# Contents

## 1 Struct System Interface — A structured-grid conceptual interface

1.1 Struct Grids ........................................... 5
1.2 Struct Stencils .................................... 8
1.3 Struct Matrices ................................... 9
1.4 Struct Vectors .................................. 15

## 2 SStruct System Interface — A semi-structured-grid conceptual interface

2.1 SStruct Grids ....................................... 21
2.2 SStruct Stencils .................................. 27
2.3 SStruct Graphs .................................... 28
2.4 SStruct Matrices .................................. 31
2.5 SStruct Vectors .................................. 37

## 3 IJ System Interface — A linear-algebraic conceptual interface

3.1 IJ Matrices ........................................ 46
3.2 IJ Vectors ....................................... 62

## 4 Struct Solvers — Linear solvers for structured grids

4.1 Struct Solvers ..................................... 58
4.2 Struct Jacobi Solver .............................. 59
4.3 Struct PF MG Solver ............................. 62
4.4 Struct SMG Solver ................................. 69
4.5 Struct PCG Solver ................................. 74
4.6 Struct GMRES Solver ............................. 80
4.7 Struct BiCGSTAB Solver ......................... 84
4.8 Struct Hybrid Solver ............................. 88

## 5 SStruct Solvers — Linear solvers for semi-structured grids

5.1 SStruct Solvers ..................................... 96
5.2 SStruct PCG Solver ............................... 97
5.3 SStruct GMRES Solver ........................... 102
5.4 SStruct BiCGSTAB Solver ....................... 107
5.5 SStruct SysPF MG Solver ....................... 112
5.6 SStruct Split Solver ............................. 118
5.7 SStruct FAC Solver ............................... 122
5.8 SStruct Maxwell Solver ......................... 130

## 6 ParCSR Solvers — Linear solvers for sparse matrix systems

6.1 ParCSR Solvers .................................... 138
6.2 ParCSR BoomerAMG Solver and Preconditioner .... 139
6.3 ParCSR ParaSails Preconditioner ................. 161
6.4 ParCSR Euclid Preconditioner ................... 166
6.5 ParCSR Pilut Preconditioner ..................... 169
6.6 ParCSR AMS Solver and Preconditioner ........... 174
6.7 ParCSR Hybrid Solver ............................ 179
6.8 ParCSR PCG Solver ............................... 191
6.9 ParCSR GMRES Solver ........................... 199
6.10 ParCSR BiCGSTAB Solver ....................... 203

## 7 Krylov Solvers — A basic interface for Krylov solvers

7.1 Krylov Solvers ..................................... 209
7.1 Krylov Solvers ................................................................. 209
7.2 PCG Solver ................................................................. 210
7.3 GMRES Solver ............................................................ 216
7.4 BiCGSTAB Solver ......................................................... 221
7.5 CGNR Solver ............................................................... 225

8 Finite Element Interface — A finite element-based conceptual interface .......... 229
8.1 FEI Functions ............................................................. 229
8.2 FEI Solver Parameters .................................................. 239
Copyright (c) 2006 The Regents of the University of California. Produced at the Lawrence Livermore National Laboratory. Written by the HYPRE team. UCRL-CODE-222953. All rights reserved.

This file is part of HYPRE (see http://www.llnl.gov/CASC/hypre/). Please see the COPYRIGHT and LICENSE file for the copyright notice, disclaimer, contact information and the GNU Lesser General Public License.

HYPRE is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License (as published by the Free Software Foundation) version 2.1 dated February 1999.

HYPRE is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the IMPLIED WARRANTY OF MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the terms and conditions of the GNU General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
1 Struct System Interface

Names

1.1 Struct Grids

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.

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

This page has been automatically generated with DOC++

DOC++ is ©1995 by Roland Wunderling
Malte Zöckler
A grid object is constructed out of several “boxes”, defined on a global abstract index space

Create an \( n_{dim} \)-dimensional grid object

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

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.

typedef struct hypre_StructStencil_struct *HYPRE_StructStencil

The stencil object

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

\textbf{int \ HYPRE\_StructStencilDestroy} (HYPRE\_StructStencil stencil)

Destroy a stencil object

1.2.4

\textbf{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 \texttt{HYPRE\_StructStencilSetEntry}.

1.3

\textbf{Struct Matrices}

Names

1.3.1 \textbf{typedef struct \ hypre\_StructMatrix\_struct *HYPRE\_StructMatrix}

\textit{The matrix object} .............................................. \[11\]

1.3.2 \textbf{int \ HYPRE\_StructMatrixCreate} (MPI\_Comm comm, HYPRE\_StructGrid grid,
HYPRE\_StructStencil stencil,
HYPRE\_StructMatrix\* matrix)

\textit{Create a matrix object} ........................................... \[11\]

1.3.3 \textbf{int \ HYPRE\_StructMatrixDestroy} (HYPRE\_StructMatrix matrix)

\textit{Destroy a matrix object} ........................................ \[11\]

1.3.4 \textbf{int \ HYPRE\_StructMatrixInitialize} (HYPRE\_StructMatrix matrix)

\textit{Prepare a matrix object for setting coefficient values} \.......................... \[11\]

1.3.5 \textbf{int \ HYPRE\_StructMatrixSetValues} (HYPRE\_StructMatrix matrix, int* index,
int nentries, int* entries, double* values)

\textit{Set matrix coefficients index by index.} ........................................ \[12\]

1.3.6 \textbf{int}
**1 Struct System Interface**

1.3.7 **int**
**HYPRE_StructMatrixSetConstantValues** (HYPRE_StructMatrix matrix, int nentries, int* entries, double* values)

*Set matrix coefficients which are constant over the grid.*

1.3.8 **int**
**HYPRE_StructMatrixAddToConstantValues** (HYPRE_StructMatrix matrix, int* index, nentries, int* entries, double* values)

*Add to matrix coefficients which are constant over the grid.*

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

*Set matrix coefficients a box at a time.*

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

*Add to matrix coefficients a box at a time.*

1.3.11 **int**
**HYPRE_StructMatrixAssemble** (HYPRE_StructMatrix matrix)

*Finalize the construction of the matrix before using*

1.3.12 **int**
**HYPRE_StructMatrixSetSymmetric** (HYPRE_StructMatrix matrix, int symmetric)

*Define symmetry properties of the matrix.*

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

*Specify which stencil entries are constant over the grid.*

1.3.14 **int**
**HYPRE_StructMatrixSetNumGhost** (HYPRE_StructMatrix matrix, int* num_ghost)

*Set the ghost layer in the matrix*

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

*Print the matrix to file.*

1.3.16 **int**
1 Struct System Interface

**HYPRE_StructMatrixMatvec** ( double alpha, HYPRE_StructMatrix A, HYPRE_StructVector x, double beta, HYPRE_StructVector y )

*Matvec operator.* ........................................

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

```c
int HYPRE_StructMatrixSetValues (HYPRE_StructMatrix matrix, int* index, int nentries, int* entries, double* 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

```c
int HYPRE_StructMatrixAddToValues (HYPRE_StructMatrix matrix, int* index, int nentries, int* entries, double* 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

```c
int HYPRE_StructMatrixSetConstantValues (HYPRE_StructMatrix matrix, int nentries, int* entries, double* values)
```

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

1.3.8

```c
int HYPRE_StructMatrixAddToConstantValues (HYPRE_StructMatrix matrix, int nentries, int* entries, double* values)
```
Add to matrix coefficients which are constant over the grid. The values array is of length nentries.

1.3.9

```c
int HYPRE_StructMatrixSetBoxValues (HYPRE_StructMatrix matrix, int* ilower, int* iupper, int nentries, int* entries, double* values)
```

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

```c
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

```c
int HYPRE_StructMatrixAddToBoxValues (HYPRE_StructMatrix matrix, int* ilower, int* iupper, int nentries, int* entries, double* values)
```

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

1.3.11

```c
int HYPRE_StructMatrixAssemble (HYPRE_StructMatrix matrix)
```

Finalize the construction of the matrix before using
### 1.3.12

```c
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.

### 1.3.13

```c
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

### 1.3.14

```c
int HYPRE_StructMatrixSetNumGhost (HYPRE_StructMatrix matrix, int* num_ghost)
```

Set the ghost layer in the matrix
1.3.15

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

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

1.3.16

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

Matvec operator. This operation is $y = \alpha Ax + \beta y$. 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* .........................................................

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_StructVectorClearGhostValues (HYPRE_StructVector vector)

*Clears the ghostvalues of vector object.* ................................

1.4.6 int
1.4.7 int HYPRE_StructVectorAddToValues (HYPRE_StructVector vector, int* index, double value)

Add to vector coefficients index by index.

1.4.8 int HYPRE_StructVectorSetBoxValues (HYPRE_StructVector vector, int* ilower, int* iupper, double* values)

Set vector coefficients a box at a time.

1.4.9 int HYPRE_StructVectorAddToBoxValues (HYPRE_StructVector vector, int* ilower, int* iupper, double* values)

Add to vector coefficients a box at a time.

1.4.10 int HYPRE_StructVectorAssemble (HYPRE_StructVector vector)

Finalize the construction of the vector before using.

1.4.11 int HYPRE_StructVectorGetValues (HYPRE_StructVector vector, int* index, double* value)

Get vector coefficients index by index.

1.4.12 int HYPRE_StructVectorGetBoxValues (HYPRE_StructVector vector, int* ilower, int* iupper, double* values)

Get vector coefficients a box at a time.

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

Print the vector to file.

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_StructVectorClearGhostValues (HYPRE_StructVector vector)
```

Clears the ghost values of vector object. Beneficial to users that re-assemble a vector object (e.g., in time-stepping).
1.4.6

```
int HYPRE_StructVectorSetValues (HYPRE_StructVector vector, int* index,
    double value)
```

Set vector coefficients index by index.

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

1.4.7

```
int HYPRE_StructVectorAddToValues (HYPRE_StructVector vector, int* index,
    double value)
```

Add to vector coefficients index by index.

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

1.4.8

```
int HYPRE_StructVectorSetBoxValues (HYPRE_StructVector vector, int* ilower,
    int* iupper, double* 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++;
            }
```
1.4.9

```c
int HYPRE_StructVectorAddToBoxValues (HYPRE_StructVector vector, int* ilower, int* iupper, double* values)
```

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

1.4.10

```c
int HYPRE_StructVectorAssemble (HYPRE_StructVector vector)
```

Finalize the construction of the vector before using.

1.4.11

```c
int HYPRE_StructVectorGetValues (HYPRE_StructVector vector, int* index, double* value)
```

Get vector coefficients index by index.

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

1.4.12

```c
int HYPRE_StructVectorGetBoxValues (HYPRE_StructVector vector, int* ilower, int* iupper, double* values)
```
Get vector coefficients a box at a time. The data in `values` is ordered as in `HYPRE_StructVectorSetBoxValues`.

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

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

2

Names

2.1 SStruct Grids

2.2 SStruct Stencils

2.3 SStruct Graphs

2.4 SStruct Matrices

2.5 SStruct Vectors

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

2.1 SStruct Grids

typedef struct hypre_SStructGrid_struct *HYPRE_SStructGrid

A grid object is constructed out of several structured “parts” and an optional unstructured “part”.

typedef enum hypre_SStructVariable_enum HYPRE_SStructVariable

An enumerated type that supports cell centered, node centered, face centered, and edge centered variables.

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

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

int HYPRE_SStructGridDestroy (HYPRE_SStructGrid grid)

Destroy a grid object.

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.

This page has been automatically generated with DOC++

DOC++ is ©1995 by Roland Wunderling
Malte Zöckler


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

       Describe how regions just outside of a part relate to other parts. .......  25

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

       Add an unstructured part to the grid. ..................................  25

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

       Set the periodicity a particular part. ...............................  25

2.1.12 int HYPRE_SStructGridSetNumGhost (HYPRE_SStructGrid grid,
                                      int* num_ghost)

       Setting ghost in the sgrids ...........................................  25

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 enum hypre_SStructVariable_enum 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 \texttt{ndim}-dimensional grid object with \texttt{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.

2.1.8

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

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

The indexes ilower and iupper map directly to the indexes nbor_ilower and nbor_iupper. Although, it is required that indexes increase from ilower to iupper, indexes may increase and/or decrease 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.

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.

2.1.9

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.

### 2.1.10

```c
int HYPRE_SStructGridAssemble (HYPRE_SStructGrid grid)
```

Finalize the construction of the grid before using.

### 2.1.11

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

Set the periodicity 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.

