_functions)

(Optional) Sets the size of the system of PDEs, if using the systems version. The default is 1, i.e. a scalar system.

6.2.10

int **HYPRE_BoomerAMGSetDofFunc** (HYPRE_Solver solver, int* dof_func)

(Optional) Sets the mapping that assigns the function to each variable, if using the systems version. If no assignment is made and the number of functions is k > 1, the mapping generated is (0,1,...,k-1,0,1,...,k-1,...).

6.2.11

int **HYPRE_BoomerAMGSetTol** (HYPRE_Solver solver, HYPRE_Real tol)

(Optional) Set the convergence tolerance, if BoomerAMG is used as a solver. If it is used as a preconditioner, it should be set to 0. The default is 1.e-7.

6.2.12

int **HYPRE_BoomerAMGSetMaxIter** (HYPRE_Solver solver, int max_iter)

(Optional) Sets maximum number of iterations, if BoomerAMG is used as a solver. If it is used as a preconditioner, it should be set to 1. The default is 20.

6.2.13

int **HYPRE_BoomerAMGSetMinIter** (HYPRE_Solver solver, int min_iter)

(Optional)

6.2.14

int
HYPRE_BoomerAMGSetMaxCoarseSize (HYPRE_Solver solver, int max_coarse_size)

(Optional) Sets maximum size of coarsest grid. The default is 9.

6.2.15

int
HYPRE_BoomerAMGSetMinCoarseSize (HYPRE_Solver solver, int min_coarse_size)

(Optional) Sets minimum size of coarsest grid. The default is 1.

6.2.16

int
HYPRE_BoomerAMGSetMaxLevels (HYPRE_Solver solver, int max_levels)

(Optional) Sets maximum number of multigrid levels. The default is 25.

6.2.17

int
HYPRE_BoomerAMGSetStrongThreshold (HYPRE_Solver solver, HYPRE_Real strong_threshold)

(Optional) Sets AMG strength threshold. The default is 0.25. For 2d Laplace operators, 0.25 is a good value, for 3d Laplace operators, 0.5 or 0.6 is a better value. For elasticity problems, a large strength threshold, such as 0.9, is often better.

6.2.18

int
HYPRE_BoomerAMGSetSCommPkgSwitch (HYPRE_Solver solver, HYPRE_Real S_commpkg_switch)

(Optional) Defines the largest strength threshold for which the strength matrix S uses the communication package of the operator A. If the strength threshold is larger than this values, a communication package is generated for S. This can save memory and decrease the amount of data that needs to be communicated, if S is substantially sparser than A. The default is 1.0.

6.2.19

int
HYPRE_BoomerAMGSetMaxRowSum (HYPRE_Solver solver, HYPRE_Real max_row_sum)

(Optional) Sets a parameter to modify the definition of strength for diagonal dominant portions of the matrix. The default is 0.9. If max_row_sum is 1, no checking for diagonally dominant rows is performed.

6.2.20

int
HYPRE_BoomerAMGSetCoarsenType (HYPRE_Solver solver, int coarsen_type)

(Optional) Defines which parallel coarsening algorithm is used. There are the following options for coarsen_type:

0	CLJP-coarsening (a parallel coarsening algorithm using independent sets.
1	classical Ruge-Stueben coarsening on each processor, no boundary treatment (not recommended!)
3	classical Ruge-Stueben coarsening on each processor, followed by a third pass, which adds coarse points on the boundaries
6	Falgout coarsening (uses 1 first, followed by CLJP using the interior coarse points generated by 1 as its first independent set)
7	CLJP-coarsening (using a fixed random vector, for debugging purposes only)
8	PMIS-coarsening (a parallel coarsening algorithm using independent sets, generating lower complexities than CLJP, might also lead to slower convergence)
9	PMIS-coarsening (using a fixed random vector, for debugging purposes only)
10	HMIS-coarsening (uses one pass Ruge-Stueben on each processor independently, followed by PMIS using the interior C-points generated as its first independent set)
11	one-pass Ruge-Stueben coarsening on each processor, no boundary treatment (not recommended!)
21	CGC coarsening by M. Griebel, B. Metsch and A. Schweitzer
22	CGC-E coarsening by M. Griebel, B. Metsch and A.Schweitzer

The default is 6.

6.2.21

int
HYPRE_BoomerAMGSetNonGalerkinTol (HYPRE_Solver solver, HYPRE_Real nongalerkin_tol)

(Optional) Defines the non-Galerkin drop-tolerance for sparsifying coarse grid operators and thus reducing communication. Value specified here is set on all levels. This routine should be used before HYPRE_BoomerAMGSetLevelNonGalerkinTol, which then can be used to change individual levels if desired

6.2.22

int
HYPRE_BoomerAMGSetLevelNonGalerkinTol (HYPRE_Solver solver, HYPRE_Real nongalerkin_tol, int level)

(Optional) Defines the level specific non-Galerkin drop-tolerances for sparsifying coarse grid operators and thus reducing communication. A drop-tolerance of 0.0 means to skip doing non-Galerkin on that level. The maximum drop tolerance for a level is 1.0, although much smaller values such as 0.03 or 0.01 are recommended.

Note that if the user wants to set a specific tolerance on all levels, HYPRE_BooemrAMGSetNonGalerkinTol should be used. Individual levels can then be changed using this routine.

In general, it is safer to drop more aggressively on coarser levels. For instance, one could use 0.0 on the finest level, 0.01 on the second level and then using 0.05 on all remaining levels. The best way to achieve this is to set 0.05 on all levels with HYPRE_BoomerAMGSetNonGalerkinTol and then change the tolerance on level 0 to 0.0 and the tolerance on level 1 to 0.01 with HYPRE_BoomerAMGSetLevelNonGalerkinTol. Like many AMG parameters, these drop tolerances can be tuned. It is also common to delay the start of the non-Galerkin process further to a later level than level 1.

Parameters:

	`solver`	[IN] solver or preconditioner object to be applied.
	`nongalerkin_tol`	[IN] level specific drop tolerance
	`level`	[IN] level on which drop tolerance is used

6.2.23

int
HYPRE_BoomerAMGSetMeasureType (HYPRE_Solver solver, int measure_type)

(Optional) Defines whether local or global measures are used

6.2.24

int
HYPRE_BoomerAMGSetAggNumLevels (HYPRE_Solver solver, int agg_num_levels)

(Optional) Defines the number of levels of aggressive coarsening. The default is 0, i.e. no aggressive coarsening.

6.2.25

int
HYPRE_BoomerAMGSetNumPaths (HYPRE_Solver solver, int num_paths)

(Optional) Defines the degree of aggressive coarsening. The default is 1. Larger numbers lead to less aggressive coarsening.

6.2.26

int **HYPRE_BoomerAMGSetCGCIts** (HYPRE_Solver solver, int its)

(optional) Defines the number of pathes for CGC-coarsening

6.2.27

int **HYPRE_BoomerAMGSetNodal** (HYPRE_Solver solver, int nodal)

(Optional) Sets whether to use the nodal systems coarsening. Should be used for linear systems generated from systems of PDEs. The default is 0 (unknown-based coarsening, only coarsens within same function). For the remaining options a nodal matrix is generated by applying a norm to the nodal blocks and applying the coarsening algorithm to this matrix.

1	Frobenius norm
2	sum of absolute values of elements in each block
3	largest element in each block (not absolute value)
4	row-sum norm
6	sum of all values in each block

6.2.28

int
HYPRE_BoomerAMGSetNodalDiag (HYPRE_Solver solver, int nodal_diag)

(Optional) Sets whether to give special treatment to diagonal elements in the nodal systems version. The default is 0. If set to 1, the diagonal entry is set to the negative sum of all off diagonal entries. If set to 2, the signs of all diagonal entries are inverted.

6.2.29

int
HYPRE_BoomerAMGSetInterpType (HYPRE_Solver solver, int interp_type)

(Optional) Defines which parallel interpolation operator is used. There are the following options for interp_type:

0	classical modified interpolation
1	LS interpolation (for use with GSMG)
2	classical modified interpolation for hyperbolic PDEs
3	direct interpolation (with separation of weights)
4	multipass interpolation
5	multipass interpolation (with separation of weights)
6	extended+i interpolation
7	extended+i (if no common C neighbor) interpolation
8	standard interpolation
9	standard interpolation (with separation of weights)
10	classical block interpolation (for use with nodal systems version only)
11	classical block interpolation (for use with nodal systems version only) with diagonalized diagonal blocks
12	FF interpolation
13	FF1 interpolation
14	extended interpolation

The default is 0.

6.2.30

int
HYPRE_BoomerAMGSetTruncFactor (HYPRE_Solver solver, HYPRE_Real trunc_factor)

(Optional) Defines a truncation factor for the interpolation. The default is 0.

6.2.31

int
HYPRE_BoomerAMGSetPMaxElmts (HYPRE_Solver solver, int P_max_elmts)

(Optional) Defines the maximal number of elements per row for the interpolation. The default is 0.

6.2.32

int
HYPRE_BoomerAMGSetSepWeight (HYPRE_Solver solver, int sep_weight)

(Optional) Defines whether separation of weights is used when defining strength for standard interpolation or multipass interpolation. Default: 0, i.e. no separation of weights used.

6.2.33

int
HYPRE_BoomerAMGSetAggInterpType (HYPRE_Solver solver, int agg_interp_type)

(Optional) Defines the interpolation used on levels of aggressive coarsening The default is 4, ie. multipass interpolation. The following options exist:

1	2-stage extended+i interpolation
2	2-stage standard interpolation
3	2-stage extended interpolation
4	multipass interpolation

6.2.34

int
HYPRE_BoomerAMGSetAggTruncFactor (HYPRE_Solver solver, HYPRE_Real agg_trunc_factor)

(Optional) Defines the truncation factor for the interpolation used for aggressive coarsening. The default is 0.

6.2.35

int
HYPRE_BoomerAMGSetAggP12TruncFactor (HYPRE_Solver solver, HYPRE_Real agg_P12_trunc_factor)

(Optional) Defines the truncation factor for the matrices P1 and P2 which are used to build 2-stage interpolation. The default is 0.

6.2.36

int
HYPRE_BoomerAMGSetAggPMaxElmts (HYPRE_Solver solver, int agg_P_max_elmts)

(Optional) Defines the maximal number of elements per row for the interpolation used for aggressive coarsening. The default is 0.

6.2.37

int
HYPRE_BoomerAMGSetAggP12MaxElmts (HYPRE_Solver solver, int agg_P12_max_elmts)

(Optional) Defines the maximal number of elements per row for the matrices P1 and P2 which are used to build 2-stage interpolation. The default is 0.

6.2.38

int
HYPRE_BoomerAMGSetInterpVectors (HYPRE_Solver solver, int num_vectors, HYPRE_ParVector* interp_vectors)

(Optional) Allows the user to incorporate additional vectors into the interpolation for systems AMG, eg. rigid body modes for linear elasticity problems. This can only be used in context with nodal coarsening and still requires the user to choose an interpolation.

6.2.39

int
HYPRE_BoomerAMGSetInterpVecVariant (HYPRE_Solver solver, int var)

(Optional) Defines the interpolation variant used for HYPRE_BoomerAMGSetInterpVectors:

1	GM approach 1
2	GM approach 2 (to be preferred over 1)
3	LN approach

6.2.40

int
HYPRE_BoomerAMGSetInterpVecQMax (HYPRE_Solver solver, int q_max)

(Optional) Defines the maximal elements per row for Q, the additional columns added to the original interpolation matrix P, to reduce complexity. The default is no truncation.

6.2.41

int
HYPRE_BoomerAMGSetInterpVecAbsQTrunc (HYPRE_Solver solver, HYPRE_Real q_trunc)

(Optional) Defines a truncation factor for Q, the additional columns added to the original interpolation matrix P, to reduce complexity. The default is no truncation.

6.2.42

int **HYPRE_BoomerAMGSetGSMG** (HYPRE_Solver solver, int gsmg)

(Optional) Specifies the use of GSMG - geometrically smooth coarsening and interpolation. Currently any nonzero value for gsmg will lead to the use of GSMG. The default is 0, i.e. (GSMG is not used)

6.2.43

int
HYPRE_BoomerAMGSetNumSamples (HYPRE_Solver solver, int num_samples)

(Optional) Defines the number of sample vectors used in GSMG or LS interpolation

6.2.44

int
HYPRE_BoomerAMGSetCycleType (HYPRE_Solver solver, int cycle_type)

(Optional) Defines the type of cycle. For a V-cycle, set cycle_type to 1, for a W-cycle set cycle_type to 2. The default is 1.

6.2.45

int **HYPRE_BoomerAMGSetAdditive** (HYPRE_Solver solver, int addlvl)

(Optional) Defines use of an additive V(1,1)-cycle using the classical additive method starting at level 'addlvl'. The multiplicative approach is used on levels 0, ...'addlvl+1'. 'addlvl' needs to be > -1 for this to have an effect. Can only be used with weighted Jacobi and l1-Jacobi(default).

Can only be used when AMG is used as a preconditioner !!!

6.2.46

int **HYPRE_BoomerAMGSetMultAdditive** (HYPRE_Solver solver, int addlvl)

(Optional) Defines use of an additive V(1,1)-cycle using the mult-additive method starting at level 'addlvl'. The multiplicative approach is used on levels 0, ...'addlvl+1'. 'addlvl' needs to be > -1 for this to have an effect. Can only be used with weighted Jacobi and l1-Jacobi(default).

Can only be used when AMG is used as a preconditioner !!!

6.2.47

int **HYPRE_BoomerAMGSetSimple** (HYPRE_Solver solver, int addlvl)

(Optional) Defines use of an additive V(1,1)-cycle using the simplified mult-additive method starting at level 'addlvl'. The multiplicative approach is used on levels 0, ...'addlvl+1'. 'addlvl' needs to be > -1 for this to have an effect. Can only be used with weighted Jacobi and l1-Jacobi(default).

Can only be used when AMG is used as a preconditioner !!!

6.2.48

int
HYPRE_BoomerAMGSetMultAddTruncFactor (HYPRE_Solver solver, HYPRE_Real add_trunc_factor)

(Optional) Defines the truncation factor for the smoothed interpolation used for mult-additive or simple method. The default is 0.

6.2.49

int
HYPRE_BoomerAMGSetMultAddPMaxElmts (HYPRE_Solver solver, int add_P_max_elmts)

(Optional) Defines the maximal number of elements per row for the smoothed interpolation used for mult-additive or simple method. The default is 0.

6.2.50

int
HYPRE_BoomerAMGSetSeqThreshold (HYPRE_Solver solver, int seq_threshold)

(Optional) Sets maximal size for agglomeration or redundant coarse grid solve. When the system is smaller than this threshold, sequential AMG is used on process 0 or on all remaining active processes (if redundant = 1).

6.2.51

int
HYPRE_BoomerAMGSetRedundant (HYPRE_Solver solver, int redundant)

(Optional) operates switch for redundancy. Needs to be used with HYPRE_BoomerAMGSetSeqThreshold. Default is 0, i.e. no redundancy.

6.2.52

int
HYPRE_BoomerAMGSetNumSweeps (HYPRE_Solver solver, int num_sweeps)

(Optional) Sets the number of sweeps. On the finest level, the up and the down cycle the number of sweeps are set to num_sweeps and on the coarsest level to 1. The default is 1.

6.2.53

int
HYPRE_BoomerAMGSetCycleNumSweeps (HYPRE_Solver solver, int num_sweeps, int k)

(Optional) Sets the number of sweeps at a specified cycle. There are the following options for k:

the down cycle	if k=1
the up cycle	if k=2
the coarsest level	if k=3.

6.2.54

int
HYPRE_BoomerAMGSetGridRelaxType (HYPRE_Solver solver, int* grid_relax_type)

(Optional) Defines which smoother is used on the fine and coarse grid, the up and down cycle.

Note: This routine will be phased out!!!! Use HYPRE_BoomerAMGSetRelaxType or HYPRE_BoomerAMGSetCycleRelaxType instead.

6.2.55

int
HYPRE_BoomerAMGSetRelaxType (HYPRE_Solver solver, int relax_type)

(Optional) Defines the smoother to be used. It uses the given smoother on the fine grid, the up and the down cycle and sets the solver on the coarsest level to Gaussian elimination (9). The default is Gauss-Seidel (3).

There are the following options for relax_type:

0	Jacobi
1	Gauss-Seidel, sequential (very slow!)
2	Gauss-Seidel, interior points in parallel, boundary sequential (slow!)
3	hybrid Gauss-Seidel or SOR, forward solve
4	hybrid Gauss-Seidel or SOR, backward solve
5	hybrid chaotic Gauss-Seidel (works only with OpenMP)
6	hybrid symmetric Gauss-Seidel or SSOR
8	ℓ_1-scaled hybrid symmetric Gauss-Seidel
9	Gaussian elimination (only on coarsest level)
15	CG (warning - not a fixed smoother - may require FGMRES)
16	Chebyshev
17	FCF-Jacobi
18	ℓ_1-scaled jacobi

6.2.56

int
HYPRE_BoomerAMGSetCycleRelaxType (HYPRE_Solver solver, int relax_type, int k)

(Optional) Defines the smoother at a given cycle. For options of relax_type see description of HYPRE_BoomerAMGSetRelaxType). Options for k are

the down cycle	if k=1
the up cycle	if k=2
the coarsest level	if k=3.

6.2.57

int
HYPRE_BoomerAMGSetRelaxOrder (HYPRE_Solver solver, int relax_order)

(Optional) Defines in which order the points are relaxed. There are the following options for relax_order:

0	the points are relaxed in natural or lexicographic order on each processor
1	CF-relaxation is used, i.e on the fine grid and the down cycle the coarse points are relaxed first, followed by the fine points; on the up cycle the F-points are relaxed first, followed by the C-points. On the coarsest level, if an iterative scheme is used, the points are relaxed in lexicographic order.

The default is 1 (CF-relaxation).

6.2.58

int
HYPRE_BoomerAMGSetRelaxWt (HYPRE_Solver solver, HYPRE_Real relax_weight)

(Optional) Defines the relaxation weight for smoothed Jacobi and hybrid SOR on all levels.

relax_weight > 0	this assigns the given relaxation weight on all levels
relax_weight = 0	the weight is determined on each level with the estimate $\frac{3}{4\|D^{-1/2}AD^{-1/2}\|}$, where D is the diagonal matrix of A (this should only be used with Jacobi)
relax_weight = -k	the relaxation weight is determined with at most k CG steps on each level this should only be used for symmetric positive definite problems)

The default is 1.

6.2.59

int
HYPRE_BoomerAMGSetLevelRelaxWt (HYPRE_Solver solver, HYPRE_Real relax_weight, int level)

(Optional) Defines the relaxation weight for smoothed Jacobi and hybrid SOR on the user defined level. Note that the finest level is denoted 0, the next coarser level 1, etc. For nonpositive relax_weight, the parameter is determined on the given level as described for HYPRE_BoomerAMGSetRelaxWt. The default is 1.

6.2.60

int
HYPRE_BoomerAMGSetOmega (HYPRE_Solver solver, HYPRE_Real* omega)

(Optional) Defines the outer relaxation weight for hybrid SOR. Note: This routine will be phased out!!!! Use HYPRE_BoomerAMGSetOuterWt or HYPRE_BoomerAMGSetLevelOuterWt instead.

6.2.61

int
HYPRE_BoomerAMGSetOuterWt (HYPRE_Solver solver, HYPRE_Real omega)

(Optional) Defines the outer relaxation weight for hybrid SOR and SSOR on all levels.

omega > 0	this assigns the same outer relaxation weight omega on each level
omega = -k	an outer relaxation weight is determined with at most k CG steps on each level (this only makes sense for symmetric positive definite problems and smoothers, e.g. SSOR)

The default is 1.

6.2.62

int
HYPRE_BoomerAMGSetLevelOuterWt (HYPRE_Solver solver, HYPRE_Real omega, int level)

(Optional) Defines the outer relaxation weight for hybrid SOR or SSOR on the user defined level. Note that the finest level is denoted 0, the next coarser level 1, etc. For nonpositive omega, the parameter is determined on the given level as described for HYPRE_BoomerAMGSetOuterWt. The default is 1.

6.2.63

int **HYPRE_BoomerAMGSetChebyOrder** (HYPRE_Solver solver, int order)

(Optional) Defines the Order for Chebyshev smoother. The default is 2 (valid options are 1-4).

6.2.64

int
HYPRE_BoomerAMGSetChebyFraction (HYPRE_Solver solver, HYPRE_Real ratio)

(Optional) Fraction of the spectrum to use for the Chebyshev smoother. The default is .3 (i.e., damp on upper 30% of the spectrum).