### 2.1.12

```c
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 ............................................... 27

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 .................................................. 27

2.2.3 int HYPRE_SStructStencilDestroy (HYPRE_SStructStencil stencil)

Destroy a stencil object ........................................ 28

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

Set a stencil entry .............................................. 28

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

2.3.1
typedef struct hypre_SStructGraph_struct *HYPRE_SStructGraph

The graph object is used to describe the nonzero structure of a matrix .... 29

2.3.2

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

Create a graph object ................................................. 29

2.3.3

int HYPRE_SStructGraphDestroy (HYPRE_SStructGraph graph)

Destroy a graph object .................................................. 29

2.3.4

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 ............ 30

2.3.5

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. ......................... 30

2.3.6

int
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

```c
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

2.3.5

```c
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.

2.3.6

```c
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 (→ 2.4.12, page 36)
2.3.7

```c
int HYPRE_SStructGraphAssemble (HYPRE_SStructGraph graph)
```

Finalize the construction of the graph before using

## SStruct Matrices

### Names

2.4.1

```c
define struct hypre_SStructMatrix_struct *HYPRE_SStructMatrix

The matrix object ..............................
```

2.4.2

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

Create a matrix object ...........................
```

2.4.3

```c
int HYPRE_SStructMatrixDestroy (HYPRE_SStructMatrix matrix)

Destroy a matrix object ...........................
```

2.4.4

```c
int HYPRE_SStructMatrixInitialize (HYPRE_SStructMatrix matrix,
    int part,
    int* index, int var, int nentries,
    int* entries, double* values)

Prepare a matrix object for setting coefficient values ................
```

2.4.5

```c
int HYPRE_SStructMatrixSetValues (HYPRE_SStructMatrix matrix,
    int part, int* index, int var, int nentries,
    int* entries, double* values)

Set matrix coefficients index by index. ............................
```

2.4.6

```c
int HYPRE_SStructMatrixAddToValues (HYPRE_SStructMatrix matrix,
    int part, int* index, int var,
    int nentries, int* entries,
    double* values)

Add to matrix coefficients index by index. ............................
```

2.4.7

```c
int HYPRE_SStructMatrixSetBoxValues (HYPRE_SStructMatrix matrix,
    int part, int* ilower, int* iupper,
    int var, int nentries, int* entries,
    double* values)

Set matrix coefficients a box at a time. ............................
```

2.4.8

```c
int
```
2.4.1

typedef struct hypre_SStructMatrix_struct *HYPRE_SStructMatrix

The matrix object

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

Add to matrix coefficients a box at a time.  ....................... 35

2.4.9 int HYPRE_SStructMatrixAssemble (HYPRE_SStructMatrix matrix)

Finalize the construction of the matrix before using  ................. 35

2.4.10 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.  ........ 36

2.4.11 int HYPRE_SStructMatrixSetNSSymmetric (HYPRE_SStructMatrix matrix,
   int symmetric)

Define symmetry properties for all non-stencil matrix entries  ............. 36

2.4.12 int HYPRE_SStructMatrixSetObjectType (HYPRE_SStructMatrix matrix,
   int type)

Set the storage type of the matrix object to be constructed. ................ 36

2.4.13 int HYPRE_SStructMatrixGetObject (HYPRE_SStructMatrix matrix,
   void** object)

Get a reference to the constructed matrix object.  ....................... 37

2.4.14 int HYPRE_SStructMatrixSetComplex (HYPRE_SStructMatrix matrix)

Set the matrix to be complex  ............................................. 37

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

Print the matrix to file.  ................................................. 37
2.4.2

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

Create a matrix object

2.4.3

```c
int HYPRE_SStructMatrixDestroy (HYPRE_SStructMatrix matrix)
```

Destroy a matrix object

2.4.4

```c
int HYPRE_SStructMatrixInitialize (HYPRE_SStructMatrix matrix)
```

Prepare a matrix object for setting coefficient values

2.4.5

```c
int HYPRE_SStructMatrixSetValues (HYPRE_SStructMatrix matrix, int part, int* index, int var, int nentries, int* entries, double* 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).
If the matrix is complex, then values consists of pairs of doubles representing the real and imaginary parts of each complex value.

See Also: HYPRE_SStructMatrixSetComplex (→2.4.14, page 37)

### 2.4.6

```
int HYPRE_SStructMatrixAddToValues (HYPRE_SStructMatrix matrix, int part, int* index, int var, int nentries, int* entries, double* 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.

If the matrix is complex, then values consists of pairs of doubles representing the real and imaginary parts of each complex value.

See Also: HYPRE_SStructMatrixSetComplex (→2.4.14, page 37)

### 2.4.7

```
int HYPRE_SStructMatrixSetBoxValues (HYPRE_SStructMatrix matrix, int part, int* ilower, int* iupper, int var, int nentries, int* entries, double* 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++;
}

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).

If the matrix is complex, then values consists of pairs of doubles representing the real and imaginary parts of each complex value.

See Also: HYPRE_SStructMatrixSetComplex (→2.4.14, page 37)

2.4.8

int HYPRE_SStructMatrixAddToBoxValues (HYPRE_SStructMatrix matrix, int part, int* ilower, int* iupper, int var, int nentries, int* entries, double* 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.

If the matrix is complex, then values consists of pairs of doubles representing the real and imaginary parts of each complex value.

See Also: HYPRE_SStructMatrixSetComplex (→2.4.14, page 37)

2.4.9

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

2.4.10

```c
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.

2.4.11

```c
int HYPRE_SStructMatrixSetNSSymmetric (HYPRE_SStructMatrix matrix, int symmetric)
```

Define symmetry properties for all non-stencil matrix entries

2.4.12

```c
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 (→2.4.13, page 37)
2.4.13

```c
int HYPRE_SStructMatrixGetObject (HYPRE_SStructMatrix matrix, void** object)
```

Get a reference to the constructed matrix object.

See Also: 

HYPRE_SStructMatrixSetObjectType (→2.4.12, page 36)

2.4.14

```c
int HYPRE_SStructMatrixSetComplex (HYPRE_SStructMatrix matrix)
```

Set the matrix to be complex

2.4.15

```c
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* ............................................................. 39

2.5.2 int HYPRE_SStructVectorCreate (MPI_Comm comm,

HYPRE_SStructGrid grid,

HYPRE_SStructVector* vector)

*Create a vector object* ..................................................... 39

2.5.3 int
2.5.4 int HYPRE_SStructVectorDestroy (HYPRE_SStructVector vector)  
Destroy a vector object ............................................. 39

2.5.5 int HYPRE_SStructVectorInitialize (HYPRE_SStructVector vector)  
Prepare a vector object for setting coefficient values .................. 39

2.5.6 int HYPRE_SStructVectorSetValues (HYPRE_SStructVector vector, int part, int* index, int var, double* value)  
Set vector coefficients index by index. .................................. 40

2.5.7 int HYPRE_SStructVectorAddToValues (HYPRE_SStructVector vector, int part, int* index, int var, double* value)  
Add to vector coefficients index by index. ................................ 40

2.5.8 int HYPRE_SStructVectorSetBoxValues (HYPRE_SStructVector vector, int part, int* ilower, int* iupper, int var, double* values)  
Set vector coefficients a box at a time. ................................... 41

2.5.9 int HYPRE_SStructVectorAddToBoxValues (HYPRE_SStructVector vector, int part, int* ilower, int* iupper, int var, double* values)  
Add to vector coefficients a box at a time. .............................. 41

2.5.10 int HYPRE_SStructVectorAssemble (HYPRE_SStructVector vector)  
Finalize the construction of the vector before using ....................... 42

2.5.11 int HYPRE_SStructVectorGetValues (HYPRE_SStructVector vector, int part, int* index, int var, double* value)  
Get vector coefficients index by index. ................................... 42

2.5.12 int HYPRE_SStructVectorGetBoxValues (HYPRE_SStructVector vector, int part, int* ilower, int* iupper, int var, double* values)  
Get vector coefficients a box at a time. ................................... 43

2.5.13 int HYPRE_SStructVectorSetObjectType (HYPRE_SStructVector vector, int type)  
Set the storage type of the vector object to be constructed. ............... 43

2.5.14 int HYPRE_SStructVectorGetObject (HYPRE_SStructVector vector, void** object)  
Get a reference to the constructed vector object. .......................... 43
2.5.1

typedef struct hypre_SStructVector *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

\begin{verbatim}
int HYPRE_SSStructVectorSetValues (HYPRE_SSStructVector vector, int part, int* index, int var, double* value)
\end{verbatim}

Set vector coefficients index by index.

NOTE: For better efficiency, use HYPRE_SSStructVectorSetBoxValues 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.

If the vector is complex, then \texttt{value} consists of a pair of doubles representing the real and imaginary parts of the complex value.

\textbf{See Also:} \ HYPRE_SSStructVectorSetComplex (\textit{\texttt{\rightarrow 2.5.15, page 44}})

2.5.6

\begin{verbatim}
int HYPRE_SSStructVectorAddToValues (HYPRE_SSStructVector vector, int part, int* index, int var, double* value)
\end{verbatim}

Add to vector coefficients index by index.

NOTE: For better efficiency, use HYPRE_SSStructVectorAddToBoxValues 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.

If the vector is complex, then \texttt{value} consists of a pair of doubles representing the real and imaginary parts of the complex value.

\textbf{See Also:} \ HYPRE_SSStructVectorSetComplex (\textit{\texttt{\rightarrow 2.5.15, page 44}})
2.5.7

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

Set vector coefficients a box at a time. The data in \texttt{values} is ordered as follows:

\begin{verbatim}
    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++;
                }
\end{verbatim}

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

If the vector is complex, then \texttt{values} consists of pairs of doubles representing the real and imaginary parts of each complex value.

\textbf{See Also:} HYPRE_SStructVectorSetComplex (\textit{\rightarrow} 2.5.15, page 44)

2.5.8

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

Add to vector coefficients a box at a time. The data in \texttt{values} is ordered as in HYPRE_SStructVectorSetBoxValues.

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

If the vector is complex, then \texttt{values} consists of pairs of doubles representing the real and imaginary parts of each complex value.
Finalize the construction of the vector before using

```c
int HYPRE_SStructVectorAssemble (HYPRE_SStructVector vector)
```

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

```c
int HYPRE_SStructVectorGather (HYPRE_SStructVector vector)
```

Get vector coefficients index by index.

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.

If the vector is complex, then value consists of a pair of doubles representing the real and imaginary parts of the complex value.

See Also: HYPRE_SStructVectorSetComplex (→2.5.15, page 44)
2.5.12

```c
int HYPRE_SStructVectorGetBoxValues (HYPRE_SStructVector vector, int part,
   int* ilower, int* iupper, int var, double* values)
```

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

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

If the vector is complex, then `values` consists of pairs of doubles representing the real and imaginary parts of each complex value.

See Also: `HYPRE_SStructVectorSetComplex` (→ 2.5.15, page 44)

2.5.13

```c
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` (→ 2.5.14, page 43)

2.5.14

```c
int HYPRE_SStructVectorGetObject (HYPRE_SStructVector vector, void** object)
```

Get a reference to the constructed vector object.

See Also: `HYPRE_SStructVectorSetObjectType` (→ 2.5.13, page 43)
2.5.15

```
int HYPRE_SStructVectorSetComplex (HYPRE_SStructVector vector)
```

Set the vector to be complex

2.5.16

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

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

**typedef struct** hypre_IJMatrix_struct *HYPRE_IJMatrix

*The matrix object*

Create a matrix object. 

Destroy a matrix object.

Prepare a matrix object for setting coefficient values.

Sets values for $nrows$ rows or partial rows of the matrix. 

Adds to values for $nrows$ rows or partial rows of the matrix.
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HYPRE_IJMatrixAssemble</td>
<td>Finalize the construction of the matrix before using</td>
</tr>
<tr>
<td>HYPRE_IJMatrixGetRowCounts</td>
<td>Gets number of nonzeros elements for nrows rows specified in rows and returns them in ncols, which needs to be allocated by the user</td>
</tr>
<tr>
<td>HYPRE_IJMatrixGetValues</td>
<td>Gets values for nrows rows or partial rows of the matrix</td>
</tr>
<tr>
<td>HYPRE_IJMatrixSetObjectType</td>
<td>Set the storage type of the matrix object to be constructed.</td>
</tr>
<tr>
<td>HYPRE_IJMatrixGetObjectType</td>
<td>Get the storage type of the constructed matrix object</td>
</tr>
<tr>
<td>HYPRE_IJMatrixGetLocalRange</td>
<td>Gets range of rows owned by this processor and range of column partitioning for this processor</td>
</tr>
<tr>
<td>HYPRE_IJMatrixGetObject</td>
<td>Get a reference to the constructed matrix object</td>
</tr>
<tr>
<td>HYPRE_IJMatrixSetRowSizes</td>
<td>(Optional) Set the max number of nonzeros to expect in each row.</td>
</tr>
<tr>
<td>HYPRE_IJMatrixSetDiagOffdSizes</td>
<td>(Optional) Set the max number of nonzeros to expect in each row of the diagonal and off-diagonal blocks.</td>
</tr>
<tr>
<td>HYPRE_IJMatrixSetMaxOffProcElmts</td>
<td>(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.</td>
</tr>
<tr>
<td>HYPRE_IJMatrixRead</td>
<td>Read the matrix from file</td>
</tr>
<tr>
<td>HYPRE_IJMatrixPrint</td>
<td>Print the matrix to file</td>
</tr>
</tbody>
</table>
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 \texttt{ilower} and \texttt{iupper}. The row data is required to be such that the value of \texttt{ilower} on any process \(p\) be exactly one more than the value of \texttt{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, \texttt{jlower} and \texttt{jupper} typically should match \texttt{ilower} and \texttt{iupper}, respectively. For rectangular matrices, \texttt{jlower} and \texttt{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 \texttt{jlower} and \texttt{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

```c
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

```c
int HYPRE_IJMatrixSetValues (HYPRE_IJMatrix matrix, int nrows, int* ncols,
const int* rows, const int* cols, const double* 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.

Not collective.

3.1.6

```c
int HYPRE_IJMatrixAddToValues (HYPRE_IJMatrix matrix, int nrows, int* ncols,
const int* rows, const int* cols, const double* 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.

Not collective.

3.1.7

```c
int HYPRE_IJMatrixAssemble (HYPRE_IJMatrix matrix)
```
Finalize the construction of the matrix before using

3.1.8

```c
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

```c
int HYPRE_IJMatrixGetValues (HYPRE_IJMatrix matrix, int nrows, int* ncols, int* rows, int* cols, double* values)
```

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

3.1.10

```c
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 50)
3.1.11

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

Get the storage type of the constructed matrix object

3.1.12

```c
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

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

Get a reference to the constructed matrix object.