6.2.65

int
HYPRE_BoomerAMGSetSmoothType (HYPRE_Solver solver, int smooth_type)

(Optional) Enables the use of more complex smoothers. The following options exist for smooth_type:

value	smoother	routines needed to set smoother parameters
6	Schwarz smoothers	HYPRE_BoomerAMGSetDomainType, HYPRE_BoomerAMGSetOverlap, HYPRE_BoomerAMGSetVariant, HYPRE_BoomerAMGSetSchwarzRlxWeight
7	Pilut	HYPRE_BoomerAMGSetDropTol, HYPRE_BoomerAMGSetMaxNzPerRow
8	ParaSails	HYPRE_BoomerAMGSetSym, HYPRE_BoomerAMGSetLevel, HYPRE_BoomerAMGSetFilter, HYPRE_BoomerAMGSetThreshold
9	Euclid	HYPRE_BoomerAMGSetEuclidFile

The default is 6. Also, if no smoother parameters are set via the routines mentioned in the table above, default values are used.

6.2.66

int **HYPRE_BoomerAMGSetSmoothNumLevels** (HYPRE_Solver solver, int smooth_num_levels)

(Optional) Sets the number of levels for more complex smoothers. The smoothers, as defined by HYPRE_BoomerAMGSetSmoothType, will be used on level 0 (the finest level) through level smooth_num_levels-1. The default is 0, i.e. no complex smoothers are used.

6.2.67

int **HYPRE_BoomerAMGSetSmoothNumSweeps** (HYPRE_Solver solver, int smooth_num_sweeps)

(Optional) Sets the number of sweeps for more complex smoothers. The default is 1.

6.2.68

int **HYPRE_BoomerAMGSetVariant** (HYPRE_Solver solver, int variant)

(Optional) Defines which variant of the Schwarz method is used. The following options exist for variant:

0	hybrid multiplicative Schwarz method (no overlap across processor boundaries)
1	hybrid additive Schwarz method (no overlap across processor boundaries)
2	additive Schwarz method
3	hybrid multiplicative Schwarz method (with overlap across processor boundaries)

The default is 0.

6.2.69

int **HYPRE_BoomerAMGSetOverlap** (HYPRE_Solver solver, int overlap)

(Optional) Defines the overlap for the Schwarz method. The following options exist for overlap:

0	no overlap
1	minimal overlap (default)
2	overlap generated by including all neighbors of domain boundaries

6.2.70

int
HYPRE_BoomerAMGSetDomainType (HYPRE_Solver solver, int domain_type)

(Optional) Defines the type of domain used for the Schwarz method. The following options exist for domain_type:

0	each point is a domain
1	each node is a domain (only of interest in "systems" AMG)
2	each domain is generated by agglomeration (default)

6.2.71

int
HYPRE_BoomerAMGSetSchwarzRlxWeight (HYPRE_Solver solver, HYPRE_Real schwarz_rlx_weight)

(Optional) Defines a smoothing parameter for the additive Schwarz method

6.2.72

int
HYPRE_BoomerAMGSetSchwarzUseNonSymm (HYPRE_Solver solver, int use_nonsymm)

(Optional) Indicates that the aggregates may not be SPD for the Schwarz method. The following options exist for use_nonsymm:

0	assume SPD (default)
1	assume non-symmetric

6.2.73

int **HYPRE_BoomerAMGSetSym** (HYPRE_Solver solver, int sym)

(Optional) Defines symmetry for ParaSAILS. For further explanation see description of ParaSAILS.

6.2.74

int **HYPRE_BoomerAMGSetLevel** (HYPRE_Solver solver, int level)

(Optional) Defines number of levels for ParaSAILS. For further explanation see description of ParaSAILS.

6.2.75

int
HYPRE_BoomerAMGSetThreshold (HYPRE_Solver solver, HYPRE_Real threshold)

(Optional) Defines threshold for ParaSAILS. For further explanation see description of ParaSAILS.

6.2.76

int **HYPRE_BoomerAMGSetFilter** (HYPRE_Solver solver, HYPRE_Real filter)

(Optional) Defines filter for ParaSAILS. For further explanation see description of ParaSAILS.

6.2.77

int
HYPRE_BoomerAMGSetDropTol (HYPRE_Solver solver, HYPRE_Real drop_tol)

(Optional) Defines drop tolerance for PILUT. For further explanation see description of PILUT.

6.2.78

int
HYPRE_BoomerAMGSetMaxNzPerRow (HYPRE_Solver solver, int max_nz_per_row)

(Optional) Defines maximal number of nonzeros for PILUT. For further explanation see description of PILUT.

6.2.79

int
HYPRE_BoomerAMGSetEuclidFile (HYPRE_Solver solver, char* euclidfile)

(Optional) Defines name of an input file for Euclid parameters. For further explanation see description of Euclid.

6.2.80

int **HYPRE_BoomerAMGSetEuLevel** (HYPRE_Solver solver, int eu_level)

(Optional) Defines number of levels for ILU(k) in Euclid. For further explanation see description of Euclid.

6.2.81

int
HYPRE_BoomerAMGSetEuSparseA (HYPRE_Solver solver, HYPRE_Real eu_sparse_A)

(Optional) Defines filter for ILU(k) for Euclid. For further explanation see description of Euclid.

6.2.82

int **HYPRE_BoomerAMGSetEuBJ** (HYPRE_Solver solver, int eu_bj)

(Optional) Defines use of block jacobi ILUT for Euclid. For further explanation see description of Euclid.

6.2.83

int **HYPRE_BoomerAMGSetPrintLevel** (HYPRE_Solver solver, int print_level)

(Optional) Requests automatic printing of setup and solve information.

0	no printout (default)
1	print setup information
2	print solve information
3	print both setup and solve information

Note, that if one desires to print information and uses BoomerAMG as a preconditioner, suggested print_level is 1 to avoid excessive output, and use print_level of solver for solve phase information.

6.2.84

int **HYPRE_BoomerAMGSetLogging** (HYPRE_Solver solver, int logging)

(Optional) Requests additional computations for diagnostic and similar data to be logged by the user. Default to 0 for do nothing. The latest residual will be available if logging > 1.

6.2.85

int
HYPRE_BoomerAMGSetDebugFlag (HYPRE_Solver solver, int debug_flag)

(Optional)

6.2.86

int
HYPRE_BoomerAMGInitGridRelaxation (int** num_grid_sweeps_ptr, int** grid_relax_type_ptr, int*** grid_relax_points_ptr, int coarsen_type, HYPRE_Real** relax_weights_ptr, int max_levels)

(Optional) This routine will be eliminated in the future

6.2.87

int **HYPRE_BoomerAMGSetRAP2** (HYPRE_Solver solver, int rap2)

(Optional) If rap2 not equal 0, the triple matrix product RAP is replaced by two matrix products

6.2.88

int
HYPRE_BoomerAMGSetKeepTranspose (HYPRE_Solver solver, int keepTranspose)

(Optional) If set to 1, the local interpolation transposes will be saved to use more efficient matvecs instead of matvecTs

6.3

ParCSR ParaSails Preconditioner

Names

6.3.1 int
HYPRE_ParaSailsCreate (MPI_Comm comm, HYPRE_Solver* solver)
Create a ParaSails preconditioner 164

6.3.2 int
HYPRE_ParaSailsDestroy (HYPRE_Solver solver)
Destroy a ParaSails preconditioner 164

6.3.3 int
HYPRE_ParaSailsSetup (HYPRE_Solver solver, HYPRE_ParCSRMatrix A, HYPRE_ParVector b, HYPRE_ParVector x)
Set up the ParaSails preconditioner. 164

6.3.4 int
HYPRE_ParaSailsSolve (HYPRE_Solver solver, HYPRE_ParCSRMatrix A, HYPRE_ParVector b, HYPRE_ParVector x)
Apply the ParaSails preconditioner. 165

6.3.5 int
HYPRE_ParaSailsSetParams (HYPRE_Solver solver, HYPRE_Real thresh, int nlevels)
Set the threshold and levels parameter for the ParaSails preconditioner. .. 165

6.3.6 int
HYPRE_ParaSailsSetFilter (HYPRE_Solver solver, HYPRE_Real filter)
Set the filter parameter for the ParaSails preconditioner. 165

6.3.7 int
HYPRE_ParaSailsSetSym (HYPRE_Solver solver, int sym)
Set the symmetry parameter for the ParaSails preconditioner. 166

6.3.8 int
HYPRE_ParaSailsSetLoadbal (HYPRE_Solver solver, HYPRE_Real loadbal)
Set the load balance parameter for the ParaSails preconditioner. 166

6.3.9 int

HYPRE_ParaSailsSetReuse (HYPRE_Solver solver, int reuse)
Set the pattern reuse parameter for the ParaSails preconditioner. 167

6.3.10 int
HYPRE_ParaSailsSetLogging (HYPRE_Solver solver, int logging)
Set the logging parameter for the ParaSails preconditioner. 167

6.3.11 int
HYPRE_ParaSailsBuildIJMatrix (HYPRE_Solver solver, HYPRE_IJMatrix* pij_A)
Build IJ Matrix of the sparse approximate inverse (factor). 167

Parallel sparse approximate inverse preconditioner for the ParCSR matrix format.

6.3.1

int **HYPRE_ParaSailsCreate** (MPI_Comm comm, HYPRE_Solver* solver)

Create a ParaSails preconditioner

6.3.2

int **HYPRE_ParaSailsDestroy** (HYPRE_Solver solver)

Destroy a ParaSails preconditioner

6.3.3

int
HYPRE_ParaSailsSetup (HYPRE_Solver solver, HYPRE_ParCSRMatrix A, HYPRE_ParVector b, HYPRE_ParVector x)

Set up the ParaSails preconditioner. This function should be passed to the iterative solver `SetPrecond` function.

Parameters:

`solver`	[IN] Preconditioner object to set up.
`A`	[IN] ParCSR matrix used to construct the preconditioner.
`b`	Ignored by this function.
`x`	Ignored by this function.

6.3.4

int
HYPRE_ParaSailsSolve (HYPRE_Solver solver, HYPRE_ParCSRMatrix A, HYPRE_ParVector b, HYPRE_ParVector x)

Apply the ParaSails preconditioner. This function should be passed to the iterative solver `SetPrecond` function.

Parameters:

`solver`	[IN] Preconditioner object to apply.
`A`	Ignored by this function.
`b`	[IN] Vector to precondition.
`x`	[OUT] Preconditioned vector.

6.3.5

int
HYPRE_ParaSailsSetParams (HYPRE_Solver solver, HYPRE_Real thresh, int nlevels)

Set the threshold and levels parameter for the ParaSails preconditioner. The accuracy and cost of ParaSails are parameterized by these two parameters. Lower values of the threshold parameter and higher values of levels parameter lead to more accurate, but more expensive preconditioners.

Parameters:

`solver`	[IN] Preconditioner object for which to set parameters.
`thresh`	[IN] Value of threshold parameter, $0 \leq$ thresh ≤ 1.The default value is 0.1.
`nlevels`	[IN] Value of levels parameter, $0 \leq$ nlevels. The default value is 1.

6.3.6

int **HYPRE_ParaSailsSetFilter** (HYPRE_Solver solver, HYPRE_Real filter)

Set the filter parameter for the ParaSails preconditioner.

Parameters:

`solver` [IN] Preconditioner object for which to set filter parameter.

`filter` [IN] Value of filter parameter. The filter parameter isused to drop small nonzeros in the preconditioner,to reduce the cost of applying the preconditioner.Values from 0.05 to 0.1 are recommended.The default value is 0.1.

6.3.7

int **HYPRE_ParaSailsSetSym** (HYPRE_Solver solver, int sym)

Set the symmetry parameter for the ParaSails preconditioner.

Parameters:

`solver` [IN] Preconditioner object for which to set symmetry parameter.

`sym` [IN] Value of the symmetry parameter:

value	meaning
0	nonsymmetric and/or indefinite problem, and nonsymmetric
1	SPD problem, and SPD (factored) preconditioner
2	nonsymmetric, definite problem, and SPD (factored) precon

6.3.8

int **HYPRE_ParaSailsSetLoadbal** (HYPRE_Solver solver, HYPRE_Real loadbal)

Set the load balance parameter for the ParaSails preconditioner.

Parameters:

`solver` [IN] Preconditioner object for which to set the load balanceparameter.

`loadbal` [IN] Value of the load balance parameter, $0 \leq$ loadbal ≤ 1. A zero value indicates thatno load balance is attempted; a value of unity indicatesthat perfect load balance will be attempted. The recommended value is 0.9 to balance the overhead ofdata exchanges for load balancing. No load balancingis needed if the preconditioner is very sparse andfast to construct. The default value when this parameter is not set is 0.

6.3.9

int **HYPRE_ParaSailsSetReuse** (HYPRE_Solver solver, int reuse)

Set the pattern reuse parameter for the ParaSails preconditioner.

Parameters:

`solver` [IN] Preconditioner object for which to set the pattern reuse parameter.

`reuse` [IN] Value of the pattern reuse parameter. A nonzero valueindicates that the pattern of the preconditioner shouldbe reused for subsequent constructions of the preconditioner. A zero value indicates that the preconditioner should be constructed from scratch.The default value when this parameter is not set is 0.

6.3.10

int **HYPRE_ParaSailsSetLogging** (HYPRE_Solver solver, int logging)

Set the logging parameter for the ParaSails preconditioner.

Parameters:

`solver` [IN] Preconditioner object for which to set the logging-parameter.

`logging` [IN] Value of the logging parameter. A nonzero value-sends statistics of the setup procedure to stdout.The default value when this parameter is not set is 0.

6.3.11

int
HYPRE_ParaSailsBuildIJMatrix (HYPRE_Solver solver, HYPRE_IJMatrix*
pij_A)

Build IJ Matrix of the sparse approximate inverse (factor). This function explicitly creates the IJ Matrix corresponding to the sparse approximate inverse or the inverse factor. Example: HYPRE_IJMatrix ij_A; HYPRE_ParaSailsBuildIJMatrix(solver, &ij_A);

Parameters:

`solver` [IN] Preconditioner object.

`pij_A` [OUT] Pointer to the IJ Matrix.

6.4

ParCSR Euclid Preconditioner

Names

6.4.1 int
HYPRE_EuclidCreate (MPI_Comm comm, HYPRE_Solver* solver)
Create a Euclid object 169

6.4.2 int
HYPRE_EuclidDestroy (HYPRE_Solver solver)
Destroy a Euclid object 169

6.4.3 int
HYPRE_EuclidSetup (HYPRE_Solver solver, HYPRE_ParCSRMatrix A, HYPRE_ParVector b, HYPRE_ParVector x)
Set up the Euclid preconditioner. 169

6.4.4 int
HYPRE_EuclidSolve (HYPRE_Solver solver, HYPRE_ParCSRMatrix A, HYPRE_ParVector b, HYPRE_ParVector x)
Apply the Euclid preconditioner. 170

6.4.5 int
HYPRE_EuclidSetParams (HYPRE_Solver solver, int argc, char* argv[])
Insert (name, value) pairs in Euclid's options database by passing Euclid the command line (or an array of strings). 170

6.4.6 int
HYPRE_EuclidSetParamsFromFile (HYPRE_Solver solver, char* filename)
Insert (name, value) pairs in Euclid's options database. 170

6.4.7 int
HYPRE_EuclidSetLevel (HYPRE_Solver solver, int level)
Set level k for ILU(k) factorization, default: 1 171

6.4.8 int
HYPRE_EuclidSetBJ (HYPRE_Solver solver, int bj)
Use block Jacobi ILU preconditioning instead of PILU 171

6.4.9 int
HYPRE_EuclidSetStats (HYPRE_Solver solver, int eu_stats)
If eu_stats not equal 0, a summary of runtime settings and timing information is printed to stdout 171

6.4.10 int
HYPRE_EuclidSetMem (HYPRE_Solver solver, int eu_mem)
If eu_mem not equal 0, a summary of Euclid's memory usage is printed to stdout 172

6.4.11 int
HYPRE_EuclidSetSparseA (HYPRE_Solver solver, HYPRE_Real sparse_A)
Defines a drop tolerance for ILU(k). 172

6.4.12 int

HYPRE_EuclidSetRowScale (HYPRE_Solver solver, int row_scale)
If row_scale not equal 0, values are scaled prior to factorization so that largest value in any row is +1 or -1. 172

6.4.13 int
HYPRE_EuclidSetILUT (HYPRE_Solver solver, HYPRE_Real drop_tol)
uses ILUT and defines a drop tolerance relative to the largest absolute value of any entry in the row being factored 172

MPI Parallel ILU preconditioner

Options summary:

Option	Default	Synopsis
-level	1	ILU(k) factorization level
-bj	0 (false)	Use Block Jacobi ILU instead of PILU
-eu_stats	0 (false)	Print internal timing and statistics
-eu_mem	0 (false)	Print internal memory usage

6.4.1

int **HYPRE_EuclidCreate** (MPI_Comm comm, HYPRE_Solver* solver)

Create a Euclid object

6.4.2

int **HYPRE_EuclidDestroy** (HYPRE_Solver solver)

Destroy a Euclid object

6.4.3

int
HYPRE_EuclidSetup (HYPRE_Solver solver, HYPRE_ParCSRMatrix A, HYPRE_ParVector b, HYPRE_ParVector x)

Set up the Euclid preconditioner. This function should be passed to the iterative solver `SetPrecond` function.

Parameters:

`solver`	[IN] Preconditioner object to set up.
`A`	[IN] ParCSR matrix used to construct the preconditioner.
`b`	Ignored by this function.
`x`	Ignored by this function.

6.4.4

int
HYPRE_EuclidSolve (HYPRE_Solver solver, HYPRE_ParCSRMatrix A, HYPRE_ParVector b, HYPRE_ParVector x)

Apply the Euclid preconditioner. This function should be passed to the iterative solver `SetPrecond` function.

Parameters:

`solver`	[IN] Preconditioner object to apply.
`A`	Ignored by this function.
`b`	[IN] Vector to precondition.
`x`	[OUT] Preconditioned vector.

6.4.5

int **HYPRE_EuclidSetParams** (HYPRE_Solver solver, int argc, char* argv[])

Insert (name, value) pairs in Euclid's options database by passing Euclid the command line (or an array of strings). All Euclid options (e.g, level, drop-tolerance) are stored in this database. If a (name, value) pair already exists, this call updates the value. See also: HYPRE_EuclidSetParamsFromFile.

Parameters:

`argc`	[IN] Length of argv array
`argv`	[IN] Array of strings

6.4.6

int **HYPRE_EuclidSetParamsFromFile** (HYPRE_Solver solver, char* filename)

Insert (name, value) pairs in Euclid's options database. Each line of the file should either begin with a "#," indicating a comment line, or contain a (name value) pair, e.g:

```
>cat optionsFile
#sample runtime parameter file
-blockJacobi 3
-matFile /home/hysom/myfile.euclid
-doSomething true
-xx_coeff -1.0
```

See also: HYPRE_EuclidSetParams.

Parameters: `filename[IN]` Pathname/filename to read

6.4.7

int **HYPRE_EuclidSetLevel** (HYPRE_Solver solver, int level)

Set level k for ILU(k) factorization, default: 1

6.4.8

int **HYPRE_EuclidSetBJ** (HYPRE_Solver solver, int bj)

Use block Jacobi ILU preconditioning instead of PILU

6.4.9

int **HYPRE_EuclidSetStats** (HYPRE_Solver solver, int eu_stats)

If eu_stats not equal 0, a summary of runtime settings and timing information is printed to stdout

6.4.10

int **HYPRE_EuclidSetMem** (HYPRE_Solver solver, int eu_mem)

If eu_mem not equal 0, a summary of Euclid's memory usage is printed to stdout

6.4.11

int **HYPRE_EuclidSetSparseA** (HYPRE_Solver solver, HYPRE_Real sparse_A)

Defines a drop tolerance for ILU(k). Default: 0 Use with HYPRE_EuclidSetRowScale. Note that this can destroy symmetry in a matrix.

6.4.12

int **HYPRE_EuclidSetRowScale** (HYPRE_Solver solver, int row_scale)

If row_scale not equal 0, values are scaled prior to factorization so that largest value in any row is +1 or -1. Note that this can destroy symmetry in a matrix.