**See Also:** [HYPRE_IJMatrixSetObjectType](#), page 49

3.1.14

```c
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) Set the max number of nonzeros to expect 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_IJMatrixRead (const char* filename, MPI_Comm comm, int type, HYPRE_IJMatrix* matrix)
```

Read the matrix from file. This is mainly for debugging purposes.
### 3.1.18

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

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

### 3.2 IJ Vectors

#### 3.2.1

```c
typedef struct hypre_IJVector_struct *HYPRE_IJVector

// The vector object
```

#### 3.2.2

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

// Create a vector object.
```

#### 3.2.3

```c
int HYPRE_IJVectorDestroy (HYPRE_IJVector vector)

// Destroy a vector object.
```

#### 3.2.4

```c
int HYPRE_IJVectorInitialize (HYPRE_IJVector vector)

// Prepare a vector object for setting coefficient values.
```

#### 3.2.5

```c
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 signifi-
// cantly improve the efficiency of matrix construction, and should always be
// utilized if possible.
```

#### 3.2.6

```c
int HYPRE_IJVectorSetValues (HYPRE_IJVector vector, int nvalues,
                             const int* indices, const double* values)

// Sets values in vector.
```

#### 3.2.7

```c
int HYPRE_IJVectorAddToValues (HYPRE_IJVector vector, int nvalues,
                               const int* indices, const double* values)

// Adds to values in vector.
```

#### 3.2.8

```c
int HYPRE_IJVectorAssemble (HYPRE_IJVector vector)

// Finalize the construction of the vector before using.
```

#### 3.2.9

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

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.

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

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

3.2.1
typedef struct hypre_IJVector_struct *HYPRE_IJVector

The vector object

3.2.2

typedef struct hypre_IJVector_struct *HYPRE_IJVector

Create a vector object. Each process owns some unique consecutive range of vector unknowns, indicated by the global indices \texttt{jlower} and \texttt{jupper}. The data is required to be such that the value of \texttt{jlower} on any
process $p$ be exactly one more than the value of $\text{jupper}$ 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

\begin{verbatim}
int HYPRE_IJVectorSetValues (HYPRE_IJVector vector, int nvalues, const int* indices, const double* values)
\end{verbatim}

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.

Not collective.

3.2.7

\begin{verbatim}
int HYPRE_IJVectorAddToValues (HYPRE_IJVector vector, int nvalues, const int* indices, const double* values)
\end{verbatim}

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

Not collective.

3.2.8

\begin{verbatim}
int HYPRE_IJVectorAssemble (HYPRE_IJVector vector)
\end{verbatim}

Finalize the construction of the vector before using.

3.2.9

\begin{verbatim}
int HYPRE_IJVectorGetValues (HYPRE_IJVector vector, int nvalues, const int* indices, double* values)
\end{verbatim}

Gets values in vector. Usage details are analogous to HYPRE_IJVectorSetValues.
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 56)

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.
3.2.14

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.

3.2.15

int HYPRE_IJVectorPrint (HYPRE_IJVector vector, const char* filename)

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

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

4.1 Struct Solvers

typedef struct hypre_StructSolver_struct *HYPRE_StructSolver

The solver object
The solver object

### Struct Jacobi Solver

#### Names

4.2.1  
**HYPRE_StructJacobiCreate**  
(MPI_Comm comm,  
HYPRE_StructSolver* solver)  

*Create a solver object.*  

4.2.2  
**HYPRE_StructJacobiDestroy**  
(HYPRE_StructSolver solver)  

*Destroy a solver object.*  

4.2.3  
**HYPRE_StructJacobiSetup**  
(HYPRE_StructSolver solver,  
HYPRE_StructMatrix A,  
HYPRE_StructVector b,  
HYPRE_StructVector x)  

*Prepare to solve the system.*  

4.2.4  
**HYPRE_StructJacobiSolve**  
(HYPRE_StructSolver solver,  
HYPRE_StructMatrix A,  
HYPRE_StructVector b,  
HYPRE_StructVector x)  

*Solve the system.*  

4.2.5  
**HYPRE_StructJacobiSetTol**  
(HYPRE_StructSolver solver, double tol)  

*(Optional) Set the convergence tolerance.*  

4.2.6  
**HYPRE_StructJacobiSetMaxIter**  
(HYPRE_StructSolver solver, int max_iter)  

*(Optional) Set maximum number of iterations.*  

4.2.7  
**HYPRE_StructJacobiSetZeroGuess**  
(HYPRE_StructSolver solver)  

*(Optional) Use a zero initial guess.*  

4.2.8  
**HYPRE_StructJacobiSetNonZeroGuess**  
(HYPRE_StructSolver solver)  

*(Optional) Use a nonzero initial guess.*  

4.2.9  
**HYPRE_StructJacobiGetNumIterations**  
(HYPRE_StructSolver solver,  
int* num_iterations)  

*Return the number of iterations taken.*  

4.2.10  
int
HYPRE_StructJacobiGetFinalRelativeResidualNorm

\[ \text{Return the norm of the final relative residual} \]

\begin{verbatim}
int HYPRE_StructJacobiCreate (MPI_Comm comm, HYPRE_StructSolver* solver)
\end{verbatim}

Create a solver object.

\begin{verbatim}
int HYPRE_StructJacobiDestroy (HYPRE_StructSolver solver)
\end{verbatim}

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.

\begin{verbatim}
int HYPRE_StructJacobiSetup (HYPRE_StructSolver solver, HYPRE_StructMatrix A, HYPRE_StructVector b, HYPRE_StructVector x)
\end{verbatim}

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

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

Solve the system

4.2.5

```c
int HYPRE_StructJacobiSetTol (HYPRE_StructSolver solver, double tol)
```

(Optional) Set the convergence tolerance

4.2.6

```c
int HYPRE_StructJacobiSetMaxIter (HYPRE_StructSolver solver, int max_iter)
```

(Optional) Set maximum number of iterations

4.2.7

```c
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 computational cost.
4.2.8

```c
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

```c
int HYPRE_StructJacobiGetNumIterations (HYPRE_StructSolver solver, int* num_iterations)
```

Return the number of iterations taken

4.2.10

```c
int HYPRE_StructJacobiGetFinalRelativeResidualNorm (HYPRE_StructSolver solver, double* 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

4.3.2 int `HYPRE_StructPFMGDestructor` (HYPRE_StructSolver solver)

Destroy a solver object

This page has been automatically generated with DOC++

DOC++ is ©1995 by Roland Wunderling
Malte Zöckler
4 Struct Solvers

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

*Prepare to solve the system.* ........................................

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,
int max_iter)

*(Optional) Set the convergence tolerance* ......................

4.3.6 int

**HYPRE_StructPFMGSetMaxIter** (HYPRE_StructSolver solver,
int max_levels)

*(Optional) Set maximum number of multigrid levels* ..........

4.3.7 int

**HYPRE_StructPFMGSetRelChange** (HYPRE_StructSolver solver,
int rel_change)

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

4.3.8 int

**HYPRE_StructPFMGSetZeroGuess** (HYPRE_StructSolver solver)

*(Optional) Use a zero initial guess.* ............................

4.3.9 int

**HYPRE_StructPFMGSetNonZeroGuess** (HYPRE_StructSolver solver)

*(Optional) Use a nonzero initial guess.* .........................

4.3.10 int

**HYPRE_StructPFMGSetRelaxType** (HYPRE_StructSolver solver,
int relax_type)

*(Optional) Set relaxation type.* ..................................

4.3.11 int

**HYPRE_StructPFMGSetRAPType** (HYPRE_StructSolver solver,
int rap_type)

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

4.3.12 int

**HYPRE_StructPFMGSetNumPreRelax** (HYPRE_StructSolver solver,
int num_pre_relax)

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

4.3.13 int

**HYPRE_StructPFMGSetMaxLevels** (HYPRE_StructSolver solver,
int max_levels)

*(Optional) Set maximum number of multigrid grid levels* ....

4.3.14 int

**HYPRE_StructPFMGSetMaxLevels** (HYPRE_StructSolver solver,
int max_levels)

*(Optional) Set maximum number of multigrid grid levels* ...

4.3.15 int

**HYPRE_StructPFMGSetRelChange** (HYPRE_StructSolver solver,
int rel_change)

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

4.3.16 int

**HYPRE_StructPFMGSetZeroGuess** (HYPRE_StructSolver solver)

*(Optional) Use a zero initial guess.* ............................

4.3.17 int

**HYPRE_StructPFMGSetNonZeroGuess** (HYPRE_StructSolver solver)

*(Optional) Use a nonzero initial guess.* .........................

4.3.18 int

**HYPRE_StructPFMGSetRelaxType** (HYPRE_StructSolver solver,
int relax_type)

*(Optional) Set relaxation type.* ..................................

4.3.19 int

**HYPRE_StructPFMGSetRAPType** (HYPRE_StructSolver solver,
int rap_type)

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

4.3.20 int

**HYPRE_StructPFMGSetNumPreRelax** (HYPRE_StructSolver solver,
int num_pre_relax)

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

4.3.21 int

**HYPRE_StructPFMGSetMaxLevels** (HYPRE_StructSolver solver,
int max_levels)

*(Optional) Set maximum number of multigrid grid levels* ...

4.3.22 int

**HYPRE_StructPFMGSetRelChange** (HYPRE_StructSolver solver,
int rel_change)

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

4.3.23 int

**HYPRE_StructPFMGSetZeroGuess** (HYPRE_StructSolver solver)

*(Optional) Use a zero initial guess.* ............................

4.3.24 int

**HYPRE_StructPFMGSetNonZeroGuess** (HYPRE_StructSolver solver)

*(Optional) Use a nonzero initial guess.* .........................

4.3.25 int

**HYPRE_StructPFMGSetRelaxType** (HYPRE_StructSolver solver,
int relax_type)

*(Optional) Set relaxation type.* ..................................

4.3.26 int

**HYPRE_StructPFMGSetRAPType** (HYPRE_StructSolver solver,
int rap_type)

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

4.3.27 int

**HYPRE_StructPFMGSetNumPreRelax** (HYPRE_StructSolver solver,
int num_pre_relax)

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

4.3.28 int

**HYPRE_StructPFMGSetMaxLevels** (HYPRE_StructSolver solver,
int max_levels)

*(Optional) Set maximum number of multigrid grid levels* ...

4.3.29 int

**HYPRE_StructPFMGSetRelChange** (HYPRE_StructSolver solver,
int rel_change)

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

4.3.30 int

**HYPRE_StructPFMGSetZeroGuess** (HYPRE_StructSolver solver)

*(Optional) Use a zero initial guess.* ............................

4.3.31 int

**HYPRE_StructPFMGSetNonZeroGuess** (HYPRE_StructSolver solver)

*(Optional) Use a nonzero initial guess.* .........................

4.3.32 int

**HYPRE_StructPFMGSetRelaxType** (HYPRE_StructSolver solver,
int relax_type)

*(Optional) Set relaxation type.* ..................................

4.3.33 int

**HYPRE_StructPFMGSetRAPType** (HYPRE_StructSolver solver,
int rap_type)

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

4.3.34 int

**HYPRE_StructPFMGSetNumPreRelax** (HYPRE_StructSolver solver,
int num_pre_relax)

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

4.3.35 int

**HYPRE_StructPFMGSetMaxLevels** (HYPRE_StructSolver solver,
int max_levels)

*(Optional) Set maximum number of multigrid grid levels* ...

4.3.36 int

**HYPRE_StructPFMGSetRelChange** (HYPRE_StructSolver solver,
int rel_change)

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

4.3.37 int

**HYPRE_StructPFMGSetZeroGuess** (HYPRE_StructSolver solver)

*(Optional) Use a zero initial guess.* ............................

4.3.38 int

**HYPRE_StructPFMGSetNonZeroGuess** (HYPRE_StructSolver solver)

*(Optional) Use a nonzero initial guess.* .........................

4.3.39 int

**HYPRE_StructPFMGSetRelaxType** (HYPRE_StructSolver solver,
int relax_type)

*(Optional) Set relaxation type.* ..................................

4.3.40 int

**HYPRE_StructPFMGSetRAPType** (HYPRE_StructSolver solver,
int rap_type)

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

4.3.41 int

**HYPRE_StructPFMGSetNumPreRelax** (HYPRE_StructSolver solver,
int num_pre_relax)

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

4.3.42 int

**HYPRE_StructPFMGSetMaxLevels** (HYPRE_StructSolver solver,
int max_levels)

*(Optional) Set maximum number of multigrid grid levels* ...

4.3.43 int

**HYPRE_StructPFMGSetRelChange** (HYPRE_StructSolver solver,
int rel_change)

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

4.3.44 int

**HYPRE_StructPFMGSetZeroGuess** (HYPRE_StructSolver solver)

*(Optional) Use a zero initial guess.* ............................

4.3.45 int

**HYPRE_StructPFMGSetNonZeroGuess** (HYPRE_StructSolver solver)

*(Optional) Use a nonzero initial guess.* .........................

4.3.46 int

**HYPRE_StructPFMGSetRelaxType** (HYPRE_StructSolver solver,
int relax_type)

*(Optional) Set relaxation type.* ..................................

4.3.47 int

**HYPRE_StructPFMGSetRAPType** (HYPRE_StructSolver solver,
int rap_type)

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

4.3.48 int

**HYPRE_StructPFMGSetNumPreRelax** (HYPRE_StructSolver solver,
int num_pre_relax)

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

4.3.49 int

**HYPRE_StructPFMGSetMaxLevels** (HYPRE_StructSolver solver,
int max_levels)

*(Optional) Set maximum number of multigrid grid levels* ...