6.4.13

int **HYPRE_EuclidSetILUT** (HYPRE_Solver solver, HYPRE_Real drop_tol)

uses ILUT and defines a drop tolerance relative to the largest absolute value of any entry in the row being factored

6.5

ParCSR Pilut Preconditioner

Names

6.5.1 int
HYPRE_ParCSRPilutCreate (MPI_Comm comm, HYPRE_Solver* solver)
Create a preconditioner object 173

6.5.2 int
HYPRE_ParCSRPilutDestroy (HYPRE_Solver solver)
Destroy a preconditioner object 174

6.5.3 int
HYPRE_ParCSRPilutSetup (HYPRE_Solver solver,
HYPRE_ParCSRMatrix A,
HYPRE_ParVector b, HYPRE_ParVector x)
.... 174

6.5.4 int
HYPRE_ParCSRPilutSolve (HYPRE_Solver solver,
HYPRE_ParCSRMatrix A,
HYPRE_ParVector b, HYPRE_ParVector x)
Precondition the system 174

6.5.5 int
HYPRE_ParCSRPilutSetMaxIter (HYPRE_Solver solver, int max_iter)
(Optional) Set maximum number of iterations 174

6.5.6 int
HYPRE_ParCSRPilutSetDropTolerance (HYPRE_Solver solver,
HYPRE_Real tol)
(Optional) 174

6.5.7 int
HYPRE_ParCSRPilutSetFactorRowSize (HYPRE_Solver solver, int size)
(Optional) 175

6.5.1

int **HYPRE_ParCSRPilutCreate** (MPI_Comm comm, HYPRE_Solver* solver)

Create a preconditioner object

6.5.2

int **HYPRE_ParCSRPilutDestroy** (HYPRE_Solver solver)

Destroy a preconditioner object

6.5.3

int
HYPRE_ParCSRPilutSetup (HYPRE_Solver solver, HYPRE_ParCSRMatrix A, HYPRE_ParVector b, HYPRE_ParVector x)

6.5.4

int
HYPRE_ParCSRPilutSolve (HYPRE_Solver solver, HYPRE_ParCSRMatrix A, HYPRE_ParVector b, HYPRE_ParVector x)

Precondition the system

6.5.5

int **HYPRE_ParCSRPilutSetMaxIter** (HYPRE_Solver solver, int max_iter)

(Optional) Set maximum number of iterations

6.5.6

int
HYPRE_ParCSRPilutSetDropTolerance (HYPRE_Solver solver, HYPRE_Real tol)

(Optional)

6.5.7

int **HYPRE_ParCSRPilutSetFactorRowSize** (HYPRE_Solver solver, int size)

(Optional)

6.6

ParCSR AMS Solver and Preconditioner

Names

6.6.1 int
HYPRE_AMSCreate (HYPRE_Solver* solver)
Create an AMS solver object 178

6.6.2 int
HYPRE_AMSDestroy (HYPRE_Solver solver)
Destroy an AMS solver object 178

6.6.3 int
HYPRE_AMSSetup (HYPRE_Solver solver, HYPRE_ParCSRMatrix A, HYPRE_ParVector b, HYPRE_ParVector x)
Set up the AMS solver or preconditioner. 178

6.6.4 int
HYPRE_AMSSolve (HYPRE_Solver solver, HYPRE_ParCSRMatrix A, HYPRE_ParVector b, HYPRE_ParVector x)
Solve the system or apply AMS as a preconditioner. 179

6.6.5 int
HYPRE_AMSSetDimension (HYPRE_Solver solver, int dim)
(Optional) Sets the problem dimension (2 or 3). 179

6.6.6 int
HYPRE_AMSSetDiscreteGradient (HYPRE_Solver solver, HYPRE_ParCSRMatrix G)
Sets the discrete gradient matrix G. 179

6.6.7 int
HYPRE_AMSSetCoordinateVectors (HYPRE_Solver solver, HYPRE_ParVector x, HYPRE_ParVector y, HYPRE_ParVector z)
Sets the x, y and z coordinates of the vertices in the mesh. 179

6.6.8 int

HYPRE_AMSSetEdgeConstantVectors (HYPRE_Solver solver,
HYPRE_ParVector Gx,
HYPRE_ParVector Gy,
HYPRE_ParVector Gz)
Sets the vectors Gx, Gy and Gz which give the representations of the constant vector fields (1, 0, 0), (0, 1, 0) *and* (0, 0, 1) *in the edge element basis.* 180

6.6.9 int
HYPRE_AMSSetInterpolations (HYPRE_Solver solver,
HYPRE_ParCSRMatrix Pi,
HYPRE_ParCSRMatrix Pix,
HYPRE_ParCSRMatrix Piy,
HYPRE_ParCSRMatrix Piz)
(Optional) Set the (components of) the Nedelec interpolation matrix $\Pi = [\Pi^x, \Pi^y, \Pi^z]$. 180

6.6.10 int
HYPRE_AMSSetAlphaPoissonMatrix (HYPRE_Solver solver,
HYPRE_ParCSRMatrix A_alpha)
(Optional) Sets the matrix A_α *corresponding to the Poisson problem with coefficient* α *(the curl-curl term coefficient in the Maxwell problem).* 181

6.6.11 int
HYPRE_AMSSetBetaPoissonMatrix (HYPRE_Solver solver,
HYPRE_ParCSRMatrix A_beta)
(Optional) Sets the matrix A_β *corresponding to the Poisson problem with coefficient* β *(the mass term coefficient in the Maxwell problem).* 181

6.6.12 int
HYPRE_AMSSetInteriorNodes (HYPRE_Solver solver,
HYPRE_ParVector interior_nodes)
(Optional) Set the list of nodes which are interior to a zero-conductivity region. 181

6.6.13 int
HYPRE_AMSSetProjectionFrequency (HYPRE_Solver solver,
int projection_frequency)
(Optional) Set the frequency at which a projection onto the compatible subspace for problems with zero-conductivity regions is performed. 182

6.6.14 int
HYPRE_AMSSetMaxIter (HYPRE_Solver solver, int maxit)
(Optional) Sets maximum number of iterations, if AMS is used as a solver. 182

6.6.15 int
HYPRE_AMSSetTol (HYPRE_Solver solver, HYPRE_Real tol)
(Optional) Set the convergence tolerance, if AMS is used as a solver. . . . 182

6.6.16 int
HYPRE_AMSSetCycleType (HYPRE_Solver solver, int cycle_type)
(Optional) Choose which three-level solver to use. 182

6.6.17 int

HYPRE_AMSSetPrintLevel (HYPRE_Solver solver, int print_level)
(Optional) Control how much information is printed during the solution iterations. 183

6.6.18 int
HYPRE_AMSSetSmoothingOptions (HYPRE_Solver solver, int relax_type, int relax_times, HYPRE_Real relax_weight, HYPRE_Real omega)
(Optional) Sets relaxation parameters for A. 183

6.6.19 int
HYPRE_AMSSetAlphaAMGOptions (HYPRE_Solver solver, int alpha_coarsen_type, int alpha_agg_levels, int alpha_relax_type, HYPRE_Real alpha_strength_threshold, int alpha_interp_type, int alpha_Pmax)
(Optional) Sets AMG parameters for B_Π. 184

6.6.20 int
HYPRE_AMSSetAlphaAMGCoarseRelaxType (HYPRE_Solver solver, int alpha_coarse_relax_type)
(Optional) Sets the coarsest level relaxation in the AMG solver for B_Π. .. 184

6.6.21 int
HYPRE_AMSSetBetaAMGOptions (HYPRE_Solver solver, int beta_coarsen_type, int beta_agg_levels, int beta_relax_type, HYPRE_Real beta_strength_threshold, int beta_interp_type, int beta_Pmax)
(Optional) Sets AMG parameters for B_G. 184

6.6.22 int
HYPRE_AMSSetBetaAMGCoarseRelaxType (HYPRE_Solver solver, int beta_coarse_relax_type)
(Optional) Sets the coarsest level relaxation in the AMG solver for B_G. . 184

6.6.23 int
HYPRE_AMSGetNumIterations (HYPRE_Solver solver, int* num_iterations)
Returns the number of iterations taken 185

6.6.24 int
HYPRE_AMSGetFinalRelativeResidualNorm (HYPRE_Solver solver, HYPRE_Real* rel_resid_norm)
Returns the norm of the final relative residual 185

6.6.25 int
HYPRE_AMSProjectOutGradients (HYPRE_Solver solver, HYPRE_ParVector x)
For problems with zero-conductivity regions, project the vector onto the compatible subspace: $x = (I - G_0(G_0^t G_0)^{-1} G_0^T)x$, where G_0 is the discrete gradient restricted to the interior nodes of the regions with zero conductivity. 185

6.6.26 int

HYPRE_AMSConstructDiscreteGradient (HYPRE_ParCSRMatrix A, HYPRE_ParVector x_coord, int* edge_vertex, int edge_orientation, HYPRE_ParCSRMatrix* G)
Construct and return the lowest-order discrete gradient matrix G using some edge and vertex information. 185

Parallel auxiliary space Maxwell solver and preconditioner

6.6.1

int **HYPRE_AMSCreate** (HYPRE_Solver* solver)

Create an AMS solver object

6.6.2

int **HYPRE_AMSDestroy** (HYPRE_Solver solver)

Destroy an AMS solver object

6.6.3

int
HYPRE_AMSSetup (HYPRE_Solver solver, HYPRE_ParCSRMatrix A, HYPRE_ParVector b, HYPRE_ParVector x)

Set up the AMS solver or preconditioner. If used as a preconditioner, this function should be passed to the iterative solver `SetPrecond` function.

Parameters:

`solver`	[IN] object to be set up.
`A`	[IN] ParCSR matrix used to construct the solver/preconditioner.
`b`	Ignored by this function.
`x`	Ignored by this function.

6.6.4

int
HYPRE_AMSSolve (HYPRE_Solver solver, HYPRE_ParCSRMatrix A,
HYPRE_ParVector b, HYPRE_ParVector x)

Solve the system or apply AMS as a preconditioner. If used as a preconditioner, this function should be passed to the iterative solver `SetPrecond` function.

Parameters:

`solver`	[IN] solver or preconditioner object to be applied.
`A`	[IN] ParCSR matrix, matrix of the linear system to be solved
`b`	[IN] right hand side of the linear system to be solved
`x`	[OUT] approximated solution of the linear system to be solved

6.6.5

int **HYPRE_AMSSetDimension** (HYPRE_Solver solver, int dim)

(Optional) Sets the problem dimension (2 or 3). The default is 3.

6.6.6

int
HYPRE_AMSSetDiscreteGradient (HYPRE_Solver solver,
HYPRE_ParCSRMatrix G)

Sets the discrete gradient matrix G. This function should be called before HYPRE_AMSSetup()!

6.6.7

int
HYPRE_AMSSetCoordinateVectors (HYPRE_Solver solver,
HYPRE_ParVector x, HYPRE_ParVector y, HYPRE_ParVector z)

Sets the x, y and z coordinates of the vertices in the mesh.

Either HYPRE_AMSSetCoordinateVectors() or HYPRE_AMSSetEdgeConstantVectors() should be called before HYPRE_AMSSetup()!

6.6.8

int
HYPRE_AMSSetEdgeConstantVectors (HYPRE_Solver solver, HYPRE_ParVector Gx, HYPRE_ParVector Gy, HYPRE_ParVector Gz)

Sets the vectors Gx, Gy and Gz which give the representations of the constant vector fields $(1, 0, 0)$, $(0, 1, 0)$ and $(0, 0, 1)$ in the edge element basis.

Either HYPRE_AMSSetCoordinateVectors() or HYPRE_AMSSetEdgeConstantVectors() should be called before HYPRE_AMSSetup()!

6.6.9

int
HYPRE_AMSSetInterpolations (HYPRE_Solver solver, HYPRE_ParCSRMatrix Pi, HYPRE_ParCSRMatrix Pix, HYPRE_ParCSRMatrix Piy, HYPRE_ParCSRMatrix Piz)

(Optional) Set the (components of) the Nedelec interpolation matrix $\Pi = [\Pi^x, \Pi^y, \Pi^z]$.

This function is generally intended to be used only for high-order Nedelec discretizations (in the lowest order case, Π is constructed internally in AMS from the discreet gradient matrix and the coordinates of the vertices), though it can also be used in the lowest-order case or for other types of discretizations (e.g. ones based on the second family of Nedelec elements).

By definition, Π is the matrix representation of the linear operator that interpolates (high-order) vector nodal finite elements into the (high-order) Nedelec space. The component matrices are defined as $\Pi^x \varphi = \Pi(\varphi, 0, 0)$ and similarly for Π^y and Π^z. Note that all these operators depend on the choice of the basis and degrees of freedom in the high-order spaces.

The column numbering of Pi should be node-based, i.e. the $x/y/z$ components of the first node (vertex or high-order dof) should be listed first, followed by the $x/y/z$ components of the second node and so on (see the documentation of HYPRE_BoomerAMGSetDofFunc).

If used, this function should be called before HYPRE_AMSSetup() and there is no need to provide the vertex coordinates. Furthermore, only one of the sets $\{\Pi\}$ and $\{\Pi^x, \Pi^y, \Pi^z\}$ needs to be specified (though it is OK to provide both). If Pix is NULL, then scalar Π-based AMS cycles, i.e. those with cycle_type > 10, will be unavailable. Similarly, AMS cycles based on monolithic Π (cycle_type < 10) require that Pi is not NULL.

6.6.10

int
HYPRE_AMSSetAlphaPoissonMatrix (HYPRE_Solver solver, HYPRE_ParCSRMatrix A_alpha)

(Optional) Sets the matrix A_α corresponding to the Poisson problem with coefficient α (the curl-curl term coefficient in the Maxwell problem).

If this function is called, the coarse space solver on the range of Π^T is a block-diagonal version of A_Π. If this function is not called, the coarse space solver on the range of Π^T is constructed as $\Pi^T A \Pi$ in HYPRE_AMSSetup(). See the user's manual for more details.

6.6.11

int
HYPRE_AMSSetBetaPoissonMatrix (HYPRE_Solver solver, HYPRE_ParCSRMatrix A_beta)

(Optional) Sets the matrix A_β corresponding to the Poisson problem with coefficient β (the mass term coefficient in the Maxwell problem).

If not given, the Poisson matrix will be computed in HYPRE_AMSSetup(). If the given matrix is NULL, we assume that β is identically 0 and use two-level (instead of three-level) methods. See the user's manual for more details.

6.6.12

int
HYPRE_AMSSetInteriorNodes (HYPRE_Solver solver, HYPRE_ParVector interior_nodes)

(Optional) Set the list of nodes which are interior to a zero-conductivity region. This way, a more robust solver is constructed, that can be iterated to lower tolerance levels. A node is interior if its entry in the array is 1.0. This function should be called before HYPRE_AMSSetup()!

6.6.13

int
HYPRE_AMSSetProjectionFrequency (HYPRE_Solver solver, int projection_frequency)

(Optional) Set the frequency at which a projection onto the compatible subspace for problems with zero-conductivity regions is performed. The default value is 5.

6.6.14

int **HYPRE_AMSSetMaxIter** (HYPRE_Solver solver, int maxit)

(Optional) Sets maximum number of iterations, if AMS is used as a solver. To use AMS as a preconditioner, set the maximum number of iterations to 1. The default is 20.

6.6.15

int **HYPRE_AMSSetTol** (HYPRE_Solver solver, HYPRE_Real tol)

(Optional) Set the convergence tolerance, if AMS is used as a solver. When using AMS as a preconditioner, set the tolerance to 0.0. The default is 10^{-6}.

6.6.16

int **HYPRE_AMSSetCycleType** (HYPRE_Solver solver, int cycle_type)

(Optional) Choose which three-level solver to use. Possible values are:

1	3-level multiplicative solver (01210)
2	3-level additive solver (0+1+2)
3	3-level multiplicative solver (02120)
4	3-level additive solver (010+2)
5	3-level multiplicative solver (0102010)
6	3-level additive solver (1+020)
7	3-level multiplicative solver (0201020)
8	3-level additive solver (0(1+2)0)
11	5-level multiplicative solver (013454310)
12	5-level additive solver (0+1+3+4+5)
13	5-level multiplicative solver (034515430)
14	5-level additive solver (01(3+4+5)10)

The default is 1. See the user's manual for more details.

6.6.17

int **HYPRE_AMSSetPrintLevel** (HYPRE_Solver solver, int print_level)

(Optional) Control how much information is printed during the solution iterations. The default is 1 (print residual norm at each step).

6.6.18

int
HYPRE_AMSSetSmoothingOptions (HYPRE_Solver solver, int relax_type, int relax_times, HYPRE_Real relax_weight, HYPRE_Real omega)

(Optional) Sets relaxation parameters for A. The defaults are 2, 1, 1.0, 1.0.

The available options for relax_type are:

1	ℓ_1-scaled Jacobi
2	ℓ_1-scaled block symmetric Gauss-Seidel/SSOR
3	Kaczmarz
4	truncated version of ℓ_1-scaled block symmetric Gauss-Seidel/SSOR
16	Chebyshev

6.6.19

int
HYPRE_AMSSetAlphaAMGOptions (HYPRE_Solver solver, int alpha_coarsen_type, int alpha_agg_levels, int alpha_relax_type, HYPRE_Real alpha_strength_threshold, int alpha_interp_type, int alpha_Pmax)

(Optional) Sets AMG parameters for B_Π. The defaults are 10, 1, 3, 0.25, 0, 0. See the user's manual for more details.

6.6.20

int
HYPRE_AMSSetAlphaAMGCoarseRelaxType (HYPRE_Solver solver, int alpha_coarse_relax_type)

(Optional) Sets the coarsest level relaxation in the AMG solver for B_Π. The default is 8 (l1-GS). Use 9, 19, 29 or 99 for a direct solver.

6.6.21

int
HYPRE_AMSSetBetaAMGOptions (HYPRE_Solver solver, int beta_coarsen_type, int beta_agg_levels, int beta_relax_type, HYPRE_Real beta_strength_threshold, int beta_interp_type, int beta_Pmax)

(Optional) Sets AMG parameters for B_G. The defaults are 10, 1, 3, 0.25, 0, 0. See the user's manual for more details.

6.6.22

int
HYPRE_AMSSetBetaAMGCoarseRelaxType (HYPRE_Solver solver, int beta_coarse_relax_type)

(Optional) Sets the coarsest level relaxation in the AMG solver for B_G. The default is 8 (l1-GS). Use 9, 19, 29 or 99 for a direct solver.

6.6.23

int
HYPRE_AMSGetNumIterations (HYPRE_Solver solver, int* num_iterations)

Returns the number of iterations taken

6.6.24

int
HYPRE_AMSGetFinalRelativeResidualNorm (HYPRE_Solver solver, HYPRE_Real* rel_resid_norm)

Returns the norm of the final relative residual

6.6.25

int
HYPRE_AMSProjectOutGradients (HYPRE_Solver solver, HYPRE_ParVector x)

For problems with zero-conductivity regions, project the vector onto the compatible subspace: $x = (I - G_0(G_0^t G_0)^{-1} G_0^T)x$, where G_0 is the discrete gradient restricted to the interior nodes of the regions with zero conductivity. This ensures that x is orthogonal to the gradients in the range of G_0.

This function is typically called after the solution iteration is complete, in order to facilitate the visualization of the computed field. Without it the values in the zero-conductivity regions contain kernel components.

6.6.26

int
HYPRE_AMSConstructDiscreteGradient (HYPRE_ParCSRMatrix A, HYPRE_ParVector x_coord, int* edge_vertex, int edge_orientation, HYPRE_ParCSRMatrix* G)

Construct and return the lowest-order discrete gradient matrix G using some edge and vertex information. We assume that edge_vertex lists the edge vertices consecutively, and that the orientation of all edges is consistent.

If edge_orientation = 1, the edges are already oriented.

If edge_orientation = 2, the orientation of edge i depends only on the sign of edge_vertex[2*i+1] - edge_vertex[2*i].

6.7

ParCSR ADS Solver and Preconditioner

Names

6.7.1 int
HYPRE_ADSCreate (HYPRE_Solver* solver)
Create an ADS solver object 188

6.7.2 int
HYPRE_ADSDestroy (HYPRE_Solver solver)
Destroy an ADS solver object 188

6.7.3 int
HYPRE_ADSSetup (HYPRE_Solver solver, HYPRE_ParCSRMatrix A, HYPRE_ParVector b, HYPRE_ParVector x)
Set up the ADS solver or preconditioner. 188

6.7.4 int
HYPRE_ADSSolve (HYPRE_Solver solver, HYPRE_ParCSRMatrix A, HYPRE_ParVector b, HYPRE_ParVector x)
Solve the system or apply ADS as a preconditioner. 189

6.7.5 int
HYPRE_ADSSetDiscreteCurl (HYPRE_Solver solver, HYPRE_ParCSRMatrix C)
Sets the discrete curl matrix C. 189

6.7.6 int
HYPRE_ADSSetDiscreteGradient (HYPRE_Solver solver, HYPRE_ParCSRMatrix G)
Sets the discrete gradient matrix G. 189

6.7.7 int
HYPRE_ADSSetCoordinateVectors (HYPRE_Solver solver, HYPRE_ParVector x, HYPRE_ParVector y, HYPRE_ParVector z)
Sets the x, y and z coordinates of the vertices in the mesh. 190

6.7.8 int

HYPRE_ADSSetInterpolations (HYPRE_Solver solver,
HYPRE_ParCSRMatrix RT_Pi,
HYPRE_ParCSRMatrix RT_Pix,
HYPRE_ParCSRMatrix RT_Piy,
HYPRE_ParCSRMatrix RT_Piz,
HYPRE_ParCSRMatrix ND_Pi,
HYPRE_ParCSRMatrix ND_Pix,
HYPRE_ParCSRMatrix ND_Piy,
HYPRE_ParCSRMatrix ND_Piz)
(Optional) Set the (components of) the Raviart-Thomas (Π_{RT}) and the Nedelec (Π_{ND}) interpolation matrices. 190

6.7.9 int
HYPRE_ADSSetMaxIter (HYPRE_Solver solver, int maxit)
(Optional) Sets maximum number of iterations, if ADS is used as a solver. 191

6.7.10 int
HYPRE_ADSSetTol (HYPRE_Solver solver, HYPRE_Real tol)
(Optional) Set the convergence tolerance, if ADS is used as a solver. 191

6.7.11 int
HYPRE_ADSSetCycleType (HYPRE_Solver solver, int cycle_type)
(Optional) Choose which auxiliary-space solver to use. 191

6.7.12 int
HYPRE_ADSSetPrintLevel (HYPRE_Solver solver, int print_level)
(Optional) Control how much information is printed during the solution iterations. 192

6.7.13 int
HYPRE_ADSSetSmoothingOptions (HYPRE_Solver solver, int relax_type,
int relax_times,
HYPRE_Real relax_weight,
HYPRE_Real omega)
(Optional) Sets relaxation parameters for A. 192

6.7.14 int
HYPRE_ADSSetChebySmoothingOptions (HYPRE_Solver solver,
int cheby_order,
int cheby_fraction)
(Optional) Sets parameters for Chebyshev relaxation. 192

6.7.15 int
HYPRE_ADSSetAMSOptions (HYPRE_Solver solver, int cycle_type,
int coarsen_type, int agg_levels,
int relax_type,
HYPRE_Real strength_threshold,
int interp_type, int Pmax)
(Optional) Sets AMS parameters for B_C. 192

6.7.16 int
HYPRE_ADSSetAMGOptions (HYPRE_Solver solver, int coarsen_type,
int agg_levels, int relax_type,
HYPRE_Real strength_threshold,
int interp_type, int Pmax)
(Optional) Sets AMG parameters for B_Π. 193

6.7.17 int

HYPRE_ADSGetNumIterations (HYPRE_Solver solver, int* num_iterations)
Returns the number of iterations taken 193

6.7.18 int
HYPRE_ADSGetFinalRelativeResidualNorm (HYPRE_Solver solver, HYPRE_Real* rel_resid_norm)
Returns the norm of the final relative residual 193

Parallel auxiliary space divergence solver and preconditioner

6.7.1

int **HYPRE_ADSCreate** (HYPRE_Solver* solver)

Create an ADS solver object

6.7.2

int **HYPRE_ADSDestroy** (HYPRE_Solver solver)

Destroy an ADS solver object

6.7.3

int
HYPRE_ADSSetup (HYPRE_Solver solver, HYPRE_ParCSRMatrix A, HYPRE_ParVector b, HYPRE_ParVector x)

Set up the ADS solver or preconditioner. If used as a preconditioner, this function should be passed to the iterative solver `SetPrecond` function.

Parameters:

`solver`	[IN] object to be set up.
`A`	[IN] ParCSR matrix used to construct the solver/preconditioner.
`b`	Ignored by this function.
`x`	Ignored by this function.

6.7.4

int
HYPRE_ADSSolve (HYPRE_Solver solver, HYPRE_ParCSRMatrix A, HYPRE_ParVector b, HYPRE_ParVector x)

Solve the system or apply ADS as a preconditioner. If used as a preconditioner, this function should be passed to the iterative solver `SetPrecond` function.

Parameters:

`solver`	[IN] solver or preconditioner object to be applied.
`A`	[IN] ParCSR matrix, matrix of the linear system to be solved
`b`	[IN] right hand side of the linear system to be solved
`x`	[OUT] approximated solution of the linear system to be solved

6.7.5

int
HYPRE_ADSSetDiscreteCurl (HYPRE_Solver solver, HYPRE_ParCSRMatrix C)

Sets the discrete curl matrix C. This function should be called before HYPRE_ADSSetup()!

6.7.6

int
HYPRE_ADSSetDiscreteGradient (HYPRE_Solver solver, HYPRE_ParCSRMatrix G)

Sets the discrete gradient matrix G. This function should be called before HYPRE_ADSSetup()!

6.7.7

int
HYPRE_ADSSetCoordinateVectors (HYPRE_Solver solver, HYPRE_ParVector x, HYPRE_ParVector y, HYPRE_ParVector z)

Sets the x, y and z coordinates of the vertices in the mesh. This function should be called before HYPRE_ADSSetup()!

6.7.8

int
HYPRE_ADSSetInterpolations (HYPRE_Solver solver, HYPRE_ParCSRMatrix RT_Pi, HYPRE_ParCSRMatrix RT_Pix, HYPRE_ParCSRMatrix RT_Piy, HYPRE_ParCSRMatrix RT_Piz, HYPRE_ParCSRMatrix ND_Pi, HYPRE_ParCSRMatrix ND_Pix, HYPRE_ParCSRMatrix ND_Piy, HYPRE_ParCSRMatrix ND_Piz)

(Optional) Set the (components of) the Raviart-Thomas (Π_{RT}) and the Nedelec (Π_{ND}) interpolation matrices.

This function is generally intended to be used only for high-order $H(div)$ discretizations (in the lowest order case, these matrices are constructed internally in ADS from the discreet gradient and curl matrices and the coordinates of the vertices), though it can also be used in the lowest-order case or for other types of discretizations.

By definition, RT_Pi and ND_Pi are the matrix representations of the linear operators Π_{RT} and Π_{ND} that interpolate (high-order) vector nodal finite elements into the (high-order) Raviart-Thomas and Nedelec spaces. The component matrices are defined in both cases as $\Pi^x \varphi = \Pi(\varphi, 0, 0)$ and similarly for Π^y and Π^z. Note that all these operators depend on the choice of the basis and degrees of freedom in the high-order spaces.

The column numbering of RT_Pi and ND_Pi should be node-based, i.e. the $x/y/z$ components of the first node (vertex or high-order dof) should be listed first, followed by the $x/y/z$ components of the second node and so on (see the documentation of HYPRE_BoomerAMGSetDofFunc).

If used, this function should be called before hypre_ADSSetup() and there is no need to provide the vertex coordinates. Furthermore, only one of the sets $\{\Pi_{RT}\}$ and $\{\Pi_{RT}^x, \Pi_{RT}^y, \Pi_{RT}^z\}$ needs to be specified (though it is OK to provide both). If RT_Pix is NULL, then scalar Π-based ADS cycles, i.e. those with cycle_type > 10, will be unavailable. Similarly, ADS cycles based on monolithic Π (cycle_type < 10) require that RT_Pi is not NULL. The same restrictions hold for the sets $\{\Pi_{ND}\}$ and $\{\Pi_{ND}^x, \Pi_{ND}^y, \Pi_{ND}^z\}$ – only one of them needs to be specified, and the availability of each enables different AMS cycle type options.

6.7.9

int **HYPRE_ADSSetMaxIter** (HYPRE_Solver solver, int maxit)

(Optional) Sets maximum number of iterations, if ADS is used as a solver. To use ADS as a preconditioner, set the maximum number of iterations to 1. The default is 20.

6.7.10

int **HYPRE_ADSSetTol** (HYPRE_Solver solver, HYPRE_Real tol)

(Optional) Set the convergence tolerance, if ADS is used as a solver. When using ADS as a preconditioner, set the tolerance to 0.0. The default is 10^{-6}.

6.7.11

int **HYPRE_ADSSetCycleType** (HYPRE_Solver solver, int cycle_type)

(Optional) Choose which auxiliary-space solver to use. Possible values are:

1	3-level multiplicative solver (01210)
2	3-level additive solver (0+1+2)
3	3-level multiplicative solver (02120)
4	3-level additive solver (010+2)
5	3-level multiplicative solver (0102010)
6	3-level additive solver (1+020)
7	3-level multiplicative solver (0201020)
8	3-level additive solver (0(1+2)0)
11	5-level multiplicative solver (013454310)
12	5-level additive solver (0+1+3+4+5)
13	5-level multiplicative solver (034515430)
14	5-level additive solver (01(3+4+5)10)

The default is 1. See the user's manual for more details.

6.7.12

int **HYPRE_ADSSetPrintLevel** (HYPRE_Solver solver, int print_level)

(Optional) Control how much information is printed during the solution iterations. The default is 1 (print residual norm at each step).

6.7.13

int
HYPRE_ADSSetSmoothingOptions (HYPRE_Solver solver, int relax_type, int relax_times, HYPRE_Real relax_weight, HYPRE_Real omega)

(Optional) Sets relaxation parameters for A. The defaults are 2, 1, 1.0, 1.0.

The available options for relax_type are:

1	ℓ_1-scaled Jacobi
2	ℓ_1-scaled block symmetric Gauss-Seidel/SSOR
3	Kaczmarz
4	truncated version of ℓ_1-scaled block symmetric Gauss-Seidel/SSOR
16	Chebyshev

6.7.14

int
HYPRE_ADSSetChebySmoothingOptions (HYPRE_Solver solver, int cheby_order, int cheby_fraction)

(Optional) Sets parameters for Chebyshev relaxation. The defaults are 2, 0.3.

6.7.15

int
HYPRE_ADSSetAMSOptions (HYPRE_Solver solver, int cycle_type, int coarsen_type, int agg_levels, int relax_type, HYPRE_Real strength_threshold, int interp_type, int Pmax)

(Optional) Sets AMS parameters for B_C. The defaults are 11, 10, 1, 3, 0.25, 0, 0. Note that cycle_type should be greater than 10, unless the high-order interface of HYPRE_ADSSetInterpolations is being used! See the user's manual for more details.

6.7.16

int
HYPRE_ADSSetAMGOptions (HYPRE_Solver solver, int coarsen_type, int agg_levels, int relax_type, HYPRE_Real strength_threshold, int interp_type, int Pmax)

(Optional) Sets AMG parameters for B_Π. The defaults are 10, 1, 3, 0.25, 0, 0. See the user's manual for more details.

6.7.17

int
HYPRE_ADSGetNumIterations (HYPRE_Solver solver, int* num_iterations)

Returns the number of iterations taken

6.7.18

int
HYPRE_ADSGetFinalRelativeResidualNorm (HYPRE_Solver solver, HYPRE_Real* rel_resid_norm)

Returns the norm of the final relative residual

6.8

ParCSR PCG Solver

Names

6.8.1 int

HYPRE_ParCSRPCGCreate (MPI_Comm comm, HYPRE_Solver* solver)
Create a solver object 194

6.8.2 int
HYPRE_ParCSRPCGDestroy (HYPRE_Solver solver)
Destroy a solver object 194

6.8.3 int
HYPRE_ParCSRDiagScaleSetup (HYPRE_Solver solver, HYPRE_ParCSRMatrix A, HYPRE_ParVector y, HYPRE_ParVector x)
Setup routine for diagonal preconditioning 194

6.8.4 int
HYPRE_ParCSRDiagScale (HYPRE_Solver solver, HYPRE_ParCSRMatrix HA, HYPRE_ParVector Hy, HYPRE_ParVector Hx)
Solve routine for diagonal preconditioning 195

These routines should be used in conjunction with the generic interface in PCG Solver.

6.8.1

int **HYPRE_ParCSRPCGCreate** (MPI_Comm comm, HYPRE_Solver* solver)

Create a solver object

6.8.2

int **HYPRE_ParCSRPCGDestroy** (HYPRE_Solver solver)

Destroy a solver object

6.8.3

int
HYPRE_ParCSRDiagScaleSetup (HYPRE_Solver solver, HYPRE_ParCSRMatrix A, HYPRE_ParVector y, HYPRE_ParVector x)

Setup routine for diagonal preconditioning

6.8.4

int
HYPRE_ParCSRDiagScale (HYPRE_Solver solver, HYPRE_ParCSRMatrix HA, HYPRE_ParVector Hy, HYPRE_ParVector Hx)

Solve routine for diagonal preconditioning

6.9

ParCSR GMRES Solver

Names

6.9.1 int
HYPRE_ParCSRGMRESCreate (MPI_Comm comm, HYPRE_Solver* solver)
Create a solver object 195

6.9.2 int
HYPRE_ParCSRGMRESDestroy (HYPRE_Solver solver)
Destroy a solver object 196

These routines should be used in conjunction with the generic interface in GMRES Solver.

6.9.1

int
HYPRE_ParCSRGMRESCreate (MPI_Comm comm, HYPRE_Solver* solver)

Create a solver object

6.9.2

int **HYPRE_ParCSRGMRESDestroy** (HYPRE_Solver solver)

Destroy a solver object

6.10

ParCSR FlexGMRES Solver

Names

6.10.1 int
HYPRE_ParCSRFlexGMRESCreate (MPI_Comm comm, HYPRE_Solver* solver)
Create a solver object 196

6.10.2 int
HYPRE_ParCSRFlexGMRESDestroy (HYPRE_Solver solver)
Destroy a solver object 196

These routines should be used in conjunction with the generic interface in FlexGMRES Solver.

6.10.1

int
HYPRE_ParCSRFlexGMRESCreate (MPI_Comm comm, HYPRE_Solver* solver)

Create a solver object

6.10.2

int **HYPRE_ParCSRFlexGMRESDestroy** (HYPRE_Solver solver)

Destroy a solver object

6.11

ParCSR LGMRES Solver

Names

6.11.1 int
HYPRE_ParCSRLGMRESCreate (MPI_Comm comm,
HYPRE_Solver* solver)
Create a solver object 197

6.11.2 int
HYPRE_ParCSRLGMRESDestroy (HYPRE_Solver solver)
Destroy a solver object 197

These routines should be used in conjunction with the generic interface in LGMRES Solver.

6.11.1

int **HYPRE_ParCSRLGMRESCreate** (MPI_Comm comm, HYPRE_Solver* solver)

Create a solver object

6.11.2

int **HYPRE_ParCSRLGMRESDestroy** (HYPRE_Solver solver)

Destroy a solver object

6.12

ParCSR BiCGSTAB Solver

Names

6.12.1 int
HYPRE_ParCSRBiCGSTABCreate (MPI_Comm comm,
HYPRE_Solver* solver)
Create a solver object 198

6.12.2 int

HYPRE_ParCSRBiCGSTABDestroy (HYPRE_Solver solver)
Destroy a solver object 198

These routines should be used in conjunction with the generic interface in BiCGSTAB Solver.

6.12.1

int
HYPRE_ParCSRBiCGSTABCreate (MPI_Comm comm, HYPRE_Solver* solver)

Create a solver object

6.12.2

int **HYPRE_ParCSRBiCGSTABDestroy** (HYPRE_Solver solver)

Destroy a solver object

6.13 ParCSR Hybrid Solver

Names

6.13.1 int
HYPRE_ParCSRHybridCreate (HYPRE_Solver* solver)
Create solver object 202

6.13.2 int
HYPRE_ParCSRHybridDestroy (HYPRE_Solver solver)
Destroy solver object 203

6.13.3 int
HYPRE_ParCSRHybridSetup (HYPRE_Solver solver,
HYPRE_ParCSRMatrix A,
HYPRE_ParVector b, HYPRE_ParVector x)
Setup the hybrid solver 203

6.13.4 int

HYPRE_ParCSRHybridSolve (HYPRE_Solver solver, HYPRE_ParCSRMatrix A, HYPRE_ParVector b, HYPRE_ParVector x)
Solve linear system 203

6.13.5 int
HYPRE_ParCSRHybridSetTol (HYPRE_Solver solver, HYPRE_Real tol)
Set the convergence tolerance for the Krylov solver. 204

6.13.6 int
HYPRE_ParCSRHybridSetAbsoluteTol (HYPRE_Solver solver, HYPRE_Real tol)
Set the absolute convergence tolerance for the Krylov solver. 204

6.13.7 int
HYPRE_ParCSRHybridSetConvergenceTol (HYPRE_Solver solver, HYPRE_Real cf_tol)
Set the desired convergence factor 204

6.13.8 int
HYPRE_ParCSRHybridSetDSCGMaxIter (HYPRE_Solver solver, int dscg_max_its)
Set the maximal number of iterations for the diagonally preconditioned solver 204

6.13.9 int
HYPRE_ParCSRHybridSetPCGMaxIter (HYPRE_Solver solver, int pcg_max_its)
Set the maximal number of iterations for the AMG preconditioned solver . 205

6.13.10 int
HYPRE_ParCSRHybridSetSolverType (HYPRE_Solver solver, int solver_type)
Set the desired solver type. 205

6.13.11 int
HYPRE_ParCSRHybridSetKDim (HYPRE_Solver solver, int k_dim)
Set the Krylov dimension for restarted GMRES. 205

6.13.12 int
HYPRE_ParCSRHybridSetTwoNorm (HYPRE_Solver solver, int two_norm)
Set the type of norm for PCG 205

6.13.13 int
HYPRE_ParCSRHybridSetPrecond (HYPRE_Solver solver, HYPRE_PtrToParSolverFcn precond, HYPRE_PtrToParSolverFcn precond_setup, HYPRE_Solver precond_solver)
Set preconditioner if wanting to use one that is not set up by the hybrid solver 206

6.13.14 int
HYPRE_ParCSRHybridSetLogging (HYPRE_Solver solver, int logging)
Set logging parameter (default: 0, no logging) 206

6.13.15 int

HYPRE_ParCSRHybridSetPrintLevel (HYPRE_Solver solver, int print_level)
Set print level (default: 0, no printing) 206

6.13.16 int **HYPRE_ParCSRHybridSetStrongThreshold** (HYPRE_Solver solver, HYPRE_Real strong_threshold)
(Optional) Sets AMG strength threshold. 206

6.13.17 int **HYPRE_ParCSRHybridSetMaxRowSum** (HYPRE_Solver solver, HYPRE_Real max_row_sum)
(Optional) Sets a parameter to modify the definition of strength for diagonal dominant portions of the matrix. 207

6.13.18 int **HYPRE_ParCSRHybridSetTruncFactor** (HYPRE_Solver solver, HYPRE_Real trunc_factor)
(Optional) Defines a truncation factor for the interpolation. 207

6.13.19 int **HYPRE_ParCSRHybridSetPMaxElmts** (HYPRE_Solver solver, int P_max_elmts)
(Optional) Defines the maximal number of elements per row for the interpolation. 207

6.13.20 int **HYPRE_ParCSRHybridSetMaxLevels** (HYPRE_Solver solver, int max_levels)
(Optional) Defines the maximal number of levels used for AMG. 207

6.13.21 int **HYPRE_ParCSRHybridSetMeasureType** (HYPRE_Solver solver, int measure_type)
(Optional) Defines whether local or global measures are used 208

6.13.22 int **HYPRE_ParCSRHybridSetCoarsenType** (HYPRE_Solver solver, int coarsen_type)
(Optional) Defines which parallel coarsening algorithm is used. 208

6.13.23 int **HYPRE_ParCSRHybridSetCycleType** (HYPRE_Solver solver, int cycle_type)
(Optional) Defines the type of cycle. 208

6.13.24 int **HYPRE_ParCSRHybridSetNumSweeps** (HYPRE_Solver solver, int num_sweeps)
(Optional) Sets the number of sweeps. 209

6.13.25 int **HYPRE_ParCSRHybridSetCycleNumSweeps** (HYPRE_Solver solver, int num_sweeps, int k)
(Optional) Sets the number of sweeps at a specified cycle. 209

6.13.26 int

HYPRE_ParCSRHybridSetRelaxType (HYPRE_Solver solver,
int relax_type)
(Optional) Defines the smoother to be used. 209

6.13.27 int
HYPRE_ParCSRHybridSetCycleRelaxType (HYPRE_Solver solver,
int relax_type, int k)
(Optional) Defines the smoother at a given cycle. 210