4.3.50 int

**HYPRE_StructPFMGSetRelChange** (HYPRE_StructSolver solver,
int rel_change)

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

4.3.51 int

**HYPRE_StructPFMGSetZeroGuess** (HYPRE_StructSolver solver)

*(Optional) Use a zero initial guess.* ............................

4.3.52 int

**HYPRE_StructPFMGSetNonZeroGuess** (HYPRE_StructSolver solver)

*(Optional) Use a nonzero initial guess.* .........................

4.3.53 int

**HYPRE_StructPFMGSetRelaxType** (HYPRE_StructSolver solver,
int relax_type)

*(Optional) Set relaxation type.* ..................................

4.3.54 int

**HYPRE_StructPFMGSetRAPType** (HYPRE_StructSolver solver,
int rap_type)

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

4.3.55 int

**HYPRE_StructPFMGSetNumPreRelax** (HYPRE_StructSolver solver,
int num_pre_relax)

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

4.3.56 int

**HYPRE_StructPFMGSetMaxLevels** (HYPRE_StructSolver solver,
int max_levels)

*(Optional) Set maximum number of multigrid grid levels* ...

This page has been automatically generated with DOC++

DOC++ is ©1995 by Roland Wunderling
Malte Zöckler

63
4 Struct Solvers

4.3.15 int HYPRE_StructPFMGSetSkipRelax (HYPRE_StructSolver solver, int skip_relax)

(Optional) Skip relaxation on certain grids for isotropic problems. .......

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, double* norm)

Return the norm of the final relative residual ..........................

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

```c
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

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

Solve the system

4.3.5

```c
int HYPRE_StructPFMGSetTol (HYPRE_StructSolver solver, double tol)
```

(Optional) Set the convergence tolerance

4.3.6

```c
int HYPRE_StructPFMGSetMaxIter (HYPRE_StructSolver solver, int max_iter)
```

(Optional) Set maximum number of iterations
4.3.7

```c
int HYPRE_StructPFMGSetMaxLevels (HYPRE_StructSolver solver, int max_levels)
```

(Optional) Set maximum number of multigrid grid levels

4.3.8

```c
int HYPRE_StructPFMGSetRelChange (HYPRE_StructSolver solver, int rel_change)
```

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

4.3.9

```c
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 computational cost.

4.3.10

```c
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

\begin{verbatim}
int HYPRE_StructPFMGGetNumIterations (HYPRE_StructSolver solver, int* num_iterations)
\end{verbatim}

Return the number of iterations taken

4.3.19

\begin{verbatim}
int HYPRE_StructPFMGGetFinalRelativeResidualNorm (HYPRE_StructSolver solver, double* norm)
\end{verbatim}

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)
\begin{verbatim}
Create a solver object ..............................
\end{verbatim}

4.4.2 int HYPRE_StructSMGDestroy (HYPRE_StructSolver solver)
\begin{verbatim}
Destroy a solver object ..............................
\end{verbatim}

4.4.3 int HYPRE_StructSMGSetup (HYPRE_StructSolver solver, HYPRE_StructMatrix A, HYPRE_StructVector b, HYPRE_StructVector x)
\begin{verbatim}
Prepare to solve the system. ..............................
\end{verbatim}

4.4.4 int HYPRE_StructSMGSolve (HYPRE_StructSolver solver, HYPRE_StructMatrix A, HYPRE_StructVector b, HYPRE_StructVector x)
\begin{verbatim}
Solve the system ..............................
\end{verbatim}

4.4.5 int
4.4.6 int HYPRE_StructSMGSetTol (HYPRE_StructSolver solver, double tol)
   (Optional) Set the convergence tolerance ............................ 72

4.4.7 int HYPRE_StructSMGSetMaxIter (HYPRE_StructSolver solver, int max_iter)
   (Optional) Set maximum number of iterations .......................... 72

4.4.8 int HYPRE_StructSMGSetRelChange (HYPRE_StructSolver solver, int rel_change)
   (Optional) Additionally require that the relative difference in successive it-
   erates be small .................................................................. 72

4.4.9 int HYPRE_StructSMGSetZeroGuess (HYPRE_StructSolver solver)
   (Optional) Use a zero initial guess. ........................................ 72

4.4.10 int HYPRE_StructSMGSetNumPreRelax (HYPRE_StructSolver solver, int num_pre_relax)
   (Optional) Set number of relaxation sweeps before coarse-grid correction . 73

4.4.11 int HYPRE_StructSMGSetNumPostRelax (HYPRE_StructSolver solver, int num_post_relax)
   (Optional) Set number of relaxation sweeps after coarse-grid correction .. 73

4.4.12 int HYPRE_StructSMGSetLogging (HYPRE_StructSolver solver, int logging)
   (Optional) Set the amount of logging to do ............................. 73

4.4.13 int HYPRE_StructSMGSetPrintLevel (HYPRE_StructSolver solver, int print_level)
   (Optional) Set the amount of printing to do to the screen .......... 73

4.4.14 int HYPRE_StructSMGGetNumIterations (HYPRE_StructSolver solver, int* num_iterations)
   Return the number of iterations taken ................................. 74

4.4.15 int HYPRE_StructSMGGetFinalRelativeResidualNorm (HYPRE_StructSolver solver, double* norm)
   Return the norm of the final relative residual .......................... 74
4.4.1

```c
int HYPRE_StructSMGCreate (MPI_Comm comm, HYPRE_StructSolver* solver)
```

Create a solver object

4.4.2

```c
int HYPRE_StructSMGDestroy (HYPRE_StructSolver solver)
```

Destroy a solver object

4.4.3

```c
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

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

Solve the system
4.4.5

int \texttt{HYPRE\_StructSMGSetTol} (HYPRE\_StructSolver solver, double tol)

(Optional) Set the convergence tolerance

4.4.6

int \texttt{HYPRE\_StructSMGSetMaxIter} (HYPRE\_StructSolver solver, int max_iter)

(Optional) Set maximum number of iterations

4.4.7

int \texttt{HYPRE\_StructSMGSetRelChange} (HYPRE\_StructSolver solver, int rel_change)

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

4.4.8

int \texttt{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 computational cost.
4.4.9

\texttt{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 \texttt{SetZeroGuess}.

4.4.10

\texttt{int HYPRE_StructSMGSetNumPreRelax} (HYPRE_StructSolver solver, int num_pre_relax)

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

4.4.11

\texttt{int HYPRE_StructSMGSetNumPostRelax} (HYPRE_StructSolver solver, int num_post_relax)

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

4.4.12

\texttt{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, double* norm)

Return the norm of the final relative residual

4.5

Struct PCG Solver

Names

4.5.1 int HYPRE_StructPCGCreate (MPI_Comm comm, HYPRE_StructSolver* solver)

Create a solver object

4.5.2 int HYPRE_StructPCGDestroy (HYPRE_StructSolver solver)

Destroy a solver object
4.5.4 \textbf{HYPRE\_StructPCGSetup} (HYPRE\_StructSolver solver,  
HYPRE\_StructMatrix A,  
HYPRE\_StructVector b,  
HYPRE\_StructVector x)  
\begin{flushright}
Prepare to solve the system.
\end{flushright}

4.5.5 \textbf{int HYPRE\_StructPCGSolve} (HYPRE\_StructSolver solver,  
HYPRE\_StructMatrix A,  
HYPRE\_StructVector b,  
HYPRE\_StructVector x)  
\begin{flushright}
Solve the system
\end{flushright}

4.5.6 \textbf{int HYPRE\_StructPCGSetTol} (HYPRE\_StructSolver solver, double tol)  
(\textit{Optional}) Set the convergence tolerance

4.5.7 \textbf{int HYPRE\_StructPCGSetMaxIter} (HYPRE\_StructSolver solver, int max_iter)  
(\textit{Optional}) Set maximum number of iterations

4.5.8 \textbf{int HYPRE\_StructPCGSetTwoNorm} (HYPRE\_StructSolver solver,  
int two_norm)  
(\textit{Optional}) Use the two-norm in stopping criteria

4.5.9 \textbf{int HYPRE\_StructPCGSetPrecond} (HYPRE\_StructSolver solver,  
HYPRE\_PtrToStructSolverFcn precond,  
HYPRE\_PtrToStructSolverFcn precond\_setup,  
HYPRE\_StructSolver precond\_solver)  
(\textit{Optional}) Set the preconditioner to use

4.5.10 \textbf{int HYPRE\_StructPCGSetLogging} (HYPRE\_StructSolver solver, int logging)  
(\textit{Optional}) Set the amount of logging to do

4.5.11 \textbf{int HYPRE\_StructPCGSetPrintLevel} (HYPRE\_StructSolver solver, int level)  
(\textit{Optional}) Set the amount of printing to do to the screen

4.5.12 \textbf{int HYPRE\_StructPCGGetNumIterations} (HYPRE\_StructSolver solver,  
int* num\_iterations)  
\begin{flushright}
Return the number of iterations taken
\end{flushright}

4.5.13 \textbf{int HYPRE\_StructPCGGetFinalRelativeResidualNorm} (HYPRE\_StructSolver solver, double* norm)  
\begin{flushright}
Return the norm of the final relative residual
\end{flushright}
**HYPRE_StructPCGGetResidual** (HYPRE_StructSolver solver, void** residual)

Return the residual ........................................... 79

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

Setup routine for diagonal preconditioning ....................... 80

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

Solve routine for diagonal preconditioning ....................... 80

### 4.5.1

int **HYPRE_StructPCGCreate** (MPI_Comm comm, HYPRE_StructSolver* solver)

Create a solver object

### 4.5.2

int **HYPRE_StructPCGDestroy** (HYPRE_StructSolver solver)

Destroy a solver object

### 4.5.3

int **HYPRE_StructPCGSetup** (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

```c
int HYPRE_StructPCGSolve (HYPRE_StructSolver solver, HYPRE_StructMatrix A, HYPRE_StructVector b, HYPRE_StructVector x)
```

Solve the system

### 4.5.5

```c
int HYPRE_StructPCGSetTol (HYPRE_StructSolver solver, double tol)
```

(Optional) Set the convergence tolerance

### 4.5.6

```c
int HYPRE_StructPCGSetMaxIter (HYPRE_StructSolver solver, int max_iter)
```

(Optional) Set maximum number of iterations

### 4.5.7

```c
int HYPRE_StructPCGSetTwoNorm (HYPRE_StructSolver solver, int two_norm)
```

(Optional) Use the two-norm in stopping criteria
4.5.8

```c
int HYPRE_StructPCGSetRelChange (HYPRE_StructSolver solver, int rel_change)
```

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

4.5.9

```c
int HYPRE_StructPCGSetPrecond (HYPRE_StructSolver solver,
                               HYPRE_PtrToStructSolverFcn precond,
                               HYPRE_PtrToStructSolverFcn precond_setup,
                               HYPRE_StructSolver precond_solver)
```

(Optional) Set the preconditioner to use

4.5.10

```c
int HYPRE_StructPCGSetLogging (HYPRE_StructSolver solver, int logging)
```

(Optional) Set the amount of logging to do

4.5.11

```c
int HYPRE_StructPCGSetPrintLevel (HYPRE_StructSolver solver, int level)
```

(Optional) Set the amount of printing to do to the screen
4.5.12

\begin{verbatim}
int HYPRE_StructPCGGetNumIterations (HYPRE_StructSolver solver, int* num_iterations)
\end{verbatim}

Return the number of iterations taken

4.5.13

\begin{verbatim}
int HYPRE_StructPCGGetFinalRelativeResidualNorm (HYPRE_StructSolver solver, double* norm)
\end{verbatim}

Return the norm of the final relative residual

4.5.14

\begin{verbatim}
int HYPRE_StructPCGGetResidual (HYPRE_StructSolver solver, void** residual)
\end{verbatim}

Return the residual

4.5.15

\begin{verbatim}
int HYPRE_StructDiagScaleSetup (HYPRE_StructSolver solver, HYPRE_StructMatrix A, HYPRE_StructVector y, HYPRE_StructVector x)
\end{verbatim}

Setup routine for diagonal preconditioning
4.5.16

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

Solve routine for diagonal preconditioning

---

4.6

**Struct GMRES Solver**

### Names

4.6.1  
int  
**HYPRE_StructGMRESCreate** ( MPI_Comm comm,  
HYPRE_StructSolver* solver )  

Create a solver object  


---

4.6.2  
int  
**HYPRE_StructGMRESDestroy** ( HYPRE_StructSolver solver )  

Destroy a solver object  


---

4.6.3  
int  
**HYPRE_StructGMRESSetup** ( HYPRE_StructSolver solver,  
HYPRE_StructMatrix A,  
HYPRE_StructVector b,  
HYPRE_StructVector x )  

Prepare to solve the system  


---

4.6.4  
int  
**HYPRE_StructGMRESSolve** ( HYPRE_StructSolver solver,  
HYPRE_StructMatrix A,  
HYPRE_StructVector b,  
HYPRE_StructVector x )  

Solve the system  


---

4.6.5  
int  
**HYPRE_StructGMRESSetTol** ( HYPRE_StructSolver solver,  
   double tol )  

(Optional) Set the convergence tolerance  


---

4.6.6  
int  
**HYPRE_StructGMRESSetMaxIter** ( HYPRE_StructSolver solver,  
   int max_iter )  

(Optional) Set maximum number of iterations  


---

4.6.7  
int
4 Struct Solvers

4.6.8 \textbf{int} \textbf{HYPRE\\_StructGMRESSetLogging} ( HYPRE\\_StructSolver solver, int logging )

\textit{(Optional) Set the amount of logging to do} ................. \textcolor{red}{83}