6.13.28 int
HYPRE_ParCSRHybridSetRelaxOrder (HYPRE_Solver solver,
int relax_order)
(Optional) Defines in which order the points are relaxed. 210

6.13.29 int
HYPRE_ParCSRHybridSetRelaxWt (HYPRE_Solver solver,
HYPRE_Real relax_wt)
(Optional) Defines the relaxation weight for smoothed Jacobi and hybrid SOR on all levels. 210

6.13.30 int
HYPRE_ParCSRHybridSetLevelRelaxWt (HYPRE_Solver solver,
HYPRE_Real relax_wt,
int level)
(Optional) Defines the relaxation weight for smoothed Jacobi and hybrid SOR on the user defined level. 211

6.13.31 int
HYPRE_ParCSRHybridSetOuterWt (HYPRE_Solver solver,
HYPRE_Real outer_wt)
(Optional) Defines the outer relaxation weight for hybrid SOR and SSOR on all levels. 211

6.13.32 int
HYPRE_ParCSRHybridSetLevelOuterWt (HYPRE_Solver solver,
HYPRE_Real outer_wt,
int level)
(Optional) Defines the outer relaxation weight for hybrid SOR or SSOR on the user defined level. 211

6.13.33 int
HYPRE_ParCSRHybridSetMaxCoarseSize (HYPRE_Solver solver,
int max_coarse_size)
(Optional) Defines the maximal coarse grid size. 212

6.13.34 int
HYPRE_ParCSRHybridSetMinCoarseSize (HYPRE_Solver solver,
int min_coarse_size)
(Optional) Defines the minimal coarse grid size. 212

6.13.35 int
HYPRE_ParCSRHybridSetSeqThreshold (HYPRE_Solver solver,
int seq_threshold)
(Optional) enables redundant coarse grid size. 212

6.13.36 int

HYPRE_ParCSRHybridSetAggNumLevels (HYPRE_Solver solver, int agg_num_levels)
(Optional) Defines the number of levels of aggressive coarsening, starting with the finest level. 212

6.13.37 int **HYPRE_ParCSRHybridSetNumPaths** (HYPRE_Solver solver, int num_paths)
(Optional) Defines the degree of aggressive coarsening. 213

6.13.38 int **HYPRE_ParCSRHybridSetNumFunctions** (HYPRE_Solver solver, int num_functions)
(Optional) Sets the size of the system of PDEs, if using the systems version. 213

6.13.39 int **HYPRE_ParCSRHybridSetDofFunc** (HYPRE_Solver solver, int* dof_func)
(Optional) Sets the mapping that assigns the function to each variable, if using the systems version. 213

6.13.40 int **HYPRE_ParCSRHybridSetNodal** (HYPRE_Solver solver, int nodal)
(Optional) Sets whether to use the nodal systems version. 213

6.13.41 int **HYPRE_ParCSRHybridGetNumIterations** (HYPRE_Solver solver, int* num_its)
Retrieves the total number of iterations 214

6.13.42 int **HYPRE_ParCSRHybridGetDSCGNumIterations** (HYPRE_Solver solver, int* dscg_num_its)
Retrieves the number of iterations used by the diagonally scaled solver ... 214

6.13.43 int **HYPRE_ParCSRHybridGetPCGNumIterations** (HYPRE_Solver solver, int* pcg_num_its)
Retrieves the number of iterations used by the AMG preconditioned solver 214

6.13.44 int **HYPRE_ParCSRHybridGetFinalRelativeResidualNorm** (HYPRE_Solver solver, HYPRE_Real* norm)
Retrieves the final relative residual norm 214

6.13.1

int **HYPRE_ParCSRHybridCreate** (HYPRE_Solver* solver)

Create solver object

6.13.2

int **HYPRE_ParCSRHybridDestroy** (HYPRE_Solver solver)

Destroy solver object

6.13.3

int
HYPRE_ParCSRHybridSetup (HYPRE_Solver solver, HYPRE_ParCSRMatrix A, HYPRE_ParVector b, HYPRE_ParVector x)

Setup the hybrid solver

Parameters:

`solver`	[IN] object to be set up.
`A`	[IN] ParCSR matrix used to construct the solver/preconditioner.
`b`	Ignored by this function.
`x`	Ignored by this function.

6.13.4

int
HYPRE_ParCSRHybridSolve (HYPRE_Solver solver, HYPRE_ParCSRMatrix A, HYPRE_ParVector b, HYPRE_ParVector x)

Solve linear system

Parameters:

`solver`	[IN] solver or preconditioner object to be applied.
`A`	[IN] ParCSR matrix, matrix of the linear system to be solved
`b`	[IN] right hand side of the linear system to be solved
`x`	[OUT] approximated solution of the linear system to be solved

6.13.5

int **HYPRE_ParCSRHybridSetTol** (HYPRE_Solver solver, HYPRE_Real tol)

Set the convergence tolerance for the Krylov solver. The default is 1.e-7.

6.13.6

int
HYPRE_ParCSRHybridSetAbsoluteTol (HYPRE_Solver solver, HYPRE_Real tol)

Set the absolute convergence tolerance for the Krylov solver. The default is 0.

6.13.7

int
HYPRE_ParCSRHybridSetConvergenceTol (HYPRE_Solver solver, HYPRE_Real cf_tol)

Set the desired convergence factor

6.13.8

int
HYPRE_ParCSRHybridSetDSCGMaxIter (HYPRE_Solver solver, int dscg_max_its)

Set the maximal number of iterations for the diagonally preconditioned solver

6.13.9

int
HYPRE_ParCSRHybridSetPCGMaxIter (HYPRE_Solver solver, int pcg_max_its)

Set the maximal number of iterations for the AMG preconditioned solver

6.13.10

int
HYPRE_ParCSRHybridSetSolverType (HYPRE_Solver solver, int solver_type)

Set the desired solver type. There are the following options:

1	PCG (default)
2	GMRES
3	BiCGSTAB

6.13.11

int **HYPRE_ParCSRHybridSetKDim** (HYPRE_Solver solver, int k_dim)

Set the Krylov dimension for restarted GMRES. The default is 5.

6.13.12

int
HYPRE_ParCSRHybridSetTwoNorm (HYPRE_Solver solver, int two_norm)

Set the type of norm for PCG

6.13.13

int
HYPRE_ParCSRHybridSetPrecond (HYPRE_Solver solver, HYPRE_PtrToParSolverFcn precond, HYPRE_PtrToParSolverFcn precond_setup, HYPRE_Solver precond_solver)

Set preconditioner if wanting to use one that is not set up by the hybrid solver

6.13.14

int **HYPRE_ParCSRHybridSetLogging** (HYPRE_Solver solver, int logging)

Set logging parameter (default: 0, no logging)

6.13.15

int
HYPRE_ParCSRHybridSetPrintLevel (HYPRE_Solver solver, int print_level)

Set print level (default: 0, no printing)

6.13.16

int
HYPRE_ParCSRHybridSetStrongThreshold (HYPRE_Solver solver, HYPRE_Real strong_threshold)

(Optional) Sets AMG strength threshold. The default is 0.25. For 2d Laplace operators, 0.25 is a good value, for 3d Laplace operators, 0.5 or 0.6 is a better value. For elasticity problems, a large strength threshold, such as 0.9, is often better.

6.13.17

int
HYPRE_ParCSRHybridSetMaxRowSum (HYPRE_Solver solver, HYPRE_Real max_row_sum)

(Optional) Sets a parameter to modify the definition of strength for diagonal dominant portions of the matrix. The default is 0.9. If max_row_sum is 1, no checking for diagonally dominant rows is performed.

6.13.18

int
HYPRE_ParCSRHybridSetTruncFactor (HYPRE_Solver solver, HYPRE_Real trunc_factor)

(Optional) Defines a truncation factor for the interpolation. The default is 0.

6.13.19

int
HYPRE_ParCSRHybridSetPMaxElmts (HYPRE_Solver solver, int P_max_elmts)

(Optional) Defines the maximal number of elements per row for the interpolation. The default is 0.

6.13.20

int
HYPRE_ParCSRHybridSetMaxLevels (HYPRE_Solver solver, int max_levels)

(Optional) Defines the maximal number of levels used for AMG. The default is 25.

6.13.21

int
HYPRE_ParCSRHybridSetMeasureType (HYPRE_Solver solver, int measure_type)

(Optional) Defines whether local or global measures are used

6.13.22

int
HYPRE_ParCSRHybridSetCoarsenType (HYPRE_Solver solver, int coarsen_type)

(Optional) Defines which parallel coarsening algorithm is used. There are the following options for coarsen_type:

0	CLJP-coarsening (a parallel coarsening algorithm using independent sets).
1	classical Ruge-Stueben coarsening on each processor, no boundary treatment
3	classical Ruge-Stueben coarsening on each processor, followed by a third pass, which adds coarse points on the boundaries
6	Falgout coarsening (uses 1 first, followed by CLJP using the interior coarse points generated by 1 as its first independent set)
7	CLJP-coarsening (using a fixed random vector, for debugging purposes only)
8	PMIS-coarsening (a parallel coarsening algorithm using independent sets with lower complexities than CLJP, might also lead to slower convergence)
9	PMIS-coarsening (using a fixed random vector, for debugging purposes only)
10	HMIS-coarsening (uses one pass Ruge-Stueben on each processor independently, followed by PMIS using the interior C-points as its first independent set)
11	one-pass Ruge-Stueben coarsening on each processor, no boundary treatment

The default is 6.

6.13.23

int
HYPRE_ParCSRHybridSetCycleType (HYPRE_Solver solver, int cycle_type)

(Optional) Defines the type of cycle. For a V-cycle, set cycle_type to 1, for a W-cycle set cycle_type to 2. The default is 1.

6.13.24

int
HYPRE_ParCSRHybridSetNumSweeps (HYPRE_Solver solver, int num_sweeps)

(Optional) Sets the number of sweeps. On the finest level, the up and the down cycle the number of sweeps are set to num_sweeps and on the coarsest level to 1. The default is 1.

6.13.25

int
HYPRE_ParCSRHybridSetCycleNumSweeps (HYPRE_Solver solver, int num_sweeps, int k)

(Optional) Sets the number of sweeps at a specified cycle. There are the following options for k:

the down cycle	if k=1
the up cycle	if k=2
the coarsest level	if k=3.

6.13.26

int
HYPRE_ParCSRHybridSetRelaxType (HYPRE_Solver solver, int relax_type)

(Optional) Defines the smoother to be used. It uses the given smoother on the fine grid, the up and the down cycle and sets the solver on the coarsest level to Gaussian elimination (9). The default is Gauss-Seidel (3).

There are the following options for relax_type:

0	Jacobi
1	Gauss-Seidel, sequential (very slow!)
2	Gauss-Seidel, interior points in parallel, boundary sequential (slow!)
3	hybrid Gauss-Seidel or SOR, forward solve
4	hybrid Gauss-Seidel or SOR, backward solve
5	hybrid chaotic Gauss-Seidel (works only with OpenMP)
6	hybrid symmetric Gauss-Seidel or SSOR
9	Gaussian elimination (only on coarsest level)

6.13.27

int
HYPRE_ParCSRHybridSetCycleRelaxType (HYPRE_Solver solver, int relax_type, int k)

(Optional) Defines the smoother at a given cycle. For options of relax_type see description of HYPRE_BoomerAMGSetRelaxType). Options for k are

the down cycle	if k=1
the up cycle	if k=2
the coarsest level	if k=3.

6.13.28

int
HYPRE_ParCSRHybridSetRelaxOrder (HYPRE_Solver solver, int relax_order)

(Optional) Defines in which order the points are relaxed. There are the following options for relax_order:

0	the points are relaxed in natural or lexicographic order on each processor
1	CF-relaxation is used, i.e on the fine grid and the down cycle the coarse points are relaxed first, followed by the fine points; on the up cycle the F-points are relaxed first, followed by the C-points. On the coarsest level, if an iterative scheme is used, the points are relaxed in lexicographic order.

The default is 1 (CF-relaxation).

6.13.29

int
HYPRE_ParCSRHybridSetRelaxWt (HYPRE_Solver solver, HYPRE_Real relax_wt)

(Optional) Defines the relaxation weight for smoothed Jacobi and hybrid SOR on all levels.

relax_weight > 0	this assigns the given relaxation weight on all levels
relax_weight = 0	the weight is determined on each level with the estimate $\frac{3}{4\|D^{-1/2}AD^{-1/2}\|}$, where D is the diagonal matrix of A (this should only be used with Jacobi)
relax_weight = -k	the relaxation weight is determined with at most k CG steps on each level this should only be used for symmetric positive definite problems)

The default is 1.

6.13.30

int
HYPRE_ParCSRHybridSetLevelRelaxWt (HYPRE_Solver solver, HYPRE_Real relax_wt, int level)

(Optional) Defines the relaxation weight for smoothed Jacobi and hybrid SOR on the user defined level. Note that the finest level is denoted 0, the next coarser level 1, etc. For nonpositive relax_weight, the parameter is determined on the given level as described for HYPRE_BoomerAMGSetRelaxWt. The default is 1.

6.13.31

int
HYPRE_ParCSRHybridSetOuterWt (HYPRE_Solver solver, HYPRE_Real outer_wt)

(Optional) Defines the outer relaxation weight for hybrid SOR and SSOR on all levels.

omega > 0	this assigns the same outer relaxation weight omega on each level
omega = -k	an outer relaxation weight is determined with at most k CG steps on each level
	(this only makes sense for symmetric positive definite problems and smoothers, e.g. SSOR)

The default is 1.

6.13.32

int
HYPRE_ParCSRHybridSetLevelOuterWt (HYPRE_Solver solver, HYPRE_Real outer_wt, int level)

(Optional) Defines the outer relaxation weight for hybrid SOR or SSOR on the user defined level. Note that the finest level is denoted 0, the next coarser level 1, etc. For nonpositive omega, the parameter is determined on the given level as described for HYPRE_BoomerAMGSetOuterWt. The default is 1.

6.13.33

int
HYPRE_ParCSRHybridSetMaxCoarseSize (HYPRE_Solver solver, int max_coarse_size)

(Optional) Defines the maximal coarse grid size. The default is 9.

6.13.34

int
HYPRE_ParCSRHybridSetMinCoarseSize (HYPRE_Solver solver, int min_coarse_size)

(Optional) Defines the minimal coarse grid size. The default is 0.

6.13.35

int
HYPRE_ParCSRHybridSetSeqThreshold (HYPRE_Solver solver, int seq_threshold)

(Optional) enables redundant coarse grid size. If the system size becomes smaller than seq_threshold, sequential AMG is used on all remaining processors. The default is 0.

6.13.36

int
HYPRE_ParCSRHybridSetAggNumLevels (HYPRE_Solver solver, int agg_num_levels)

(Optional) Defines the number of levels of aggressive coarsening, starting with the finest level. The default is 0, i.e. no aggressive coarsening.

6.13.37

int
HYPRE_ParCSRHybridSetNumPaths (HYPRE_Solver solver, int num_paths)

(Optional) Defines the degree of aggressive coarsening. The default is 1, which leads to the most aggressive coarsening. Setting num_paths to 2 will increase complexity somewhat, but can lead to better convergence.*

6.13.38

int
HYPRE_ParCSRHybridSetNumFunctions (HYPRE_Solver solver, int num_functions)

(Optional) Sets the size of the system of PDEs, if using the systems version. The default is 1.

6.13.39

int **HYPRE_ParCSRHybridSetDofFunc** (HYPRE_Solver solver, int* dof_func)

(Optional) Sets the mapping that assigns the function to each variable, if using the systems version. If no assignment is made and the number of functions is k > 1, the mapping generated is (0,1,...,k-1,0,1,...,k-1,...).

6.13.40

int **HYPRE_ParCSRHybridSetNodal** (HYPRE_Solver solver, int nodal)

(Optional) Sets whether to use the nodal systems version. The default is 0 (the unknown based approach).

6.13.41

int
HYPRE_ParCSRHybridGetNumIterations (HYPRE_Solver solver, int* num_its)

Retrieves the total number of iterations

6.13.42

int
HYPRE_ParCSRHybridGetDSCGNumIterations (HYPRE_Solver solver, int* dscg_num_its)

Retrieves the number of iterations used by the diagonally scaled solver

6.13.43

int
HYPRE_ParCSRHybridGetPCGNumIterations (HYPRE_Solver solver, int* pcg_num_its)

Retrieves the number of iterations used by the AMG preconditioned solver

6.13.44

int
HYPRE_ParCSRHybridGetFinalRelativeResidualNorm (HYPRE_Solver solver, HYPRE_Real* norm)

Retrieves the final relative residual norm

6.14 ParCSR LOBPCG Eigensolver

Names

6.14.1 int
HYPRE_ParCSRSetupInterpreter (mv_InterfaceInterpreter* i)
Load interface interpreter. 215

6.14.2 int
HYPRE_ParCSRSetupMatvec (HYPRE_MatvecFunctions* mv)
Load Matvec interpreter with hypre_ParKrylov functions 215

These routines should be used in conjunction with the generic interface in LOBPCG Eigensolver.

6.14.1 int **HYPRE_ParCSRSetupInterpreter** (mv_InterfaceInterpreter* i)

Load interface interpreter. Vector part loaded with hypre_ParKrylov functions and multivector part loaded with mv_TempMultiVector functions.

6.14.2 int **HYPRE_ParCSRSetupMatvec** (HYPRE_MatvecFunctions* mv)

Load Matvec interpreter with hypre_ParKrylov functions

7

Krylov Solvers

Names

7.1 **Krylov Solvers** 216

7.2 **PCG Solver** 217

7.3 **GMRES Solver** 225

7.4 **FlexGMRES Solver** 232

7.5 **LGMRES Solver** 238

7.6 **BiCGSTAB Solver** 244

7.7 **CGNR Solver** 249

These solvers support many of the matrix/vector storage schemes in hypre. They should be used in conjunction with the storage-specific interfaces, particularly the specific Create() and Destroy() functions.

7.1

Krylov Solvers

Names

7.1.1 typedef struct hypre_Solver_struct ***HYPRE_Solver**
The solver object 217

7.1.2 typedef struct hypre_Matrix_struct ***HYPRE_Matrix**
The matrix object 217

7.1.3 typedef struct hypre_Vector_struct ***HYPRE_Vector**
The vector object 217

7.1.1

typedef struct hypre_Solver_struct ***HYPRE_Solver**

The solver object

7.1.2

typedef struct hypre_Matrix_struct ***HYPRE_Matrix**

The matrix object

7.1.3

typedef struct hypre_Vector_struct ***HYPRE_Vector**

The vector object

7.2 PCG Solver

Names

7.2.1 int
HYPRE_PCGSetup (HYPRE_Solver solver, HYPRE_Matrix A, HYPRE_Vector b, HYPRE_Vector x)
Prepare to solve the system. 219

7.2.2 int
HYPRE_PCGSolve (HYPRE_Solver solver, HYPRE_Matrix A, HYPRE_Vector b, HYPRE_Vector x)
Solve the system 219

7.2.3 int
HYPRE_PCGSetTol (HYPRE_Solver solver, HYPRE_Real tol)
(Optional) Set the relative convergence tolerance 220

7.2.4 int

HYPRE_PCGSetAbsoluteTol (HYPRE_Solver solver, HYPRE_Real a_tol)
(Optional) Set the absolute convergence tolerance (default is 0). 220

7.2.5 int
HYPRE_PCGSetResidualTol (HYPRE_Solver solver, HYPRE_Real rtol)
(Optional) Set a residual-based convergence tolerance which checks if $\|r_{old} - r_{new}\| < rtol\|b\|$. 220

7.2.6 int
HYPRE_PCGSetMaxIter (HYPRE_Solver solver, int max_iter)
(Optional) Set maximum number of iterations 220

7.2.7 int
HYPRE_PCGSetTwoNorm (HYPRE_Solver solver, int two_norm)
(Optional) Use the two-norm in stopping criteria 221

7.2.8 int
HYPRE_PCGSetRelChange (HYPRE_Solver solver, int rel_change)
(Optional) Additionally require that the relative difference in successive iterates be small 221

7.2.9 int
HYPRE_PCGSetRecomputeResidual (HYPRE_Solver solver, int recompute_residual)
(Optional) Recompute the residual at the end to double-check convergence 221

7.2.10 int
HYPRE_PCGSetRecomputeResidualP (HYPRE_Solver solver, int recompute_residual_p)
(Optional) Periodically recompute the residual while iterating 221

7.2.11 int
HYPRE_PCGSetPrecond (HYPRE_Solver solver, HYPRE_PtrToSolverFcn precond, HYPRE_PtrToSolverFcn precond_setup, HYPRE_Solver precond_solver)
(Optional) Set the preconditioner to use 222

7.2.12 int
HYPRE_PCGSetLogging (HYPRE_Solver solver, int logging)
(Optional) Set the amount of logging to do 222

7.2.13 int
HYPRE_PCGSetPrintLevel (HYPRE_Solver solver, int level)
(Optional) Set the amount of printing to do to the screen 222

7.2.14 int
HYPRE_PCGGetNumIterations (HYPRE_Solver solver, int* num_iterations)
Return the number of iterations taken 222

7.2.15 int
HYPRE_PCGGetFinalRelativeResidualNorm (HYPRE_Solver solver, HYPRE_Real* norm)
Return the norm of the final relative residual 223

7.2.16 int
HYPRE_PCGGetResidual (HYPRE_Solver solver, void** residual)
Return the residual 223

7.2.17 int

HYPRE_PCGGetTol (HYPRE_Solver solver, HYPRE_Real* tol) 223

7.2.18 int
HYPRE_PCGGetResidualTol (HYPRE_Solver solver, HYPRE_Real* rtol) ... 223

7.2.19 int
HYPRE_PCGGetMaxIter (HYPRE_Solver solver, int* max_iter) 223

7.2.20 int
HYPRE_PCGGetTwoNorm (HYPRE_Solver solver, int* two_norm) 224

7.2.21 int
HYPRE_PCGGetRelChange (HYPRE_Solver solver, int* rel_change) 224

7.2.22 int
HYPRE_GMRESGetSkipRealResidualCheck (HYPRE_Solver solver, int* skip_real_r_check) 224

7.2.23 int
HYPRE_PCGGetPrecond (HYPRE_Solver solver, HYPRE_Solver* precond_data_ptr) 224

7.2.24 int
HYPRE_PCGGetLogging (HYPRE_Solver solver, int* level) 224

7.2.25 int
HYPRE_PCGGetPrintLevel (HYPRE_Solver solver, int* level) 225

7.2.26 int
HYPRE_PCGGetConverged (HYPRE_Solver solver, int* converged) 225

7.2.1

int
HYPRE_PCGSetup (HYPRE_Solver solver, HYPRE_Matrix A, HYPRE_Vector b, HYPRE_Vector x)

Prepare to solve the system. The coefficient data in `b` and `x` is ignored here, but information about the layout of the data may be used.

7.2.2

int
HYPRE_PCGSolve (HYPRE_Solver solver, HYPRE_Matrix A, HYPRE_Vector b, HYPRE_Vector x)

Solve the system

7.2.3

int **HYPRE_PCGSetTol** (HYPRE_Solver solver, HYPRE_Real tol)

(Optional) Set the relative convergence tolerance

7.2.4

int **HYPRE_PCGSetAbsoluteTol** (HYPRE_Solver solver, HYPRE_Real a_tol)

(Optional) Set the absolute convergence tolerance (default is 0). If one desires the convergence test to check the absolute convergence tolerance *only*, then set the relative convergence tolerance to 0.0. (The default convergence test is $< C*r, r > \leq$ max(relative_tolerance$^2 * < C*b, b >$, absolute_tolerance^2).)

7.2.5

int **HYPRE_PCGSetResidualTol** (HYPRE_Solver solver, HYPRE_Real rtol)

(Optional) Set a residual-based convergence tolerance which checks if $\|r_{old} - r_{new}\| < rtol\|b\|$. This is useful when trying to converge to very low relative and/or absolute tolerances, in order to bail-out before roundoff errors affect the approximation.

7.2.6

int **HYPRE_PCGSetMaxIter** (HYPRE_Solver solver, int max_iter)

(Optional) Set maximum number of iterations

7.2.7

int **HYPRE_PCGSetTwoNorm** (HYPRE_Solver solver, int two_norm)

(Optional) Use the two-norm in stopping criteria

7.2.8

int **HYPRE_PCGSetRelChange** (HYPRE_Solver solver, int rel_change)

(Optional) Additionally require that the relative difference in successive iterates be small

7.2.9

int
HYPRE_PCGSetRecomputeResidual (HYPRE_Solver solver, int recompute_residual)

(Optional) Recompute the residual at the end to double-check convergence

7.2.10

int
HYPRE_PCGSetRecomputeResidualP (HYPRE_Solver solver, int recompute_residual_p)

(Optional) Periodically recompute the residual while iterating

7.2.11

int
HYPRE_PCGSetPrecond (HYPRE_Solver solver, HYPRE_PtrToSolverFcn precond, HYPRE_PtrToSolverFcn precond_setup, HYPRE_Solver precond_solver)

(Optional) Set the preconditioner to use

7.2.12

int **HYPRE_PCGSetLogging** (HYPRE_Solver solver, int logging)

(Optional) Set the amount of logging to do

7.2.13

int **HYPRE_PCGSetPrintLevel** (HYPRE_Solver solver, int level)

(Optional) Set the amount of printing to do to the screen

7.2.14

int
HYPRE_PCGGetNumIterations (HYPRE_Solver solver, int* num_iterations)

Return the number of iterations taken

7.2.15

int
HYPRE_PCGGetFinalRelativeResidualNorm (HYPRE_Solver solver, HYPRE_Real* norm)

Return the norm of the final relative residual

7.2.16

int **HYPRE_PCGGetResidual** (HYPRE_Solver solver, void** residual)

Return the residual

7.2.17

int **HYPRE_PCGGetTol** (HYPRE_Solver solver, HYPRE_Real* tol)

7.2.18

int **HYPRE_PCGGetResidualTol** (HYPRE_Solver solver, HYPRE_Real* rtol)

7.2.19

int **HYPRE_PCGGetMaxIter** (HYPRE_Solver solver, int* max_iter)

7.2.20

int **HYPRE_PCGGetTwoNorm** (HYPRE_Solver solver, int* two_norm)

7.2.21

int **HYPRE_PCGGetRelChange** (HYPRE_Solver solver, int* rel_change)

7.2.22

int
HYPRE_GMRESGetSkipRealResidualCheck (HYPRE_Solver solver, int*
skip_real_r_check)

7.2.23

int
HYPRE_PCGGetPrecond (HYPRE_Solver solver, HYPRE_Solver*
precond_data_ptr)

7.2.24

int **HYPRE_PCGGetLogging** (HYPRE_Solver solver, int* level)

7.2.25

int **HYPRE_PCGGetPrintLevel** (HYPRE_Solver solver, int* level)

7.2.26

int **HYPRE_PCGGetConverged** (HYPRE_Solver solver, int* converged)

7.3

GMRES Solver

Names

7.3.1 int
HYPRE_GMRESSetup (HYPRE_Solver solver, HYPRE_Matrix A, HYPRE_Vector b, HYPRE_Vector x)
Prepare to solve the system. 227

7.3.2 int
HYPRE_GMRESSolve (HYPRE_Solver solver, HYPRE_Matrix A, HYPRE_Vector b, HYPRE_Vector x)
Solve the system 227

7.3.3 int
HYPRE_GMRESSetTol (HYPRE_Solver solver, HYPRE_Real tol)
(Optional) Set the relative convergence tolerance 227

7.3.4 int
HYPRE_GMRESSetAbsoluteTol (HYPRE_Solver solver, HYPRE_Real a_tol)
(Optional) Set the absolute convergence tolerance (default is 0). 227

7.3.5 int
HYPRE_GMRESSetMaxIter (HYPRE_Solver solver, int max_iter)
(Optional) Set maximum number of iterations 228

7.3.6 int
HYPRE_GMRESSetKDim (HYPRE_Solver solver, int k_dim)
(Optional) Set the maximum size of the Krylov space 228

7.3.7 int
HYPRE_GMRESSetRelChange (HYPRE_Solver solver, int rel_change)
(Optional) Additionally require that the relative difference in successive iterates be small 228

7.3.8 int

HYPRE_GMRESSetSkipRealResidualCheck (HYPRE_Solver solver, int skip_real_r_check)
(Optional) By default, hypre checks for convergence by evaluating the actual residual before returnig from GMRES (with restart if the true residual does not indicate convergence). 228

7.3.9 int **HYPRE_GMRESSetPrecond** (HYPRE_Solver solver, HYPRE_PtrToSolverFcn precond, HYPRE_PtrToSolverFcn precond_setup, HYPRE_Solver precond_solver)
(Optional) Set the preconditioner to use 229

7.3.10 int **HYPRE_GMRESSetLogging** (HYPRE_Solver solver, int logging)
(Optional) Set the amount of logging to do 229

7.3.11 int **HYPRE_GMRESSetPrintLevel** (HYPRE_Solver solver, int level)
(Optional) Set the amount of printing to do to the screen 229

7.3.12 int **HYPRE_GMRESGetNumIterations** (HYPRE_Solver solver, int* num_iterations)
Return the number of iterations taken 229

7.3.13 int **HYPRE_GMRESGetFinalRelativeResidualNorm** (HYPRE_Solver solver, HYPRE_Real* norm)
Return the norm of the final relative residual 230

7.3.14 int **HYPRE_GMRESGetResidual** (HYPRE_Solver solver, void** residual)
Return the residual 230

7.3.15 int **HYPRE_GMRESGetTol** (HYPRE_Solver solver, HYPRE_Real* tol) 230

7.3.16 int **HYPRE_GMRESGetAbsoluteTol** (HYPRE_Solver solver, HYPRE_Real* tol) 230

7.3.17 int **HYPRE_GMRESGetMaxIter** (HYPRE_Solver solver, int* max_iter) 230

7.3.18 int **HYPRE_GMRESGetKDim** (HYPRE_Solver solver, int* k_dim) 231

7.3.19 int **HYPRE_GMRESGetRelChange** (HYPRE_Solver solver, int* rel_change) 231

7.3.20 int **HYPRE_GMRESGetPrecond** (HYPRE_Solver solver, HYPRE_Solver* precond_data_ptr) 231

7.3.21 int **HYPRE_GMRESGetLogging** (HYPRE_Solver solver, int* level) 231

7.3.22 int **HYPRE_GMRESGetPrintLevel** (HYPRE_Solver solver, int* level) 231

7.3.23 int

HYPRE_GMRESGetConverged (HYPRE_Solver solver, int* converged) 231

7.3.1

int
HYPRE_GMRESSetup (HYPRE_Solver solver, HYPRE_Matrix A,
HYPRE_Vector b, HYPRE_Vector x)

Prepare to solve the system. The coefficient data in `b` and `x` is ignored here, but information about the layout of the data may be used.

7.3.2

int
HYPRE_GMRESSolve (HYPRE_Solver solver, HYPRE_Matrix A,
HYPRE_Vector b, HYPRE_Vector x)

Solve the system

7.3.3

int **HYPRE_GMRESSetTol** (HYPRE_Solver solver, HYPRE_Real tol)

(Optional) Set the relative convergence tolerance

7.3.4

int
HYPRE_GMRESSetAbsoluteTol (HYPRE_Solver solver, HYPRE_Real a_tol)

(Optional) Set the absolute convergence tolerance (default is 0). If one desires the convergence test to check the absolute convergence tolerance *only*, then set the relative convergence tolerance to 0.0. (The convergence test is $\|r\| \leq$ max(relative_tolerance$*\|b\|$, absolute_tolerance).)

7.3.5

int **HYPRE_GMRESSetMaxIter** (HYPRE_Solver solver, int max_iter)

(Optional) Set maximum number of iterations

7.3.6

int **HYPRE_GMRESSetKDim** (HYPRE_Solver solver, int k_dim)

(Optional) Set the maximum size of the Krylov space

7.3.7

int **HYPRE_GMRESSetRelChange** (HYPRE_Solver solver, int rel_change)

(Optional) Additionally require that the relative difference in successive iterates be small

7.3.8

int
HYPRE_GMRESSetSkipRealResidualCheck (HYPRE_Solver solver, int skip_real_r_check)

(Optional) By default, hypre checks for convergence by evaluating the actual residual before returnig from GMRES (with restart if the true residual does not indicate convergence). This option allows users to skip the evaluation and the check of the actual residual for badly conditioned problems where restart is not expected to be beneficial.

7.3.9

int
HYPRE_GMRESSetPrecond (HYPRE_Solver solver, HYPRE_PtrToSolverFcn precond, HYPRE_PtrToSolverFcn precond_setup, HYPRE_Solver precond_solver)

(Optional) Set the preconditioner to use

7.3.10

int **HYPRE_GMRESSetLogging** (HYPRE_Solver solver, int logging)

(Optional) Set the amount of logging to do

7.3.11

int **HYPRE_GMRESSetPrintLevel** (HYPRE_Solver solver, int level)

(Optional) Set the amount of printing to do to the screen

7.3.12

int
HYPRE_GMRESGetNumIterations (HYPRE_Solver solver, int* num_iterations)

Return the number of iterations taken

7.3.13

int
HYPRE_GMRESGetFinalRelativeResidualNorm (HYPRE_Solver solver, HYPRE_Real* norm)

Return the norm of the final relative residual

7.3.14

int **HYPRE_GMRESGetResidual** (HYPRE_Solver solver, void** residual)

Return the residual

7.3.15

int **HYPRE_GMRESGetTol** (HYPRE_Solver solver, HYPRE_Real* tol)

7.3.16

int
HYPRE_GMRESGetAbsoluteTol (HYPRE_Solver solver, HYPRE_Real* tol)

7.3.17

int **HYPRE_GMRESGetMaxIter** (HYPRE_Solver solver, int* max_iter)

7.3.18

int **HYPRE_GMRESGetKDim** (HYPRE_Solver solver, int* k_dim)

7.3.19

int **HYPRE_GMRESGetRelChange** (HYPRE_Solver solver, int* rel_change)

7.3.20

int
HYPRE_GMRESGetPrecond (HYPRE_Solver solver, HYPRE_Solver*
precond_data_ptr)

7.3.21

int **HYPRE_GMRESGetLogging** (HYPRE_Solver solver, int* level)

7.3.22

int **HYPRE_GMRESGetPrintLevel** (HYPRE_Solver solver, int* level)

7.3.23

int **HYPRE_GMRESGetConverged** (HYPRE_Solver solver, int* converged)

7.4

FlexGMRES Solver

Names

7.4.1 int
HYPRE_FlexGMRESSetup (HYPRE_Solver solver, HYPRE_Matrix A, HYPRE_Vector b, HYPRE_Vector x)
Prepare to solve the system. 233

7.4.2 int
HYPRE_FlexGMRESSolve (HYPRE_Solver solver, HYPRE_Matrix A, HYPRE_Vector b, HYPRE_Vector x)
Solve the system 234

7.4.3 int
HYPRE_FlexGMRESSetTol (HYPRE_Solver solver, HYPRE_Real tol)
(Optional) Set the convergence tolerance 234

7.4.4 int
HYPRE_FlexGMRESSetAbsoluteTol (HYPRE_Solver solver, HYPRE_Real a_tol)
(Optional) Set the absolute convergence tolerance (default is 0). 234

7.4.5 int
HYPRE_FlexGMRESSetMaxIter (HYPRE_Solver solver, int max_iter)
(Optional) Set maximum number of iterations 234

7.4.6 int
HYPRE_FlexGMRESSetKDim (HYPRE_Solver solver, int k_dim)
(Optional) Set the maximum size of the Krylov space 235

7.4.7 int
HYPRE_FlexGMRESSetPrecond (HYPRE_Solver solver, HYPRE_PtrToSolverFcn precond, HYPRE_PtrToSolverFcn precond_setup, HYPRE_Solver precond_solver)
(Optional) Set the preconditioner to use 235

7.4.8 int
HYPRE_FlexGMRESSetLogging (HYPRE_Solver solver, int logging)
(Optional) Set the amount of logging to do 235

7.4.9 int
HYPRE_FlexGMRESSetPrintLevel (HYPRE_Solver solver, int level)
(Optional) Set the amount of printing to do to the screen 235

7.4.10 int
HYPRE_FlexGMRESGetNumIterations (HYPRE_Solver solver, int* num_iterations)
Return the number of iterations taken 236

7.4.11 int

HYPRE_FlexGMRESGetFinalRelativeResidualNorm (HYPRE_Solver solver, HYPRE_Real* norm)
Return the norm of the final relative residual 236

7.4.12 int
HYPRE_FlexGMRESGetResidual (HYPRE_Solver solver, void** residual)
Return the residual 236

7.4.13 int
HYPRE_FlexGMRESGetTol (HYPRE_Solver solver, HYPRE_Real* tol) 236

7.4.14 int
HYPRE_FlexGMRESGetMaxIter (HYPRE_Solver solver, HYPRE_Intm mxx_ieer)
.... 236

7.4.15 int
HYPRE_FlexGMRESGetKDim (HYPRE_Solver solver, int* k_dim) 237

7.4.16 int
HYPRE_FlexGMRESGetPrecond (HYPRE_Solver solver, HYPRE_Solver* precond_data_ptr)
.... 237

7.4.17 int
HYPRE_FlexGMRESGetLogging (HYPRE_Solver solver, int* leeel) 237

7.4.18 int
HYPRE_FlexGMRESGetPrintLevel (HYPRE_Solver solver, int* level) 237

7.4.19 int
HYPRE_FlexGMRESGetConverged (HYPRE_Solver solver, int* converged) 237

7.4.20 int
HYPRE_FlexGMRESSetModifyPC (HYPRE_Solver solver, HYPRE_PtrToModifyPCFcn modify_pc)
(Optional) Set a user-defined function to modify solve-time preconditioner attributes 238

7.4.1

int
HYPRE_FlexGMRESSetup (HYPRE_Solver solver, HYPRE_Matrix A, HYPRE_Vector b, HYPRE_Vector x)

Prepare to solve the system. The coefficient data in `b` and `x` is ignored here, but information about the layout of the data may be used.

7.4.2

int
HYPRE_FlexGMRESSolve (HYPRE_Solver solver, HYPRE_Matrix A, HYPRE_Vector b, HYPRE_Vector x)

Solve the system

7.4.3

int **HYPRE_FlexGMRESSetTol** (HYPRE_Solver solver, HYPRE_Real tol)

(Optional) Set the convergence tolerance

7.4.4

int
HYPRE_FlexGMRESSetAbsoluteTol (HYPRE_Solver solver, HYPRE_Real a_tol)

(Optional) Set the absolute convergence tolerance (default is 0). If one desires the convergence test to check the absolute convergence tolerance *only*, then set the relative convergence tolerance to 0.0. (The convergence test is $\|r\| \leq$ max(relative_tolerance$*\|b\|$, absolute_tolerance).)