4.6.10 \textbf{int} \textbf{HYPRE\\_StructGMRESSetPrintLevel} ( HYPRE\\_StructSolver solver, int level )

\textit{(Optional) Set the amount of printing to do to the screen} ............ \textcolor{red}{83}

4.6.12 \textbf{int} \textbf{HYPRE\\_StructGMRESGetResidual} ( HYPRE\\_StructSolver solver, void** residual)

\textit{Return the residual} ........................................................... \textcolor{red}{84}

4.6.1 \textbf{int} \textbf{HYPRE\\_StructGMRESCreate} ( MPI\\_Comm comm, HYPRE\\_StructSolver* solver )

Create a solver object

4.6.2 \textbf{int} \textbf{HYPRE\\_StructGMRESDestroy} ( HYPRE\\_StructSolver solver )
Destroy a solver object

4.6.3

```c
int HYPRE_StructGMRESSetup ( 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.

4.6.4

```c
int HYPRE_StructGMRESSolve ( HYPRE_StructSolver solver,
HYPRE_StructMatrix A, HYPRE_StructVector b, HYPRE_StructVector x )
```

Solve the system

4.6.5

```c
int HYPRE_StructGMRESSetTol ( HYPRE_StructSolver solver, double tol )
```

(Optional) Set the convergence tolerance

4.6.6

```c
int HYPRE_StructGMRESSetMaxIter ( HYPRE_StructSolver solver, int max_iter )
```

(Optional) Set maximum number of iterations
4.6.7

\texttt{HYPRE_StructGMRESSetPrecond ( HYPRE_StructSolver solver, HYPRE_PtrToStructSolverFcn precond, HYPRE_PtrToStructSolverFcn precond\_setup, HYPRE_StructSolver precond\_solver )}

(Optional) Set the preconditioner to use

4.6.8

\texttt{int HYPRE_StructGMRESSetLogging ( HYPRE_StructSolver solver, int logging )}

(Optional) Set the amount of logging to do

4.6.9

\texttt{int HYPRE_StructGMRESSetPrintLevel ( HYPRE_StructSolver solver, int level )}

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

4.6.10

\texttt{int HYPRE_StructGMRESSetNumIterations ( HYPRE_StructSolver solver, int* num\_iterations )}

Return the number of iterations taken
4.6.11

```c
int HYPRE_StructGMRESGetFinalRelativeResidualNorm ( HYPRE_StructSolver solver, double* norm )
```

Return the norm of the final relative residual

4.6.12

```c
int HYPRE_StructGMRESGetResidual ( HYPRE_StructSolver solver, void** residual)
```

Return the residual

4.7

Struct BiCGSTAB Solver

Names

4.7.1 int `HYPRE_StructBiCGSTABCreate` ( MPI_Comm comm,
HYPRE_StructSolver* solver )

Create a solver object ................................. 85

4.7.2 int `HYPRE_StructBiCGSTABDestroy` ( HYPRE_StructSolver solver )

Destroy a solver object ................................. 86

4.7.3 int `HYPRE_StructBiCGSTABSetup` ( HYPRE_StructSolver solver,
HYPRE_StructMatrix A,
HYPRE_StructVector b,
HYPRE_StructVector x )

Prepare to solve the system. ................................. 86

4.7.4 int `HYPRE_StructBiCGSTABSolve` ( HYPRE_StructSolver solver,
HYPRE_StructMatrix A,
HYPRE_StructVector b,
HYPRE_StructVector x )

Solve the system ................................. 86
4.7.1

int HYPRE_StructBiCGSTABCreate ( MPI_Comm comm, HYPRE_StructSolver* solver )

4.7.6

int HYPRE_StructBiCGSTABSetTol ( HYPRE_StructSolver solver, double tol )
(Optional) Set the convergence tolerance

86

4.7.7

int HYPRE_StructBiCGSTABSetMaxIter ( HYPRE_StructSolver solver,
int max_iter )
(Optional) Set maximum number of iterations

87

4.7.8

int HYPRE_StructBiCGSTABSetLogging ( HYPRE_StructSolver solver,
int logging )
(Optional) Set the amount of logging to do

88

4.7.9

int HYPRE_StructBiCGSTABSetPrintLevel ( HYPRE_StructSolver solver,
int level )
(Optional) Set the amount of printing to do to the screen

88

4.7.10

int HYPRE_StructBiCGSTABGetNumIterations ( HYPRE_StructSolver solver,
int* num_iterations )
Return the number of iterations taken

88

4.7.11

int HYPRE_StructBiCGSTABGetFinalRelativeResidualNorm ( HYPRE_StructSolver solver,
double* norm )
Return the norm of the final relative residual

88

4.7.12

int HYPRE_StructBiCGSTABGetResidual ( HYPRE_StructSolver solver,
void** residual )
Return the residual

88
Create a solver object

4.7.2

int HYPRE_StructBiCGSTABDestroy ( HYPRE_StructSolver solver )

Destroy a solver object

4.7.3

int HYPRE_StructBiCGSTABSetup ( 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.

4.7.4

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

Solve the system

4.7.5

int HYPRE_StructBiCGSTABSetTol ( HYPRE_StructSolver solver, double tol )

(Optional) Set the convergence tolerance
4.7.6

```c
int HYPRE_StructBiCGSTABSetMaxIter ( HYPRE_StructSolver solver, int max_iter )
```

(Optional) Set maximum number of iterations

4.7.7

```c
int HYPRE_StructBiCGSTABSetPrecond ( HYPRE_StructSolver solver, HYPRE_PtrToStructSolverFcn precond, HYPRE_PtrToStructSolverFcn precond_setup, HYPRE_StructSolver precond_solver )
```

(Optional) Set the preconditioner to use

4.7.8

```c
int HYPRE_StructBiCGSTABSetLogging ( HYPRE_StructSolver solver, int logging )
```

(Optional) Set the amount of logging to do

4.7.9

```c
int HYPRE_StructBiCGSTABSetPrintLevel ( HYPRE_StructSolver solver, int level )
```

(Optional) Set the amount of printing to do to the screen
4.7.10

\begin{verbatim}
int HYPRE_StructBiCGSTABGetNumIterations ( HYPRE_StructSolver solver, int* num_iterations )
\end{verbatim}

Return the number of iterations taken

4.7.11

\begin{verbatim}
int HYPRE_StructBiCGSTABGetFinalRelativeResidualNorm ( HYPRE_StructSolver solver, double* norm )
\end{verbatim}

Return the norm of the final relative residual

4.7.12

\begin{verbatim}
int HYPRE_StructBiCGSTABGetResidual ( HYPRE_StructSolver solver, void** residual)
\end{verbatim}

Return the residual

4.8

Struct Hybrid Solver

Names

4.8.1 \begin{verbatim}
int HYPRE_StructHybridCreate (MPI_Comm comm, HYPRE_StructSolver* solver)
\end{verbatim}

Create a solver object

4.8.2 \begin{verbatim}
int HYPRE_StructHybridDestroy (HYPRE_StructSolver solver)
\end{verbatim}

Destroy a solver object
4.8.4 int HYPRE_StructHybridSetup (HYPRE_StructSolver solver, 
    HYPRE_StructMatrix A, 
    HYPRE_StructVector b, 
    HYPRE_StructVector x)

    Prepare to solve the system.  

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

    Solve the system

4.8.6 int HYPRE_StructHybridSetTol (HYPRE_StructSolver solver, 
    double tol)

    (Optional) Set the convergence tolerance

4.8.7 int HYPRE_StructHybridSetDSCGMaxIter (HYPRE_StructSolver solver, 
    int ds_max_its)

    (Optional) Set maximum number of iterations for diagonal scaling (DS).

4.8.8 int HYPRE_StructHybridSetPCGMaxIter (HYPRE_StructSolver solver, 
    int pre_max_its)

    (Optional) Set maximum number of iterations for general preconditioner (PRE).

4.8.9 int HYPRE_StructHybridSetTwoNorm (HYPRE_StructSolver solver, 
    int two_norm)

    (Optional) Use the two-norm in stopping criteria

4.8.10 int HYPRE_StructHybridSetRelChange (HYPRE_StructSolver solver, 
    int rel_change)

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

4.8.11 int HYPRE_StructHybridSetSolverType (HYPRE_StructSolver solver, 
    int solver_type)

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

4.8.12 int HYPRE_StructHybridSetKDim (HYPRE_StructSolver solver, 
    int k_dim)

    (Optional) Set the maximum size of the Krylov space when using GMRES
4.8.14 int HYPRE_StructHybridSetLogging (HYPRE_StructSolver solver, int logging)

(Optional) Set the amount of logging to do .................................. 91

4.8.15 int HYPRE_StructHybridSetPrintLevel (HYPRE_StructSolver solver, int print_level)

(Optional) Set the amount of printing to do to the screen ................. 91

4.8.16 int HYPRE_StructHybridGetNumIterations (HYPRE_StructSolver solver, int* num_its)

Return the number of iterations taken ........................................ 91

4.8.17 int HYPRE_StructHybridGetDSCGNumIterations (HYPRE_StructSolver solver, int* ds_num_its)

Return the number of diagonal scaling iterations taken .................... 91

4.8.18 int HYPRE_StructHybridGetPCGNumIterations (HYPRE_StructSolver solver, int* pre_num_its)

Return the number of general preconditioning iterations taken ............. 92

4.8.19 int HYPRE_StructHybridGetFinalRelativeResidualNorm (HYPRE_StructSolver solver, double* norm)

Return the norm of the final relative residual ................................ 92

4.8.1 int HYPRE_StructHybridCreate (MPI_Comm comm, HYPRE_StructSolver* solver)

Create a solver object
4.8.2

```c
int HYPRE_StructHybridDestroy (HYPRE_StructSolver solver)
```

Destroy a solver object

4.8.3

```c
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.

4.8.4

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

Solve the system

4.8.5

```c
int HYPRE_StructHybridSetTol (HYPRE_StructSolver solver, double tol)
```

(Optional) Set the convergence tolerance
4.8.6  

int  
HYPRE_StructHybridSetConvergenceTol (HYPRE_StructSolver solver,  
double 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 $cf\_tol$.

4.8.7  

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 $ds\_max\_its$.

4.8.8  

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 $pre\_max\_its$.

4.8.9  

int  
HYPRE_StructHybridSetTwoNorm (HYPRE_StructSolver solver, int  
two_norm)

(Optional) Use the two-norm in stopping criteria.
4.8.10

int HYPRE_StructHybridSetRelChange (HYPRE_StructSolver solver, int rel_change)

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

4.8.11

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

4.8.12

int HYPRE_StructHybridSetKDim (HYPRE_StructSolver solver, int k_dim)

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

4.8.13

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

(Optional) Set the preconditioner to use
### 4.8.14

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

(Optional) Set the amount of logging to do

### 4.8.15

**HYPRE_StructHybridSetPrintLevel** (HYPRE_StructSolver solver, int print_level)

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

### 4.8.16

**HYPRE_StructHybridGetNumIterations** (HYPRE_StructSolver solver, int* num_its)

Return the number of iterations taken

### 4.8.17

**HYPRE_StructHybridGetDSCGNumIterations** (HYPRE_StructSolver solver, int* ds_num_its)

Return the number of diagonal scaling iterations taken
4.8.18

```
int HYPRE_StructHybridGetPCGNumIterations (HYPRE_StructSolver solver,
int* pre_num_its)
```

Return the number of general preconditioning iterations taken

4.8.19

```
int HYPRE_StructHybridGetFinalRelativeResidualNorm
(HYPRE_StructSolver solver, double* norm)
```

Return the norm of the final relative residual
SStruct Solvers

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

5.1.1 typedef struct hypre_SStructSolver *HYPRE_SStructSolver
The solver object

5.2

SStruct PCG Solver

Names

5.2.1

int HYPRE_SStructPCGCreate (MPI_Comm comm,
   HYPRE_SStructSolver* solver)

Create a solver object

5.2.2

int HYPRE_SStructPCGDestroy (HYPRE_SStructSolver solver)

Destroy a solver object.

5.2.3

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

Prepare to solve the system.

5.2.4

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

Solve the system

5.2.5

int HYPRE_SStructPCGSetTol (HYPRE_SStructSolver solver, double tol)

(Optional) Set the convergence tolerance

5.2.6

int HYPRE_SStructPCGSetMaxIter (HYPRE_SStructSolver solver,
   int max_iter)

(Optional) Set maximum number of iterations

5.2.7

int HYPRE_SStructPCGSetTwoNorm ( HYPRE_SStructSolver solver,
   int two_norm )

(Optional) Use the two-norm in stopping criteria

5.2.8

int HYPRE_SStructPCGSetRelChange ( HYPRE_SStructSolver solver,
   int rel_change )

(Optional) Additionally require that the relative difference in successive it-
erates be small
5.2.10 `int HYPRE_SStructPCGSetLogging (HYPRE_SStructSolver solver, int logging)`

(Optional) Set the amount of logging to do .............................................. 101

5.2.11 `int HYPRE_SStructPCGSetPrintLevel (HYPRE_SStructSolver solver, int level)`

(Optional) Set the amount of printing to do to the screen .......................... 101

5.2.12 `int HYPRE_SStructPCGGetNumIterations (HYPRE_SStructSolver solver, int* num_iterations)`

Return the number of iterations taken ..................................................... 101

5.2.13 `int HYPRE_SStructPCGGetFinalRelativeResidualNorm (HYPRE_SStructSolver solver, double* norm)`

Return the norm of the final relative residual ........................................... 101

5.2.14 `int HYPRE_SStructPCGGetResidual (HYPRE_SStructSolver solver, void** residual)`

Return the residual ...................................................................................... 102

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

Setup routine for diagonal preconditioning ................................................. 102

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

Solve routine for diagonal preconditioning ............................................... 102

5.2.1

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

5.2.2  

\[
\text{int } \text{HYPRE\_SStructPCGDestroy} \ (\text{HYPRE\_SStructSolver } \text{solver})
\]

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  

\[
\text{int } \text{HYPRE\_SStructPCGSetup} \ (\text{HYPRE\_SStructSolver } \text{solver}, \\
\text{HYPRE\_SStructMatrix } \text{A}, \text{HYPRE\_SStructVector } \text{b}, \text{HYPRE\_SStructVector } \text{x})
\]

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  

\[
\text{int } \text{HYPRE\_SStructPCGSolve} \ (\text{HYPRE\_SStructSolver } \text{solver}, \\
\text{HYPRE\_SStructMatrix } \text{A}, \text{HYPRE\_SStructVector } \text{b}, \text{HYPRE\_SStructVector } \text{x})
\]

Solve the system

5.2.5  

\[
\text{int } \text{HYPRE\_SStructPCGSetTol} \ (\text{HYPRE\_SStructSolver } \text{solver}, \text{double } \text{tol})
\]

(Optional) Set the convergence tolerance
5.2.6

\begin{verbatim}
int HYPRE_SStructPCGSetMaxIter ( HYPRE_SStructSolver solver, int max_iter )
\end{verbatim}

(Optional) Set maximum number of iterations

5.2.7

\begin{verbatim}
int HYPRE_SStructPCGSetTwoNorm ( HYPRE_SStructSolver solver, int two_norm )
\end{verbatim}

(Optional) Use the two-norm in stopping criteria

5.2.8

\begin{verbatim}
int HYPRE_SStructPCGSetRelChange ( HYPRE_SStructSolver solver, int rel_change )
\end{verbatim}

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

5.2.9

\begin{verbatim}
int HYPRE_SStructPCGSetPrecond ( HYPRE_SStructSolver solver, 
      HYPRE_PtrToSStructSolverFcn precond, HYPRE_PtrToSStructSolverFcn
      precond_setup, void* precond_solver)
\end{verbatim}

(Optional) Set the preconditioner to use
5.2.10

int HYPRE_SStructPCGSetLogging (HYPRE_SStructSolver solver, int logging)

(Optional) Set the amount of logging to do

5.2.11

int HYPRE_SStructPCGSetPrintLevel (HYPRE_SStructSolver solver, int level)

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

5.2.12

int HYPRE_SStructPCGGetNumIterations (HYPRE_SStructSolver solver, int* num_iterations)

Return the number of iterations taken

5.2.13

int HYPRE_SStructPCGGetFinalRelativeResidualNorm (HYPRE_SStructSolver solver, double* norm)

Return the norm of the final relative residual
5.2.14

```c
int HYPRE_SStructPCGGetResidual (HYPRE_SStructSolver solver, void** residual)
```

Return the residual

5.2.15

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

Setup routine for diagonal preconditioning

5.2.16

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

Solve routine for diagonal preconditioning

5.3

**SStruct GMRES Solver**

Names

5.3.1 int

```c
HYPRE_SStructGMRESCreate (MPI_Comm comm,
                          HYPRE_SStructSolver* solver)
```

Create a solver object

5.3.2 int

```c
HYPRE_SStructGMRESDestroy (HYPRE_SStructSolver solver)
```

Destroy a solver object.
5.3.4 int HYPRE_SStructGMRESSetup (HYPRE_SStructSolver solver, HYPRE_SStructMatrix A, HYPRE_SStructVector b, HYPRE_SStructVector x)

Prepare to solve the system.

5.3.5 int HYPRE_SStructGMRESSolve (HYPRE_SStructSolver solver, HYPRE_SStructMatrix A, HYPRE_SStructVector b, HYPRE_SStructVector x)

Solve the system

5.3.6 int HYPRE_SStructGMRESSetTol (HYPRE_SStructSolver solver, double tol)

(Optional) Set the convergence tolerance

5.3.7 int HYPRE_SStructGMRESSetMaxIter (HYPRE_SStructSolver solver, int max_iter)

(Optional) Set maximum number of iterations

5.3.8 int HYPRE_SStructGMRESSetKDim (HYPRE_SStructSolver solver, int k_dim)

(Optional) Set the maximum size of the Krylov space

5.3.9 int HYPRE_SStructGMRESSetPrecond (HYPRE_SStructSolver solver, HYPRE_PtrToSStructSolverFcn precond, HYPRE_PtrToSStructSolverFcn precond_setup, void* precond_solver)

(Optional) Set the preconditioner to use

5.3.10 int HYPRE_SStructGMRESSetLogging (HYPRE_SStructSolver solver, int logging)

(Optional) Set the amount of logging to do

5.3.11 int HYPRE_SStructGMRESSetPrintLevel (HYPRE_SStructSolver solver, int print_level)

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

5.3.12 int HYPRE_SStructGMRESSetFinalRelativeResidualNorm (HYPRE_SStructSolver solver, double* norm)

Return the number of iterations taken

5.3.13 int HYPRE_SStructGMRESGetFinalRelativeResidualNorm (HYPRE_SStructSolver solver, double* norm)

Return the norm of the final relative residual
5.3.1

\textbf{int HYPRE_SStructGMRESCreate} (MPI_Comm comm, HYPRE_SStructSolver* solver)

Create a solver object

5.3.2

\textbf{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.

5.3.3

\textbf{int HYPRE_SStructGMRESSetup} (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

```c
int HYPRE_SStructGMRESSolve (HYPRE_SStructSolver solver, HYPRE_SStructMatrix A, HYPRE_SStructVector b, HYPRE_SStructVector x)
```

Solve the system

5.3.5

```c
int HYPRE_SStructGMRESSetTol (HYPRE_SStructSolver solver, double tol)
```

(Optional) Set the convergence tolerance

5.3.6

```c
int HYPRE_SStructGMRESSetMaxIter (HYPRE_SStructSolver solver, int max_iter)
```

(Optional) Set maximum number of iterations

5.3.7

```c
int HYPRE_SStructGMRESSetKDim (HYPRE_SStructSolver solver, int k_dim)
```

(Optional) Set the maximum size of the Krylov space
5.3.8

```c
int HYPRE_SStructGMRESSetPrecond (HYPRE_SStructSolver solver,
                               HYPRE_PtrToSStructSolverFcn precond,
                               HYPRE_PtrToSStructSolverFcn precond_setup,
                               void* precond_solver)
```

(Optional) Set the preconditioner to use

5.3.9

```c
int HYPRE_SStructGMRESSetLogging (HYPRE_SStructSolver solver, int logging)
```

(Optional) Set the amount of logging to do

5.3.10

```c
int HYPRE_SStructGMRESSetPrintLevel (HYPRE_SStructSolver solver, int print_level)
```

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

5.3.11

```c
int HYPRE_SStructGMRESGetNumIterations (HYPRE_SStructSolver solver,
                                         int* num_iterations)
```

Return the number of iterations taken
5.3.12

int HYPRE_SStructGMRESGetFinalRelativeResidualNorm (HYPRE_SStructSolver solver, double* norm)

Return the norm of the final relative residual

5.3.13

int HYPRE_SStructGMRESGetResidual (HYPRE_SStructSolver solver, void** residual)

Return the residual

5.4

SStruct BiCGSTAB Solver

Names

5.4.1 int HYPRE_SStructBiCGSTABCreate (MPI_Comm comm, HYPRE_SStructSolver* solver)

Create a solver object

5.4.2 int HYPRE_SStructBiCGSTABDestroy (HYPRE_SStructSolver solver)

Destroy a solver object.

5.4.3 int HYPRE_SStructBiCGSTABSetup (HYPRE_SStructSolver solver, HYPRE_SStructMatrix A, HYPRE_SStructVector b, HYPRE_SStructVector x)

Prepare to solve the system.

5.4.4 int HYPRE_SStructBiCGSTABSolve (HYPRE_SStructSolver solver, HYPRE_SStructMatrix A, HYPRE_SStructVector b, HYPRE_SStructVector x)

Solve the system
SStruct Solvers

5.4.6 int HYPRE_SStructBiCGSTABSetMaxIter (HYPRE_SStructSolver solver, int max_iter)

(Optional) Set the maximum number of iterations

5.4.7 int HYPRE_SStructBiCGSTABSetPrecond (HYPRE_SStructSolver solver, HYPRE_PtrToSStructSolverFcn precond, HYPRE_PtrToSStructSolverFcn precond_setup, void* precond_solver)

(Optional) Set the preconditioner to use

5.4.8 int HYPRE_SStructBiCGSTABSetLogging (HYPRE_SStructSolver solver, int logging)

(Optional) Set the amount of logging to do

5.4.9 int HYPRE_SStructBiCGSTABSetPrintLevel (HYPRE_SStructSolver solver, int level)

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

5.4.10 int HYPRE_SStructBiCGSTABGetNumIterations (HYPRE_SStructSolver solver, int* num_iterations)

Return the number of iterations taken

5.4.11 int HYPRE_SStructBiCGSTABGetFinalRelativeResidualNorm (HYPRE_SStructSolver solver, double* norm)

Return the norm of the final relative residual

5.4.12 int HYPRE_SStructBiCGSTABGetResidual (HYPRE_SStructSolver solver, void** residual)

Return the residual
5.4.1

```c
int HYPRE_SStructBiCGSTABCreate (MPI_Comm comm,
                                    HYPRE_SStructSolver* solver)
```

Create a solver object

5.4.2

```c
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.

5.4.3

```c
int HYPRE_SStructBiCGSTABSetup (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.

5.4.4

```c
int HYPRE_SStructBiCGSTABSolve (HYPRE_SStructSolver solver,
                                 HYPRE_SStructMatrix A, HYPRE_SStructVector b, HYPRE_SStructVector x)
```

Solve the system
5.4.5

\begin{verbatim}
int HYPRE_SStructBiCGSTABSetTol (HYPRE_SStructSolver solver, double tol)
\end{verbatim}

(Optional) Set the convergence tolerance

5.4.6

\begin{verbatim}
int HYPRE_SStructBiCGSTABSetMaxIter (HYPRE_SStructSolver solver, int max_iter)
\end{verbatim}

(Optional) Set maximum number of iterations

5.4.7

\begin{verbatim}
int HYPRE_SStructBiCGSTABSetPrecond (HYPRE_SStructSolver solver, HYPRE_PtrToSStructSolverFcn precond, HYPRE_PtrToSStructSolverFcn precond_setup, void* precond_solver)
\end{verbatim}

(Optional) Set the preconditioner to use

5.4.8

\begin{verbatim}
int HYPRE_SStructBiCGSTABSetLogging (HYPRE_SStructSolver solver, int logging)
\end{verbatim}

(Optional) Set the amount of logging to do
### 5.4.9

```c
int HYPRE_SStructBiCGSTABSetPrintLevel (HYPRE_SStructSolver solver, int level)
```

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

### 5.4.10

```c
int HYPRE_SStructBiCGSTABGetNumIterations (HYPRE_SStructSolver solver, int* num_iterations)
```

Return the number of iterations taken

### 5.4.11

```c
int HYPRE_SStructBiCGSTABGetFinalRelativeResidualNorm (HYPRE_SStructSolver solver, double* norm)
```

Return the norm of the final relative residual

### 5.4.12

```c
int HYPRE_SStructBiCGSTABGetResidual (HYPRE_SStructSolver solver, void** residual)
```

Return the residual
5.5 SStruct SysPFMG Solver

Names

5.5.1 int HYPRE_SStructSysPFMGCreate ( MPI_Comm comm, HYPRE_SStructSolver* solver )

Create a solver object ........................................... 113

5.5.2 int HYPRE_SStructSysPFMGDestroy (HYPRE_SStructSolver solver)

Destroy a solver object. ........................................ 114

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

Prepare to solve the system. ................................. 115

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

Solve the system ................................................... 115

5.5.5 int HYPRE_SStructSysPFMGSetTol (HYPRE_SStructSolver solver, double tol)

(Optional) Set the convergence tolerance ..................... 115

5.5.6 int HYPRE_SStructSysPFMGSetMaxIter (HYPRE_SStructSolver solver, int max_iter)

(Optional) Set maximum number of iterations .................. 116

5.5.7 int HYPRE_SStructSysPFMGSetRelChange (HYPRE_SStructSolver solver, int rel_change)

(Optional) Additionally require that the relative difference in successive iterates be small ........................................... 117

5.5.8 int HYPRE_SStructSysPFMGSetZeroGuess (HYPRE_SStructSolver solver)

(Optional) Use a zero initial guess. ............................ 117

5.5.9 int HYPRE_SStructSysPFMGSetNonZeroGuess (HYPRE_SStructSolver solver, int relax_type)

(Optional) Use a nonzero initial guess. .......................... 117

5.5.10 int HYPRE_SStructSysPFMGSetRelaxType (HYPRE_SStructSolver solver, int relax_type)

(Optional) Set relaxation type. .................................... 118

5.5.11 int
5.5.12 int HYPRE_SStructSysPFMGSetNumPreRelax (HYPRE_SStructSolver solver, int num_pre_relax)

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

5.5.13 int HYPRE_SStructSysPFMGSetNumPostRelax (HYPRE_SStructSolver solver, int num_post_relax)

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

5.5.14 int HYPRE_SStructSysPFMGSetSkipRelax (HYPRE_SStructSolver solver, int skip_relax)

(Optional) Skip relaxation on certain grids for isotropic problems.

5.5.15 int HYPRE_SStructSysPFMGSetLogging (HYPRE_SStructSolver solver, int logging)

(Optional) Set the amount of logging to do.