7.4.5

int **HYPRE_FlexGMRESSetMaxIter** (HYPRE_Solver solver, int max_iter)

(Optional) Set maximum number of iterations

7.4.6

int **HYPRE_FlexGMRESSetKDim** (HYPRE_Solver solver, int k_dim)

(Optional) Set the maximum size of the Krylov space

7.4.7

int
HYPRE_FlexGMRESSetPrecond (HYPRE_Solver solver,
HYPRE_PtrToSolverFcn precond, HYPRE_PtrToSolverFcn precond_setup,
HYPRE_Solver precond_solver)

(Optional) Set the preconditioner to use

7.4.8

int **HYPRE_FlexGMRESSetLogging** (HYPRE_Solver solver, int logging)

(Optional) Set the amount of logging to do

7.4.9

int **HYPRE_FlexGMRESSetPrintLevel** (HYPRE_Solver solver, int level)

(Optional) Set the amount of printing to do to the screen

7.4.10

int
HYPRE_FlexGMRESGetNumIterations (HYPRE_Solver solver, int* num_iterations)

Return the number of iterations taken

7.4.11

int
HYPRE_FlexGMRESGetFinalRelativeResidualNorm (HYPRE_Solver solver, HYPRE_Real* norm)

Return the norm of the final relative residual

7.4.12

int **HYPRE_FlexGMRESGetResidual** (HYPRE_Solver solver, void** residual)

Return the residual

7.4.13

int **HYPRE_FlexGMRESGetTol** (HYPRE_Solver solver, HYPRE_Real* tol)

7.4.14

int
HYPRE_FlexGMRESGetMaxIter (HYPRE_Solver solver, HYPRE_Intm mxx_ieer)

7.4.15

int **HYPRE_FlexGMRESGetKDim** (HYPRE_Solver solver, int* k_dim)

7.4.16

int
HYPRE_FlexGMRESGetPrecond (HYPRE_Solver solver, HYPRE_Solver* precond_data_ptr)

7.4.17

int **HYPRE_FlexGMRESGetLogging** (HYPRE_Solver solver, int* leeel)

7.4.18

int **HYPRE_FlexGMRESGetPrintLevel** (HYPRE_Solver solver, int* level)

7.4.19

int
HYPRE_FlexGMRESGetConverged (HYPRE_Solver solver, int* converged)

7.4.20

int
HYPRE_FlexGMRESSetModifyPC (HYPRE_Solver solver, HYPRE_PtrToModifyPCFcn modify_pc)

(Optional) Set a user-defined function to modify solve-time preconditioner attributes

7.5

LGMRES Solver

Names

7.5.1 int
HYPRE_LGMRESSetup (HYPRE_Solver solver, HYPRE_Matrix A, HYPRE_Vector b, HYPRE_Vector x)
Prepare to solve the system. 240

7.5.2 int
HYPRE_LGMRESSolve (HYPRE_Solver solver, HYPRE_Matrix A, HYPRE_Vector b, HYPRE_Vector x)
Solve the system. 240

7.5.3 int
HYPRE_LGMRESSetTol (HYPRE_Solver solver, HYPRE_Real tol)
(Optional) Set the convergence tolerance 240

7.5.4 int
HYPRE_LGMRESSetAbsoluteTol (HYPRE_Solver solver, HYPRE_Real a_tol)
(Optional) Set the absolute convergence tolerance (default is 0). 240

7.5.5 int
HYPRE_LGMRESSetMaxIter (HYPRE_Solver solver, int max_iter)
(Optional) Set maximum number of iterations 241

7.5.6 int
HYPRE_LGMRESSetKDim (HYPRE_Solver solver, int k_dim)
(Optional) Set the maximum size of the approximation space (includes the augmentation vectors) 241

7.5.7 int
HYPRE_LGMRESSetAugDim (HYPRE_Solver solver, int aug_dim)
(Optional) Set the number of augmentation vectors (default: 2) 241

7.5.8 int

HYPRE_LGMRESSetPrecond (HYPRE_Solver solver, HYPRE_PtrToSolverFcn precond, HYPRE_PtrToSolverFcn precond_setup, HYPRE_Solver precond_solver)
(Optional) Set the preconditioner to use 241

7.5.9 int
HYPRE_LGMRESSetLogging (HYPRE_Solver solver, int logging)
(Optional) Set the amount of logging to do 242

7.5.10 int
HYPRE_LGMRESSetPrintLevel (HYPRE_Solver solver, int level)
(Optional) Set the amount of printing to do to the screen 242

7.5.11 int
HYPRE_LGMRESGetNumIterations (HYPRE_Solver solver, int* num_iterations)
Return the number of iterations taken 242

7.5.12 int
HYPRE_LGMRESGetFinalRelativeResidualNorm (HYPRE_Solver solver, HYPRE_Real* norm)
Return the norm of the final relative residual 242

7.5.13 int
HYPRE_LGMRESGetResidual (HYPRE_Solver solver, void** residual)
Return the residual 243

7.5.14 int
HYPRE_LGMRESGetTol (HYPRE_Solver solver, HYPRE_Real* tol) 243

7.5.15 int
HYPRE_LGMRESGetMaxIter (HYPRE_Solver solver, int* max_iter) 243

7.5.16 int
HYPRE_LGMRESGetKDim (HYPRE_Solver solver, int* k_dim) 243

7.5.17 int
HYPRE_LGMRESGetAugDim (HYPRE_Solver solver, int* k_dim).......... 243

7.5.18 int
HYPRE_LGMRESGetPrecond (HYPRE_Solver solver, HYPRE_Solver* precond_data_ptr)
.... 244

7.5.19 int
HYPRE_LGMRESGetLogging (HYPRE_Solver solver, int* level)............ 244

7.5.20 int
HYPRE_LGMRESGetPrintLevel (HYPRE_Solver solver, int* level)......... 244

7.5.21 int
HYPRE_LGMRESGetConverged (HYPRE_Solver solver, int* converged) ... 244

7.5.1

int
HYPRE_LGMRESSetup (HYPRE_Solver solver, HYPRE_Matrix A,
HYPRE_Vector b, HYPRE_Vector x)

Prepare to solve the system. The coefficient data in `b` and `x` is ignored here, but information about the layout of the data may be used.

7.5.2

int
HYPRE_LGMRESSolve (HYPRE_Solver solver, HYPRE_Matrix A,
HYPRE_Vector b, HYPRE_Vector x)

Solve the system. Details on LGMRES may be found in A. H. Baker, E.R. Jessup, and T.A. Manteuffel, "A technique for accelerating the convergence of restarted GMRES." SIAM Journal on Matrix Analysis and Applications, 26 (2005), pp. 962-984. LGMRES(m,k) in the paper corresponds to LGMRES(Kdim+AugDim, AugDim).

7.5.3

int **HYPRE_LGMRESSetTol** (HYPRE_Solver solver, HYPRE_Real tol)

(Optional) Set the convergence tolerance

7.5.4

int
HYPRE_LGMRESSetAbsoluteTol (HYPRE_Solver solver, HYPRE_Real a_tol)

(Optional) Set the absolute convergence tolerance (default is 0). If one desires the convergence test to check the absolute convergence tolerance *only*, then set the relative convergence tolerance to 0.0. (The convergence test is $\|r\| \leq$ max(relative_tolerance$*\|b\|$, absolute_tolerance).)

7.5.5

int **HYPRE_LGMRESSetMaxIter** (HYPRE_Solver solver, int max_iter)

(Optional) Set maximum number of iterations

7.5.6

int **HYPRE_LGMRESSetKDim** (HYPRE_Solver solver, int k_dim)

(Optional) Set the maximum size of the approximation space (includes the augmentation vectors)

7.5.7

int **HYPRE_LGMRESSetAugDim** (HYPRE_Solver solver, int aug_dim)

(Optional) Set the number of augmentation vectors (default: 2)

7.5.8

int
HYPRE_LGMRESSetPrecond (HYPRE_Solver solver, HYPRE_PtrToSolverFcn precond, HYPRE_PtrToSolverFcn precond_setup, HYPRE_Solver precond_solver)

(Optional) Set the preconditioner to use

7.5.9

int **HYPRE_LGMRESSetLogging** (HYPRE_Solver solver, int logging)

(Optional) Set the amount of logging to do

7.5.10

int **HYPRE_LGMRESSetPrintLevel** (HYPRE_Solver solver, int level)

(Optional) Set the amount of printing to do to the screen

7.5.11

int
HYPRE_LGMRESGetNumIterations (HYPRE_Solver solver, int*
num_iterations)

Return the number of iterations taken

7.5.12

int
HYPRE_LGMRESGetFinalRelativeResidualNorm (HYPRE_Solver solver,
HYPRE_Real* norm)

Return the norm of the final relative residual

7.5.13

int **HYPRE_LGMRESGetResidual** (HYPRE_Solver solver, void** residual)

Return the residual

7.5.14

int **HYPRE_LGMRESGetTol** (HYPRE_Solver solver, HYPRE_Real* tol)

7.5.15

int **HYPRE_LGMRESGetMaxIter** (HYPRE_Solver solver, int* max_iter)

7.5.16

int **HYPRE_LGMRESGetKDim** (HYPRE_Solver solver, int* k_dim)

7.5.17

int **HYPRE_LGMRESGetAugDim** (HYPRE_Solver solver, int* k_dim)

7.5.18

int
HYPRE_LGMRESGetPrecond (HYPRE_Solver solver, HYPRE_Solver* precond_data_ptr)

7.5.19

int **HYPRE_LGMRESGetLogging** (HYPRE_Solver solver, int* level)

7.5.20

int **HYPRE_LGMRESGetPrintLevel** (HYPRE_Solver solver, int* level)

7.5.21

int **HYPRE_LGMRESGetConverged** (HYPRE_Solver solver, int* converged)

7.6 BiCGSTAB Solver

Names

7.6.1 int
HYPRE_BiCGSTABSetup (HYPRE_Solver solver, HYPRE_Matrix A, HYPRE_Vector b, HYPRE_Vector x)
Prepare to solve the system. 246

7.6.2 int

HYPRE_BiCGSTABSolve (HYPRE_Solver solver, HYPRE_Matrix A, HYPRE_Vector b, HYPRE_Vector x)
Solve the system 246

7.6.3 int
HYPRE_BiCGSTABSetTol (HYPRE_Solver solver, HYPRE_Real tol)
(Optional) Set the convergence tolerance 246

7.6.4 int
HYPRE_BiCGSTABSetAbsoluteTol (HYPRE_Solver solver, HYPRE_Real a_tol)
(Optional) Set the absolute convergence tolerance (default is 0). 246

7.6.5 int
HYPRE_BiCGSTABSetMaxIter (HYPRE_Solver solver, int max_iter)
(Optional) Set maximum number of iterations 247

7.6.6 int
HYPRE_BiCGSTABSetPrecond (HYPRE_Solver solver, HYPRE_PtrToSolverFcn precond, HYPRE_PtrToSolverFcn precond_setup, HYPRE_Solver precond_solver)
(Optional) Set the preconditioner to use 247

7.6.7 int
HYPRE_BiCGSTABSetLogging (HYPRE_Solver solver, int logging)
(Optional) Set the amount of logging to do 247

7.6.8 int
HYPRE_BiCGSTABSetPrintLevel (HYPRE_Solver solver, int level)
(Optional) Set the amount of printing to do to the screen 247

7.6.9 int
HYPRE_BiCGSTABGetNumIterations (HYPRE_Solver solver, int* num_iterations)
Return the number of iterations taken 248

7.6.10 int
HYPRE_BiCGSTABGetFinalRelativeResidualNorm (HYPRE_Solver solver, HYPRE_Real* norm)
Return the norm of the final relative residual 248

7.6.11 int
HYPRE_BiCGSTABGetResidual (HYPRE_Solver solver, void** residual)
Return the residual 248

7.6.12 int
HYPRE_BiCGSTABGetPrecond (HYPRE_Solver solver, HYPRE_Solver* precond_data_ptr)
.... 248

7.6.1

int
HYPRE_BiCGSTABSetup (HYPRE_Solver solver, HYPRE_Matrix A, HYPRE_Vector b, HYPRE_Vector x)

Prepare to solve the system. The coefficient data in `b` and `x` is ignored here, but information about the layout of the data may be used.

7.6.2

int
HYPRE_BiCGSTABSolve (HYPRE_Solver solver, HYPRE_Matrix A, HYPRE_Vector b, HYPRE_Vector x)

Solve the system

7.6.3

int **HYPRE_BiCGSTABSetTol** (HYPRE_Solver solver, HYPRE_Real tol)

(Optional) Set the convergence tolerance

7.6.4

int
HYPRE_BiCGSTABSetAbsoluteTol (HYPRE_Solver solver, HYPRE_Real a_tol)

(Optional) Set the absolute convergence tolerance (default is 0). If one desires the convergence test to check the absolute convergence tolerance *only*, then set the relative convergence tolerance to 0.0. (The convergence test is $\|r\| \leq$ max(relative_tolerance $*\|b\|$, absolute_tolerance).)

7.6.5

int **HYPRE_BiCGSTABSetMaxIter** (HYPRE_Solver solver, int max_iter)

(Optional) Set maximum number of iterations

7.6.6

int
HYPRE_BiCGSTABSetPrecond (HYPRE_Solver solver,
HYPRE_PtrToSolverFcn precond, HYPRE_PtrToSolverFcn precond_setup,
HYPRE_Solver precond_solver)

(Optional) Set the preconditioner to use

7.6.7

int **HYPRE_BiCGSTABSetLogging** (HYPRE_Solver solver, int logging)

(Optional) Set the amount of logging to do

7.6.8

int **HYPRE_BiCGSTABSetPrintLevel** (HYPRE_Solver solver, int level)

(Optional) Set the amount of printing to do to the screen

7.6.9

int
HYPRE_BiCGSTABGetNumIterations (HYPRE_Solver solver, int* num_iterations)

Return the number of iterations taken

7.6.10

int
HYPRE_BiCGSTABGetFinalRelativeResidualNorm (HYPRE_Solver solver, HYPRE_Real* norm)

Return the norm of the final relative residual

7.6.11

int **HYPRE_BiCGSTABGetResidual** (HYPRE_Solver solver, void** residual)

Return the residual

7.6.12

int
HYPRE_BiCGSTABGetPrecond (HYPRE_Solver solver, HYPRE_Solver* precond_data_ptr)

7.7

CGNR Solver

Names

7.7.1 int
HYPRE_CGNRSetup (HYPRE_Solver solver, HYPRE_Matrix A, HYPRE_Vector b, HYPRE_Vector x)
Prepare to solve the system. 250

7.7.2 int
HYPRE_CGNRSolve (HYPRE_Solver solver, HYPRE_Matrix A, HYPRE_Vector b, HYPRE_Vector x)
Solve the system 250

7.7.3 int
HYPRE_CGNRSetTol (HYPRE_Solver solver, HYPRE_Real tol)
(Optional) Set the convergence tolerance 250

7.7.4 int
HYPRE_CGNRSetMaxIter (HYPRE_Solver solver, int max_iter)
(Optional) Set maximum number of iterations 250

7.7.5 int
HYPRE_CGNRSetPrecond (HYPRE_Solver solver, HYPRE_PtrToSolverFcn precond, HYPRE_PtrToSolverFcn precondT, HYPRE_PtrToSolverFcn precond_setup, HYPRE_Solver precond_solver)
(Optional) Set the preconditioner to use. 251

7.7.6 int
HYPRE_CGNRSetLogging (HYPRE_Solver solver, int logging)
(Optional) Set the amount of logging to do 251

7.7.7 int
HYPRE_CGNRGetNumIterations (HYPRE_Solver solver, int* num_iterations)
Return the number of iterations taken 251

7.7.8 int
HYPRE_CGNRGetFinalRelativeResidualNorm (HYPRE_Solver solver, HYPRE_Real* norm)
Return the norm of the final relative residual 251

7.7.9 int
HYPRE_CGNRGetPrecond (HYPRE_Solver solver, HYPRE_Solver* precond_data_ptr)
.... 252

7.7.1

int
HYPRE_CGNRSetup (HYPRE_Solver solver, HYPRE_Matrix A, HYPRE_Vector b, HYPRE_Vector x)

Prepare to solve the system. The coefficient data in `b` and `x` is ignored here, but information about the layout of the data may be used.

7.7.2

int
HYPRE_CGNRSolve (HYPRE_Solver solver, HYPRE_Matrix A, HYPRE_Vector b, HYPRE_Vector x)

Solve the system

7.7.3

int **HYPRE_CGNRSetTol** (HYPRE_Solver solver, HYPRE_Real tol)

(Optional) Set the convergence tolerance

7.7.4

int **HYPRE_CGNRSetMaxIter** (HYPRE_Solver solver, int max_iter)

(Optional) Set maximum number of iterations

7.7.5

int
HYPRE_CGNRSetPrecond (HYPRE_Solver solver, HYPRE_PtrToSolverFcn precond, HYPRE_PtrToSolverFcn precondT, HYPRE_PtrToSolverFcn precond_setup, HYPRE_Solver precond_solver)

(Optional) Set the preconditioner to use. Note that the only preconditioner available in hypre for use with CGNR is currently BoomerAMG. It requires to use Jacobi as a smoother without CF smoothing, i.e. relax_type needs to be set to 0 or 7 and relax_order needs to be set to 0 by the user, since these are not default values. It can be used with a relaxation weight for Jacobi, which can significantly improve convergence.

7.7.6

int **HYPRE_CGNRSetLogging** (HYPRE_Solver solver, int logging)

(Optional) Set the amount of logging to do

7.7.7

int
HYPRE_CGNRGetNumIterations (HYPRE_Solver solver, int* num_iterations)

Return the number of iterations taken

7.7.8

int
HYPRE_CGNRGetFinalRelativeResidualNorm (HYPRE_Solver solver, HYPRE_Real* norm)

Return the norm of the final relative residual

7.7.9

int
HYPRE_CGNRGetPrecond (HYPRE_Solver solver, HYPRE_Solver* precond_data_ptr)

8

Eigensolvers

Names

8.1 **EigenSolvers** 253

8.2 **LOBPCG Eigensolver** 254

These eigensolvers support many of the matrix/vector storage schemes in hypre. They should be used in conjunction with the storage-specific interfaces.

8.1

EigenSolvers

Names

8.1.1 typedef struct hypre_Solver_struct ***HYPRE_Solver**
The solver object 253

8.1.2 typedef struct hypre_Matrix_struct ***HYPRE_Matrix**
The matrix object 254

8.1.3 typedef struct hypre_Vector_struct ***HYPRE_Vector**
The vector object 254

8.1.1

typedef struct hypre_Solver_struct ***HYPRE_Solver**

The solver object

8.1.2

typedef struct hypre_Matrix_struct ***HYPRE_Matrix**

The matrix object

8.1.3

typedef struct hypre_Vector_struct ***HYPRE_Vector**

The vector object

8.2

LOBPCG Eigensolver

Names

8.2.1 int
HYPRE_LOBPCGCreate (mv_InterfaceInterpreter* interpreter, HYPRE_MatvecFunctions* mvfunctions, HYPRE_Solver* solver)
LOBPCG constructor 255

8.2.2 int
HYPRE_LOBPCGDestroy (HYPRE_Solver solver)
LOBPCG destructor 255

8.2.3 int
HYPRE_LOBPCGSetPrecond (HYPRE_Solver solver, HYPRE_PtrToSolverFcn precond, HYPRE_PtrToSolverFcn precond_setup, HYPRE_Solver precond_solver)
(Optional) Set the preconditioner to use. 256

8.2.4 int
HYPRE_LOBPCGGetPrecond (HYPRE_Solver solver, HYPRE_Solver* precond_data_ptr)
.... 256

8.2.5 int
HYPRE_LOBPCGSetup (HYPRE_Solver solver, HYPRE_Matrix A, HYPRE_Vector b, HYPRE_Vector x)
Set up `A` *and the preconditioner (if there is one)* 256

8.2.6 int

HYPRE_LOBPCGSetupB (HYPRE_Solver solver, HYPRE_Matrix B, HYPRE_Vector x)
(Optional) Set up `B`. 256

8.2.7 int
HYPRE_LOBPCGSetupT (HYPRE_Solver solver, HYPRE_Matrix T, HYPRE_Vector x)
(Optional) Set the preconditioning to be applied to $Tx = b$, not $Ax = b$. 257

8.2.8 int
HYPRE_LOBPCGSolve (HYPRE_Solver solver, mv_MultiVectorPtr y, mv_MultiVectorPtr x, HYPRE_Real* lambda)
Solve A x = lambda B x, y'x = 0 257

8.2.9 int
HYPRE_LOBPCGSetTol (HYPRE_Solver solver, HYPRE_Real tol)
(Optional) Set the absolute convergence tolerance 257

8.2.10 int
HYPRE_LOBPCGSetMaxIter (HYPRE_Solver solver, int max_iter)
(Optional) Set maximum number of iterations 257

8.2.11 int
HYPRE_LOBPCGSetPrecondUsageMode (HYPRE_Solver solver, int mode)
Define which initial guess for inner PCG iterations to use: `mode` *= 0: use zero initial guess, otherwise use RHS* 258

8.2.12 int
HYPRE_LOBPCGSetPrintLevel (HYPRE_Solver solver, int level)
(Optional) Set the amount of printing to do to the screen 258

8.2.1

int
HYPRE_LOBPCGCreate (mv_InterfaceInterpreter* interpreter, HYPRE_MatvecFunctions* mvfunctions, HYPRE_Solver* solver)

LOBPCG constructor

8.2.2

int **HYPRE_LOBPCGDestroy** (HYPRE_Solver solver)

LOBPCG destructor

8.2.3

int
HYPRE_LOBPCGSetPrecond (HYPRE_Solver solver, HYPRE_PtrToSolverFcn precond, HYPRE_PtrToSolverFcn precond_setup, HYPRE_Solver precond_solver)

(Optional) Set the preconditioner to use. If not called, preconditioning is not used.

8.2.4

int
HYPRE_LOBPCGGetPrecond (HYPRE_Solver solver, HYPRE_Solver* precond_data_ptr)

8.2.5

int
HYPRE_LOBPCGSetup (HYPRE_Solver solver, HYPRE_Matrix A, HYPRE_Vector b, HYPRE_Vector x)

Set up `A` and the preconditioner (if there is one)

8.2.6

int
HYPRE_LOBPCGSetupB (HYPRE_Solver solver, HYPRE_Matrix B, HYPRE_Vector x)

(Optional) Set up `B`. If not called, B = I.