5.5.16 int HYPRE_SStructSysPFMGGetNumIterations (HYPRE_SStructSolver solver, int* num_iterations)

Return the number of iterations taken.

5.5.17 int HYPRE_SStructSysPFMGGetFinalRelativeResidualNorm (HYPRE_SStructSolver solver, double* norm)

Return the norm of the final relative residual.

5.5.1

int HYPRE_SStructSysPFMGCreate ( MPI_Comm comm, HYPRE_SStructSolver* solver )

Create a solver object
5.5.2  

```c
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.

5.5.3  

```c
int HYPRE_SStructSysPFMGSsetup (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  

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

Solve the system

5.5.5  

```c
int HYPRE_SStructSysPFMGSetTol (HYPRE_SStructSolver solver, double tol)
```

(Optional) Set the convergence tolerance
5.5.6

```c
int HYPRE_SStructSysPFMGSetMaxIter (HYPRE_SStructSolver solver, int max_iter)
```

(Optional) Set maximum number of iterations

5.5.7

```c
int HYPRE_SStructSysPFMGSetRelChange (HYPRE_SStructSolver solver, int rel_change)
```

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

5.5.8

```c
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 computational cost.

5.5.9

```c
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`. 
5.5.10

```c
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)

5.5.11

```c
int HYPRE_SStructSysPFMGSetNumPreRelax (HYPRE_SStructSolver solver, int num_pre_relax)
```

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

5.5.12

```c
int HYPRE_SStructSysPFMGSetNumPostRelax (HYPRE_SStructSolver solver, int num_post_relax)
```

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

5.5.13

```c
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.

5.5.14

```c
int HYPRE_SStructSysPFMGSetLogging (HYPRE_SStructSolver solver, int logging)
```

(Optional) Set the amount of logging to do

5.5.15

```c
int HYPRE_SStructSysPFMGSetPrintLevel (HYPRE_SStructSolver solver, int print_level)
```

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

5.5.16

```c
int HYPRE_SStructSysPFMGGetNumIterations (HYPRE_SStructSolver solver, int* num_iterations)
```

Return the number of iterations taken

5.5.17

```c
int HYPRE_SStructSysPFMGGetFinalRelativeResidualNorm (HYPRE_SStructSolver solver, double* norm)
```

Return the norm of the final relative residual
5.6 SStruct Split Solver

Names

5.6.1 int HYPRE_SStructSplitCreate (MPI_Comm comm, HYPRE_SStructSolver* solver)

Create a solver object ................................. [119]

5.6.2 int HYPRE_SStructSplitDestroy (HYPRE_SStructSolver solver)

Destroy a solver object. ................................. [119]

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

Prepare to solve the system. ............................. [119]

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

Solve the system ........................................... [120]

5.6.5 int HYPRE_SStructSplitSetTol (HYPRE_SStructSolver solver, double tol)

(Optional) Set the convergence tolerance .................. [120]

5.6.6 int HYPRE_SStructSplitSetMaxIter (HYPRE_SStructSolver solver,
int max_iter)

(Optional) Set maximum number of iterations .............. [120]

5.6.7 int HYPRE_SStructSplitSetZeroGuess (HYPRE_SStructSolver solver)

(Optional) Use a zero initial guess. .................... [121]

5.6.8 int HYPRE_SStructSplitSetNonZeroGuess (HYPRE_SStructSolver solver)

(Optional) Use a nonzero initial guess. .................. [121]

5.6.9 int HYPRE_SStructSplitSetStructSolver (HYPRE_SStructSolver solver,
int ssolver)

(Optional) Set up the type of diagonal struct solver. .. [121]

5.6.10 int HYPRE_SStructSplitGetNumIterations (HYPRE_SStructSolver solver,
int* num_iterations)

Return the number of iterations taken ........................ [121]

5.6.11 int

This page has been automatically generated with DOC++

DOC++ is ©1995 by Roland Wunderling
Malte Zöckler
**HYPRE_SStructSplitGetFinalRelativeResidualNorm**

(HYPRE_SStructSolver solver, double* norm)

*Return the norm of the final relative residual*

---

### 5.6.1

```c
int HYPRE_SStructSplitCreate (MPI_Comm comm, HYPRE_SStructSolver* solver)
```

Create a solver object.

### 5.6.2

```c
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.

### 5.6.3

```c
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.
5.6.4

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

Solve the system

5.6.5

int HYPRE_SStructSplitSetTol (HYPRE_SStructSolver solver, double tol)

(Optional) Set the convergence tolerance

5.6.6

int HYPRE_SStructSplitSetMaxIter (HYPRE_SStructSolver solver, int max_iter)

(Optional) Set maximum number of iterations

5.6.7

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 computational cost.
5.6.8

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.

5.6.9

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.

5.6.10

int HYPRE_SStructSplitGetNumIterations (HYPRE_SStructSolver solver, int* num_iterations)

Return the number of iterations taken

5.6.11

int HYPRE_SStructSplitGetFinalRelativeResidualNorm (HYPRE_SStructSolver solver, double* norm)

Return the norm of the final relative residual
5.7

SStruct FAC Solver

Names

5.7.1  int  HYPRE_SStructFACCreate ( MPI_Comm comm,
                                      HYPRE_SStructSolver* solver )
                                    
Create a solver object  .................................................. [124]

5.7.2  int  HYPRE_SStructFACDestroy2 ( HYPRE_SStructSolver solver )

Destroy a solver object. .................................................. [124]

5.7.3  int  HYPRE_SStructFACAMR_RAP ( HYPRE_SStructMatrix A,
                                      int (*rfactors)[3],
                                      HYPRE_SStructMatrix* fac_A )

Re-distribute the composite matrix so that the amr hierarchy is approximately
nested. ................................................................. [125]

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

Set up the FAC solver structure  ........................................ [125]

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

Solve the system  ........................................................... [125]

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

Set up amr structure  ..................................................... [125]

5.7.7  int  HYPRE_SStructFACSetPRefinements (HYPRE_SStructSolver solver,
                                           int nparts,  int (*rfactors)[3] )

Set up amr refinement factors  ......................................... [126]

5.7.8  int  HYPRE_SStructFACZeroCFSten (HYPRE_SStructMatrix A,
                                        HYPRE_SStructGrid grid,  int part,
                                        int rfactors[3])

(Optional, but user must make sure that they do this function otherwise)
Zero off the coarse level stencils reaching into a fine level grid  ....... [126]

5.7.9  int
5.7.10 int HYPRE_SStructFACZeroAMRMatrixData (HYPRE_SStructMatrix A, int part, crse, int rfactors[3])
(Optional, but user must make sure that they do this function otherwise)
Places the identity in the coarse grid matrix underlying the fine patches. ........ 126

5.7.11 int HYPRE_SStructFACZeroAMRVectorData (HYPRE_SStructVector b, int* plevels, int (*rfactors)[3])
(Optional, but user must make sure that they do this function otherwise)
Places zeros in the coarse grid vector underlying the fine patches. ........ 127

5.7.12 int HYPRE_SStructFACSetMaxLevels (HYPRE_SStructSolver solver, int max_levels)
(Optional) Set maximum number of FAC levels .......................... 127

5.7.13 int HYPRE_SStructFACSetTol (HYPRE_SStructSolver solver, double tol)
(Optional) Set the convergence tolerance ............................... 127

5.7.14 int HYPRE_SStructFACSetMaxIter (HYPRE_SStructSolver solver, int max_iter)
(Optional) Set maximum number of iterations .......................... 127

5.7.15 int HYPRE_SStructFACSetRelChange (HYPRE_SStructSolver solver, int rel_change)
(Optional) Additionally require that the relative difference in successive iterates be small ........................................... 128

5.7.16 int HYPRE_SStructFACSetZeroGuess (HYPRE_SStructSolver solver)
(Optional) Use a zero initial guess. ................................. 128

5.7.17 int HYPRE_SStructFACSetNonZeroGuess (HYPRE_SStructSolver solver)
(Optional) Use a nonzero initial guess. ............................ 128

5.7.18 int HYPRE_SStructFACSetRelaxType (HYPRE_SStructSolver solver, int relax_type)
(Optional) Set relaxation type. ...................................... 128

5.7.19 int HYPRE_SStructFACSetNumPreRelax (HYPRE_SStructSolver solver, int num_pre_relax)
(Optional) Set number of relaxation sweeps before coarse-grid correction . 129

5.7.20 int HYPRE_SStructFACSetNumPostRelax (HYPRE_SStructSolver solver, int num_post_relax)
(Optional) Set number of relaxation sweeps after coarse-grid correction . 129
**5.7.1**

**int HYPRE_SStructFACCreate** ( MPI_Comm comm, HYPRE_SStructSolver* solver )

Create a solver object

**5.7.2**

**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.
5.7.3

```c
int HYPRE_SStructFACAMR_RAP ( HYPRE_SStructMatrix A, int (*rfactors)[3],
                                HYPRE_SStructMatrix* fac_A )
```

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

5.7.4

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

Set up the FAC solver structure

5.7.5

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

Solve the system

5.7.6

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

Set up amr structure
5.7.7

```c
int HYPRE_SStructFACSetPRefinements (HYPRE_SStructSolver solver, int nparts, int (*rfactors)[3])
```

Set up amr refinement factors

5.7.8

```c
int HYPRE_SStructFACZeroCFSten (HYPRE_SStructMatrix A,
                                HYPRE_SStructGrid grid, int part, int rfactors[3])
```

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

5.7.9

```c
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

5.7.10

```c
int HYPRE_SStructFACZeroAMRMatrixData (HYPRE_SStructMatrix A, int part_crse, int rfactors[3])
```

(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.
5.7.11

```c
int HYPRE_SStructFACZeroAMRVectorData (HYPRE_SStructVector b, int* plevels, int (*rfactors)[3] )
```

(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.

5.7.12

```c
int HYPRE_SStructFACSetMaxLevels ( HYPRE_SStructSolver solver, int max_levels )
```

(Optional) Set maximum number of FAC levels

5.7.13

```c
int HYPRE_SStructFACSetTol (HYPRE_SStructSolver solver, double tol)
```

(Optional) Set the convergence tolerance

5.7.14

```c
int HYPRE_SStructFACSetMaxIter (HYPRE_SStructSolver solver, int max_iter)
```

(Optional) Set maximum number of iterations
5.7.15

int HYPRE_SStructFACSetRelChange (HYPRE_SStructSolver solver, int rel_change)

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

5.7.16

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 computational cost.

5.7.17

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.

5.7.18

int HYPRE_SStructFACSetRelaxType (HYPRE_SStructSolver solver, int relax_type)

(Optional) Set relaxation type. See HYPRE_SStructSysPFMGSetRelaxType for appropriate values of relax_type.
5.7.19

```c
int HYPRE_SStructFACSetNumPreRelax (HYPRE_SStructSolver solver, int num_pre_relax)
```

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

5.7.20

```c
int HYPRE_SStructFACSetNumPostRelax (HYPRE_SStructSolver solver, int num_post_relax)
```

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

5.7.21

```c
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

5.7.22

```c
int HYPRE_SStructFACSetLogging (HYPRE_SStructSolver solver, int logging)
```

(Optional) Set the amount of logging to do
5.7.23

```c
int HYPRE_SStructFACGetNumIterations (HYPRE_SStructSolver solver, int* num_iterations)
```

Return the number of iterations taken

5.7.24

```c
int HYPRE_SStructFACGetFinalRelativeResidualNorm (HYPRE_SStructSolver solver, double* norm)
```

Return the norm of the final relative residual

5.8

SStruct Maxwell Solver

Names

5.8.1 int

```c
HYPRE_SStructMaxwellCreate ( MPI_Comm comm,
    HYPRE_SStructSolver* solver )
```

Create a solver object

5.8.2 int

```c
HYPRE_SStructMaxwellDestroy ( HYPRE_SStructSolver solver )
```

Destroy a solver object.

5.8.3 int

```c
HYPRE_SStructMaxwellSetup (HYPRE_SStructSolver solver,
    HYPRE_SStructMatrix A,
    HYPRE_SStructVector b,
    HYPRE_SStructVector x)
```

Prepare to solve the system.