8.2.7

int
HYPRE_LOBPCGSetupT (HYPRE_Solver solver, HYPRE_Matrix T, HYPRE_Vector x)

(Optional) Set the preconditioning to be applied to Tx = b, not Ax = b

8.2.8

int
HYPRE_LOBPCGSolve (HYPRE_Solver solver, mv_MultiVectorPtr y, mv_MultiVectorPtr x, HYPRE_Real* lambda)

Solve A x = lambda B x, y'x = 0

8.2.9

int **HYPRE_LOBPCGSetTol** (HYPRE_Solver solver, HYPRE_Real tol)

(Optional) Set the absolute convergence tolerance

8.2.10

int **HYPRE_LOBPCGSetMaxIter** (HYPRE_Solver solver, int max_iter)

(Optional) Set maximum number of iterations

8.2.11

int
HYPRE_LOBPCGSetPrecondUsageMode (HYPRE_Solver solver, int mode)

Define which initial guess for inner PCG iterations to use: `mode` = 0: use zero initial guess, otherwise use RHS

8.2.12

int **HYPRE_LOBPCGSetPrintLevel** (HYPRE_Solver solver, int level)

(Optional) Set the amount of printing to do to the screen

9 Finite Element Interface

Names

9.1 **FEI Functions** 259

9.2 **FEI Solver Parameters** 269

9.1 FEI Functions

Names

9.1.1 **LLNL_FEI_Impl** (MPI_Comm comm)
Finite element interface constructor: this function creates an instantiation of the HYPRE fei class. 261

9.1.2 **~LLNL_FEI_Impl** ()
Finite element interface destructor: this function destroys the object as well as its internal memory allocations. 261

9.1.3 int
parameters (int numParams, char** paramStrings)
The parameter function is the single most important function to pass solver information (which solver, which preconditioner, tolerance, other solver parameters) to HYPRE. 262

9.1.4 int
initFields (int numFields, int* fieldSizes, int* fieldIDs)
Each node or element variable has one or more fields. 262

9.1.5 int
initElemBlock (int elemBlockID, int numElements, int numNodesPerElement, int* numFieldsPerNode, int** nodalFieldIDs, int numElemDOFFieldsPerElement, int* elemDOFFieldIDs, int interleaveStrategy)
The whole finite element mesh can be broken down into a number of element blocks. 262

9.1.6 int
initElem (int elemBlockID, int elemID, int* elemConn)
This function initializes element connectivity (that is, the node identifiers associated with the current element) given an element block identifier and the element identifier with the element block. 263

9.1.7 int

initSharedNodes (int nShared, int* sharedIDs, int* sharedLengs, int** sharedProcs)
This function initializes the nodes that are shared between the current processor and its neighbors. 263

9.1.8 int
initCRMult (int CRListLen, int* CRNodeList, int* CRFieldList, int* CRID)
This function initializes the Lagrange multiplier constraints 263

9.1.9 int
initComplete ()
This function signals to the FEI that the initialization step has been completed. 264

9.1.10 int
resetSystem (double s)
This function resets the global matrix to be of the same sparsity pattern as before but with every entry set to s. 264

9.1.11 int
resetMatrix (double s)
This function resets the global matrix to be of the same sparsity pattern as before but with every entry set to s. 264

9.1.12 int
resetRHSVector (double s)
This function resets the right hand side vector to s. 265

9.1.13 int
resetInitialGuess (double s)
This function resets the solution vector to s. 265

9.1.14 int
loadNodeBCs (int nNodes, int* nodeIDs, int fieldID, double** alpha, double** beta, double** gamma)
This function loads the nodal boundary conditions. 265

9.1.15 int
sumInElem (int elemBlockID, int elemID, int* elemConn, double** elemStiff, double* elemLoad, int elemFormat)
This function adds the element contribution to the global stiffness matrix and also the element load to the right hand side vector 266

9.1.16 int
sumInElemMatrix (int elemBlock, int elemID, int* elemConn, double** elemStiffness, int elemFormat)
This function differs from the sumInElem function in that the right hand load vector is not passed. 266

9.1.17 int
sumInElemRHS (int elemBlock, int elemID, int* elemConn, double* elemLoad)
This function adds the element load to the right hand side vector 266

9.1.18 int
loadComplete ()
This function signals to the FEI that the loading phase has been completed. 267

9.1.19 int

getNumBlockActNodes (int elemBlockID, int* nNodes)
This function returns the number of nodes given the element block. 267

9.1.20 int
getNumBlockActEqns (int elemBlockID, int* nEqns)
This function returns the number of unknowns given the element block. .. 267

9.1.21 int
getBlockNodeIDList (int elemBlockID, int numNodes, int* nodeIDList)
This function returns the node identifiers given the element block. 268

9.1.22 int
getBlockNodeSolution (int elemBlockID, int numNodes, int* nodeIDList, int* solnOffsets, double* solnValues)
This function returns the nodal solutions given the element block number. 268

9.1.23 int
loadCRMult (int CRID, int CRListLen, int* CRNodeList, int* CRFieldList, double* CRWeightList, double CRValue)
This function loads the Lagrange multiplier constraints 268

9.1.1

LLNL_FEI_Impl (MPI_Comm comm)

Finite element interface constructor: this function creates an instantiation of the HYPRE fei class.

Parameters: `comm` - an MPI communicator

9.1.2

~LLNL_FEI_Impl ()

Finite element interface destructor: this function destroys the object as well as its internal memory allocations.

Parameters: - no parameter needed

9.1.3

int **parameters** (int numParams, char** paramStrings)

The parameter function is the single most important function to pass solver information (which solver, which preconditioner, tolerance, other solver parameters) to HYPRE.

Parameters:

`numParams`	- number of command strings
`paramStrings`	- the command strings

9.1.4

int **initFields** (int numFields, int* fieldSizes, int* fieldIDs)

Each node or element variable has one or more fields. The field information can be set up using this function.

Parameters:

`numFields`	- total number of fields for all variable types
`fieldSizes`	- degree of freedom for each field type
`fieldIDs`	- a list of field identifiers

9.1.5

int
initElemBlock (int elemBlockID, int numElements, int numNodesPerElement, int* numFieldsPerNode, int** nodalFieldIDs, int numElemDOFFieldsPerElement, int* elemDOFFieldIDs, int interleaveStrategy)

The whole finite element mesh can be broken down into a number of element blocks. The attributes for each element block are: an identifier, number of elements, number of nodes per elements, the number of fields in each element node, etc.

Parameters:

`elemblockID`	- element block identifier
`numElements`	- number of element in this block
`numNodesPerElement`	- number of nodes per element in this block
`numFieldsPerNode`	- number of fields for each node
`nodalFieldIDs`	- field identifiers for the nodal unknowns
`numElemDOFFieldsPerElement`	- number of fields for the element
`elemDOFFieldIDs`	- field identifier for the element unknowns
`interleaveStratety`	- indicates how unknowns are ordered

9.1.6

int **initElem** (int elemBlockID, int elemID, int* elemConn)

This function initializes element connectivity (that is, the node identifiers associated with the current element) given an element block identifier and the element identifier with the element block.

Parameters:

`elemblockID`	- element block identifier
`elemID`	- element identifier
`elemConn`	- a list of node identifiers for this element

9.1.7

int
initSharedNodes (int nShared, int* sharedIDs, int* sharedLengs, int** sharedProcs)

This function initializes the nodes that are shared between the current processor and its neighbors. The FEI will decide a unique processor each shared node will be assigned to.

Parameters:

`nShared`	- number of shared nodes
`sharedIDs`	- shared node identifiers
`sharedLengs`	- the number of processors each node shares with
`sharedProcs`	- the processor identifiers each node shares with

9.1.8

int **initCRMult** (int CRListLen, int* CRNodeList, int* CRFieldList, int* CRID)

This function initializes the Lagrange multiplier constraints

Parameters:

`CRListLen`	- the number of constraints
`CRNodeList`	- node identifiers where constraints are applied
`CRFieldList`	- field identifiers within nodes where constraints are applied
`CRID`	- the constraint identifier

9.1.9

int **initComplete** ()

This function signals to the FEI that the initialization step has been completed. The loading step will follow.

Parameters: `-` no parameter needed

9.1.10

int **resetSystem** (double s)

This function resets the global matrix to be of the same sparsity pattern as before but with every entry set to s. The right hand side is set to 0.

Parameters: `s` - the value each matrix entry is set to.

9.1.11

int **resetMatrix** (double s)

This function resets the global matrix to be of the same sparsity pattern as before but with every entry set to s.

Parameters: `s` - the value each matrix entry is set to.

9.1.12

int **resetRHSVector** (double s)

This function resets the right hand side vector to s.

Parameters: `s` - the value each right hand side vector entry is set to.

9.1.13

int **resetInitialGuess** (double s)

This function resets the solution vector to s.

Parameters: `s` - the value each solution vector entry is set to.

9.1.14

int
loadNodeBCs (int nNodes, int* nodeIDs, int fieldID, double** alpha, double** beta, double** gamma)

This function loads the nodal boundary conditions. The boundary conditions

Parameters:

`nNodes`	- number of nodes boundary conditions are imposed
`nodeIDs`	- nodal identifiers
`fieldID`	- field identifier with nodes where BC are imposed
`alpha`	- the multipliers for the field
`beta`	- the multipliers for the normal derivative of the field
`gamma`	- the boundary values on the right hand side of the equations

9.1.15

int
sumInElem (int elemBlockID, int elemID, int* elemConn, double** elemStiff, double* elemLoad, int elemFormat)

This function adds the element contribution to the global stiffness matrix and also the element load to the right hand side vector

Parameters:

`elemBlockID`	- element block identifier
`elemID`	- element identifier
`elemConn`	- a list of node identifiers for this element
`elemStiff`	- element stiffness matrix
`elemLoad`	- right hand side (load) for this element
`elemFormat`	- the format the unknowns are passed in

9.1.16

int
sumInElemMatrix (int elemBlock, int elemID, int* elemConn, double** elemStiffness, int elemFormat)

This function differs from the sumInElem function in that the right hand load vector is not passed.

Parameters:

`elemBlockID`	- element block identifier
`elemID`	- element identifier
`elemConn`	- a list of node identifiers for this element
`elemStiff`	- element stiffness matrix
`elemFormat`	- the format the unknowns are passed in

9.1.17

int
sumInElemRHS (int elemBlock, int elemID, int* elemConn, double* elemLoad)

This function adds the element load to the right hand side vector

Parameters:

`elemBlockID`	- element block identifier
`elemID`	- element identifier
`elemConn`	- a list of node identifiers for this element
`elemLoad`	- right hand side (load) for this element

9.1.18

int **loadComplete** ()

This function signals to the FEI that the loading phase has been completed.

Parameters: `-` no parameter needed

9.1.19

int **getNumBlockActNodes** (int elemBlockID, int* nNodes)

This function returns the number of nodes given the element block.

Parameters:

`elemBlockID`	- element block identifier
`nNodes`	- the number of nodes to be returned

9.1.20

int **getNumBlockActEqns** (int elemBlockID, int* nEqns)

This function returns the number of unknowns given the element block.

Parameters:

`elemBlockID`	- element block identifier
`nEqns`	- the number of unknowns to be returned

9.1.21

int **getBlockNodeIDList** (int elemBlockID, int numNodes, int* nodeIDList)

This function returns the node identifiers given the element block.

Parameters:

`elemBlockID`	- element block identifier
`numNodes`	- the number of nodes
`nodeIDList`	- the node identifiers

9.1.22

int
getBlockNodeSolution (int elemBlockID, int numNodes, int* nodeIDList, int* solnOffsets, double* solnValues)

This function returns the nodal solutions given the element block number.

Parameters:

`elemBlockID`	- element block identifier
`numNodes`	- the number of nodes
`nodeIDList`	- the node identifiers
`solnOffsets`	- the equation number for each nodal solution
`solnValues`	- the nodal solution values

9.1.23

int
loadCRMult (int CRID, int CRListLen, int* CRNodeList, int* CRFieldList, double* CRWeightList, double CRValue)

This function loads the Lagrange multiplier constraints

Parameters:

`CRID`	- the constraint identifier
`CRListLen`	- the number of constraints
`CRNodeList`	- node identifiers where constraints are applied
`CRFieldList`	- field identifiers within nodes where constraints are applied
`CRWeightList`	- a list of weights applied to each specified field
`CRValue`	- the constraint value (right hand side of the constraint)

9.2

FEI Solver Parameters

Names

9.2.1 Preconditioners and **Solvers**
Here the various options for solvers and preconditioners are defined. 269

9.2.2 **BoomerAMG**
Parameter options for the algebraic multigrid preconditioner BoomerAMG. 270

9.2.3 **MLI**
Parameter options for the smoothed aggregation preconditioner MLI. 271

9.2.4 **Various**
Parameter options for ILUT, ParaSails and polynomial preconditioners are defined. 272

9.2.5 Matrix **Reduction**
Parameters which define different reduction modes. 273

9.2.6 Performance Tuning and **Diagnostics**
Parameters control diagnostic information, memory use, etc. 273

9.2.7 **Miscellaneous**
Parameters that are helpful for finite element information. 274

9.2.1

Preconditioners and **Solvers**

Here the various options for solvers and preconditioners are defined.

solver xxx where xxx specifies one of cg, gmres, fgmres, bicgs, bicgstab, tfqmr, symqmr, superlu, or superlux. The default is gmres. The solver type can be followed by override to specify its priority when multiple solvers are declared at random order.

preconditioner xxx where xxx is one of diagonal, pilut, euclid, parasails, boomeramg, poly, or mli. The default is diagonal. Another option for xxx is reuse which allows the preconditioner to be reused (this should only be set after a preconditioner has been set up already). The preconditioner type can be followed by override to specify its priority when multiple preconditioners are declared at random order.

maxIterations xxx where xxx is an integer specifying the maximum number of iterations permitted for the iterative solvers. The default value is 1000.

tolerance xxx where xxx is a floating point number specifying the termination criterion for the iterative solvers. The default value is 1.0E-6.

gmresDim xxx where xxx is an integer specifying the value of m in restarted GMRES(m). The default value is 100.

stopCrit xxx where xxx is one of absolute or relative stopping criterion.

superluOrdering xxx - where xxx specifies one of natural or mmd (minimum degree ordering). This ordering is used to minimize the number of nonzeros generated in the LU decomposition. The default is natural ordering.

superluScale xxx where xxx specifies one of y (perform row and column scalings before decomposition) or n. The default is no scaling.

9.2.2 BoomerAMG

Parameter options for the algebraic multigrid preconditioner BoomerAMG.

amgMaxLevels xxx where xxx is an integer specifying the maximum number of levels to be used for the grid hierarchy.

amgCoarsenType xxx where xxx specifies one of falgout or ruge, or default (CLJP) coarsening for BoomerAMG.

amgMeasureType xxx where xxx specifies one of local or or global. This parameter affects how coarsening is performed in parallel.

amgRelaxType xxx where xxx is one of jacobi (Damped Jacobi), gs-slow (sequential Gauss-Seidel), gs-fast (Gauss-Seidel on interior nodes), or hybrid. The default is hybrid.

amgNumSweeps xxx where xxx is an integer specifying the number of pre- and post-smoothing at each level of BoomerAMG. The default is two pre- and two post-smoothings.

amgRelaxWeight xxx where xxx is a floating point number between 0 and 1 specifying the damping factor for BoomerAMG's damped Jacobi and GS smoothers. The default value is 1.0.

amgRelaxOmega xxx where xxx is a floating point number between 0 and 1 specifying the damping factor for BoomerAMG's hybrid smoother for multiple processors. The default value is 1.0.

amgStrongThreshold xxx where xxx is a floating point number between 0 and 1 specifying the threshold used to determine strong coupling in BoomerAMG's coasening. The default value is 0.25.

amgSystemSize xxx where xxx is the degree of freedom per node.

amgMaxLevels xxx where xxx is an integer specifying the maximum number of iterations to be used during the solve phase.

amgUseGSMG - tells BoomerAMG to use a different coarsening called GSMG.

amgGSMGNumSamples where xxx is the number of samples to generate to determine how to coarsen for GSMG.

9.2.3

MLI

Parameter options for the smoothed aggregation preconditioner MLI.

outputLevel xxx where xxx is the output level for diagnostics.

method xxx where xxx is either AMGSA (default), AMGSAe, to indicate which MLI algorithm is to be used.

numLevels xxx where xxx is the maximum number of levels (default=30) used.

maxIterations xxx where xxx is the maximum number of iterations (default = 1 as preconditioner).

cycleType xxx where xxx is either 'V' or 'W' cycle (default = 'V').

strengthThreshold xxx strength threshold for coarsening (default = 0).

smoother xxx where xxx is either Jacobi, BJacobi, GS, SGS, HSGS (SSOR,default), BSGS, ParaSails, MLS, CGJacobi, CGBJacobi, or Chebyshev.

numSweeps xxx where xxx is the number of smoother sweeps (default = 2).

coarseSolver xxx where xxx is one of those in 'smoother' or SuperLU (default).

minCoarseSize xxx where xxx is the minimum coarse grid size to control the number of levels used (default = 3000).

Pweight xxx where xxx is the relaxation parameter for the prolongation smoother (default 0.0).

nodeDOF xxx where xxx is the degree of freedom for each node (default = 1).

nullSpaceDim xxx where xxx is the dimension of the null space for the coarse grid (default = 1).

useNodalCoord xxx where xxx is either 'on' or 'off' (default) to indicate whether the nodal coordinates are used to generate the initial null space.

saAMGCalibrationSize xxx where xxx is the additional null space vectors to be generated via calibration (default = 0).

numSmoothVecs xxx where xxx is the number of near null space vectors used to create the prolongation operator (default = 0).

smoothVecSteps xxx where xxx is the number of smoothing steps used to generate the smooth vectors (default = 0).

In addition, to use 'AMGSAe', the parameter 'haveSFEI' has to be sent into the FEI using the parameters function (this option is valid only for the Sandia FEI implementation).

9.2.4

Various

Parameter options for ILUT, ParaSails and polynomial preconditioners are defined.

euclidNlevels xxx where xxx is an non-negative integer specifying the desired sparsity of the incomplete factors. The default value is 0.

euclidThreshold xxx where xxx is a floating point number specifying the threshold used to sparsify the incomplete factors. The default value is 0.0.

parasailsThreshold xxx where xxx is a floating point number between 0 and 1 specifying the threshold used to prune small entries in setting up the sparse approximate inverse. The default value is 0.0.

parasailsNlevels xxx where xxx is an integer larger than 0 specifying the desired sparsity of the approximate inverse. The default value is 1.

parasailsFilter xxx where xxx is a floating point number between 0 and 1 specifying the threshold used to prune small entries in A. The default value is 0.0.

parasailsLoadbal xxx where xxx is a floating point number between 0 and 1 specifying how load balancing has to be done (Edmond, explain please). The default value is 0.0.

parasailsSymmetric sets Parasails to take A as symmetric.

parasailsUnSymmetric sets Parasails to take A as nonsymmetric (default).

parasailsReuse sets Parasails to reuse the sparsity pattern of A.

polyorder xxx where xxx is the order of the least-squares polynomial preconditioner.

9.2.5

Matrix **Reduction**

Parameters which define different reduction modes.

schurReduction turns on the Schur reduction mode.

slideReduction turns on the slide reduction mode.

slideReduction2 turns on the slide reduction mode version 2 (see section 2).

slideReduction3 turns on the slide reduction mode version 3 (see section 2).

9.2.6

Performance Tuning and **Diagnostics**

Parameters control diagnostic information, memory use, etc.

outputLevel xxx where xxx is an integer specifying the output level. An output level of 1 prints only the solver information such as number of iterations and timings. An output level of 2 prints debug information such as the functions visited and preconditioner information. An output level of 3 or higher prints more debug information such as the matrix and right hand side loaded via the LinearSystemCore functions to the standard output.

setDebug xxx where xxx is one of slideReduction1, slideReduction2, slideReduction3 (level 1,2,3 diagnostics in the slide surface reduction code), printMat (print the original matrix into a file), printReducedMat (print the reduced matrix into a file), printSol (print the solution into a file), ddilut (output diagnostic information for DDIlut preconditioner setup), and amgDebug (output diagnostic information for AMG).

optimizeMemory cleans up the matrix sparsity pattern after the matrix has been loaded. (It has been kept to allow matrix reuse.)

imposeNoBC turns off the boundary condition to allow diagnosing the matrix (for example, checking the null space.)

9.2.7

Miscellaneous

Parameters that are helpful for finite element information.

AConjugateProjection xxx where xxx specifies the number of previous solution vectors to keep for the A-conjugate projection. The default is 0 (the projection is off).

minResProjection xxx where xxx specifies the number of previous solution vectors to keep for projection. The default is 0 (the projection is off).

haveFEData indicates that additional finite element information are available to assist in building more efficient solvers.

haveSFEI indicates that the simplified finite element information are available to assist in building more efficient solvers.