5.8.4 int

```c
HYPRE_SStructMaxwellSolve (HYPRE_SStructSolver solver,
    HYPRE_SStructMatrix A,
    HYPRE_SStructVector b,
    HYPRE_SStructVector x)
```

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

Solve the system. ............................................................... [133]

5.8.6
int
HYPRE_SStructMaxwellSetGrad (HYPRE_SStructSolver solver,
    HYPRE_ParCSRMatrix T)

Sets the gradient operator in the Maxwell solver ................... [133]

5.8.7
int
HYPRE_SStructMaxwellSetRfactors (HYPRE_SStructSolver solver,
    int rfactors[3])

Sets the coarsening factor .................................................. [133]

5.8.8
int
HYPRE_SStructMaxwellPhysBdy (HYPRE_SStructGrid* grid,
    int num_levels, int rfactors[3],
    int** BdryRanks_ptr,
    int** BdryRanksCnt_ptr)

Finds the physical boundary row ranks on all levels ................. [134]

5.8.9
int
HYPRE_SStructMaxwellEliminateRowsCols (HYPRE_ParCSRMatrix
    parA, int nrows, int* rows)

Eliminates the rows and cols corresponding to the physical boundary in a
parcsr matrix ................................................................. [134]

5.8.10
int
HYPRE_SStructMaxwellZeroVector (HYPRE_ParVector b, int* rows,
    int nrows)

Zeros the rows corresponding to the physical boundary in a par vector .... [134]

5.8.11
int
HYPRE_SStructMaxwellSetSetConstantCoef (HYPRE_SStructSolver
    solver, int flag)

(Optional) Set the constant coefficient flag- Nedelec interpolation used ... [135]

5.8.12
int
HYPRE_SStructMaxwellGrad (HYPRE_SStructGrid grid,
    HYPRE_ParCSRMatrix* T)

(Optional) Creates a gradient matrix from the grid. ................. [135]

5.8.13
int
HYPRE_SStructMaxwellSetTol (HYPRE_SStructSolver solver, double tol)

(Optional) Set the convergence tolerance ............................. [135]

5.8.14
int
HYPRE_SStructMaxwellSetMaxIter (HYPRE_SStructSolver solver,
    int max_iter)

(Optional) Set maximum number of iterations ......................... [135]

5.8.15
int
HYPRE_SStructMaxwellSetRelChange (HYPRE_SStructSolver solver,
    int rel_change)

(Optional) Additionally require that the relative difference in successive it-
erates be small ......................................................... [136]

5.8.16
int
5.8.17 int 
**HYPRE_SStructMaxwellSetNumPreRelax** (HYPRE_SStructSolver solver, int num_pre_relax)

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

5.8.18 int 
**HYPRE_SStructMaxwellSetNumPostRelax** (HYPRE_SStructSolver solver, int num_post_relax)

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

5.8.19 int 
**HYPRE_SStructMaxwellSetLogging** (HYPRE_SStructSolver solver, int logging)

*(Optional)* Set the amount of logging to do

5.8.20 int 
**HYPRE_SStructMaxwellGetNumIterations** (HYPRE_SStructSolver solver, int* num_iterations)

Return the number of iterations taken

5.8.21 int 
**HYPRE_SStructMaxwellGetFinalRelativeResidualNorm** (HYPRE_SStructSolver solver, double* norm)

Return the norm of the final relative residual

---

**5.8.1**

```
int HYPRE_SStructMaxwellCreate ( MPI_Comm comm, HYPRE_SStructSolver* solver )
```

Create a solver object

**5.8.2**

```
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.
5.8.3

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.

5.8.4

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.

5.8.5

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.

5.8.6

int

HYPRE_SStructMaxwellSetGrad (HYPRE_SStructSolver solver, HYPRE_ParCSRMatrix T)

Sets the gradient operator in the Maxwell solver
5.8.7

```c
int HYPRE_SStructMaxwellSetRfactors (HYPRE_SStructSolver solver, int rfactors[3])
```

Sets the coarsening factor

5.8.8

```c
int HYPRE_SStructMaxwellPhysBdy (HYPRE_SStructGrid* grid_l, int num_levels, int rfactors[3], int*** BdryRanks_ptr, int** BdryRanksCnt_ptr)
```

Finds the physical boundary row ranks on all levels

5.8.9

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

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

5.8.10

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

Zeros the rows corresponding to the physical boundary in a par vector
5.8.11

```
int HYPRE_SStructMaxwellSetSetConstantCoef (HYPRE_SStructSolver solver,
int flag)
```

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

5.8.12

```
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.

5.8.13

```
int HYPRE_SStructMaxwellSetTol (HYPRE_SStructSolver solver, double tol)
```

(Optional) Set the convergence tolerance

5.8.14

```
int HYPRE_SStructMaxwellSetMaxIter (HYPRE_SStructSolver solver, int max_iter)
```

(Optional) Set maximum number of iterations


5.8.15

int 
HYPRE_SStructMaxwellSetRelChange (HYPRE_SStructSolver solver, int rel_change)

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

5.8.16

int 
HYPRE_SStructMaxwellSetNumPreRelax (HYPRE_SStructSolver solver, int num_pre_relax)

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

5.8.17

int 
HYPRE_SStructMaxwellSetNumPostRelax (HYPRE_SStructSolver solver, int num_post_relax)

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

5.8.18

int 
HYPRE_SStructMaxwellSetLogging (HYPRE_SStructSolver solver, int logging)

(Optional) Set the amount of logging to do
Return the number of iterations taken

```c
int HYPRE_SStructMaxwellGetNumIterations (HYPRE_SStructSolver solver, int* num_iterations)
```

Return the norm of the final relative residual

```c
int HYPRE_SStructMaxwellGetFinalRelativeResidualNorm (HYPRE_SStructSolver solver, double* norm)
```
6 ParCSR Solvers

Names

6.1 ParCSR Solvers

6.2 ParCSR BoomerAMG Solver and Preconditioner

6.3 ParCSR ParaSails Preconditioner

6.4 ParCSR Euclid Preconditioner

6.5 ParCSR Pilut Preconditioner

6.6 ParCSR AMS Solver and Preconditioner

6.7 ParCSR Hybrid Solver

6.8 ParCSR PCG Solver

6.9 ParCSR GMRES Solver

6.10 ParCSR BiCGSTAB Solver

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

6.1 ParCSR Solvers

Names

6.1.1 #define HYPRE_SOLVER_STRUCT

The solver object
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 ........................................ 144

6.2.2 int HYPRE_BoomerAMGDestroy (HYPRE_Solver solver)

Destroy a solver object ........................................ 144

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. ............... 145

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. .......... 145

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 . ............................ 145

6.2.6 int HYPRE_BoomerAMGSetTol (HYPRE_Solver solver, double tol)

(Optional) Set the convergence tolerance, if BoomerAMG is used as a solver. ........................................ 146

6.2.7 int HYPRE_BoomerAMGSetMaxIter (HYPRE_Solver solver, int max_iter)

(Optional) Sets maximum number of iterations, if BoomerAMG is used as a solver. ................................. 146

6.2.8 int
**HYPRE_BoomerAMGSetMaxLevels** (HYPRE_Solver solver, int max_levels)

(Optional) Sets maximum number of multigrid levels.

---

**int**

**HYPRE_BoomerAMGSetStrongThreshold** (HYPRE_Solver solver, double strong_threshold)

(Optional) Sets AMG strength threshold.

---

**int**

**HYPRE_BoomerAMGSetMaxRowSum** (HYPRE_Solver solver, double max_row_sum)

(Optional) Sets a parameter to modify the definition of strength for diagonal dominant portions of the matrix.

---

**int**

**HYPRE_BoomerAMGSetCoarsenType** (HYPRE_Solver solver, int coarsen_type)

(Optional) Defines which parallel coarsening algorithm is used.

---

**int**

**HYPRE_BoomerAMGSetMeasureType** (HYPRE_Solver solver, int measure_type)

(Optional) Defines whether local or global measures are used.

---

**int**

**HYPRE_BoomerAMGSetCycleType** (HYPRE_Solver solver, int cycle_type)

(Optional) Defines the type of cycle.

---

**int**

**HYPRE_BoomerAMGSetNumGridSweeps** (HYPRE_Solver solver, int* num_grid_sweeps)

(Optional) Defines the number of sweeps for the fine and coarse grid, the up and down cycle.

---

**int**

**HYPRE_BoomerAMGSetNumSweeps** (HYPRE_Solver solver, int num_sweeps)

(Optional) Sets the number of sweeps.

---

**int**

**HYPRE_BoomerAMGSetCycleNumSweeps** (HYPRE_Solver solver, int num_sweeps, int k)

(Optional) Sets the number of sweeps at a specified cycle.

---

**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.

---

**int**

**HYPRE_BoomerAMGSetRelaxType** (HYPRE_Solver solver, int relax_type)

(Optional) Defines the smoother to be used.

---

**int**

**HYPRE_BoomerAMGSetCycleRelaxType** (HYPRE_Solver solver, int relax_type, int k)

(Optional) Defines the smoother at a given cycle.

---

**int**
HYPRE_BoomerAMGSetRelaxOrder (HYPRE_Solver solver,
    int relax_order)

    (Optional) Defines in which order the points are relaxed. .......... 150

6.2.21 int
HYPRE_BoomerAMGSetGridRelaxPoints (HYPRE_Solver solver,
    int** grid_relax_points)

    (Optional) Defines in which order the points are relaxed. .......... 150

6.2.22 int
HYPRE_BoomerAMGSetRelaxWeight (HYPRE_Solver solver,
    double* relax_weight)

    (Optional) Defines the relaxation weight for smoothed Jacobi and hybrid
    SOR. ................................................................. 151

6.2.23 int
HYPRE_BoomerAMGSetRelaxWt (HYPRE_Solver solver,
    double relax_weight)

    (Optional) Defines the relaxation weight for smoothed Jacobi and hybrid
    SOR on all levels. .................................................. 151

6.2.24 int
HYPRE_BoomerAMGSetLevelRelaxWt (HYPRE_Solver solver,
    double relax_weight, int level)

    (Optional) Defines the relaxation weight for smoothed Jacobi and hybrid
    SOR on the user defined level. .................................. 151

6.2.25 int
HYPRE_BoomerAMGSetOmega (HYPRE_Solver solver, double* omega)

    (Optional) Defines the outer relaxation weight for hybrid SOR. ....... 152

6.2.26 int
HYPRE_BoomerAMGSetOuterWt (HYPRE_Solver solver, double omega)

    (Optional) Defines the outer relaxation weight for hybrid SOR and SSOR
    on all levels. ....................................................... 152

6.2.27 int
HYPRE_BoomerAMGSetLevelOuterWt (HYPRE_Solver solver,
    double omega, int level)

    (Optional) Defines the outer relaxation weight for hybrid SOR or SSOR on
    the user defined level. ........................................... 152

6.2.28 int
HYPRE_BoomerAMGSetDebugFlag (HYPRE_Solver solver, int debug_flag)

    (Optional) .......................................................... 152

6.2.29 int
HYPRE_BoomerAMGGetResidual (HYPRE_Solver solver,
    HYPRE_ParVector* residual)

    Returns the residual ............................................... 153

6.2.30 int
HYPRE_BoomerAMGGetNumIterations (HYPRE_Solver solver,
    int* num_iterations)

    Returns the number of iterations taken .......................... 153

6.2.31 int
HYPRE_BoomerAMGGetFinalRelativeResidualNorm (HYPRE_Solver solver, double* rel_resid_norm)

Returns the norm of the final relative residual

6.2.32 int HYPRE_BoomerAMGSetTruncFactor (HYPRE_Solver solver, double trunc_factor)

(Optional) Defines a truncation factor for the interpolation.

6.2.33 int HYPRE_BoomerAMGSetPMaxElmts (HYPRE_Solver solver, int P_max_elmts)

(Optional) Defines the maximal number of elements per row for the interpolation.

6.2.34 int HYPRE_BoomerAMGSetSCommPkgSwitch (HYPRE_Solver solver, double S_commpkg_switch)

(Optional) Defines the largest strength threshold for which the strength matrix S uses the communication package of the operator A.

6.2.35 int HYPRE_BoomerAMGSetInterpType (HYPRE_Solver solver, int interp_type)

(Optional) Defines which parallel interpolation operator is used.

6.2.36 int HYPRE_BoomerAMGSetMinIter (HYPRE_Solver solver, int min_iter)

(Optional) Sets the number of min_iter.

6.2.37 int HYPRE_BoomerAMGInitGridRelaxation (int** num_grid_sweeps_ptr, int** grid_relax_type_ptr, int** grid_relax_points_ptr, int coarsen_type, double** relax_weights_ptr, int max_levels)

(Optional) This routine will be eliminated in the future.

6.2.38 int HYPRE_BoomerAMGSetSmoothType (HYPRE_Solver solver, int smooth_type)

(Optional) Enables the use of more complex smoothers.

6.2.39 int HYPRE_BoomerAMGSetSmoothNumLevels (HYPRE_Solver solver, int smooth_num_levels)

(Optional) Sets the number of levels for more complex smoothers.

6.2.40 int HYPRE_BoomerAMGSetSmoothNumSweeps (HYPRE_Solver solver, int smooth_num_sweeps)

(Optional) Sets the number of sweeps for more complex smoothers.

6.2.41 int HYPRE_BoomerAMGSetPrintLevel (HYPRE_Solver solver, int print_level)

(Optional) Requests automatic printing of setup and solve information.

6.2.42 int
**HYPRE_BoomerAMGSetLogging** (HYPRE_Solver solver, int logging)  
(Optional) Requests additional computations for diagnostic and similar data to be logged by the user. ............................................ 156

6.2.43

**int**  
**HYPRE_BoomerAMGSetNumFunctions** (HYPRE_Solver solver, int num_functions)  
(Optional) Sets the size of the system of PDEs, if using the systems version. ................................................................. 157

6.2.44

**int**  
**HYPRE_BoomerAMGSetNodal** (HYPRE_Solver solver, int nodal)  
(Optional) Sets whether to use the nodal systems version. .............................. 157

6.2.45

**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. ...................................................... 157

6.2.46

**int**  
**HYPRE_BoomerAMGSetAggNumLevels** (HYPRE_Solver solver, int agg_num_levels)  
(Optional) Defines the number of levels of aggressive coarsening. .................. 157

6.2.47

**int**  
**HYPRE_BoomerAMGSetNumPaths** (HYPRE_Solver solver, int num_paths)  
(Optional) Defines the degree of aggressive coarsening. ............................... 158

6.2.48

**int**  
**HYPRE_BoomerAMGSetVariant** (HYPRE_Solver solver, int variant)  
(Optional) Defines which variant of the Schwarz method is used. .................... 158

6.2.49

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

6.2.50

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

6.2.51

**int**  
**HYPRE_BoomerAMGSetSchwarzRlxWeight** (HYPRE_Solver solver, double schwarz_rlx_weight)  
(Optional) Defines a smoothing parameter for the additive Schwarz method  
......................................................................................... 159

6.2.52

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

6.2.53

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

6.2.54

**int**  
**HYPRE_BoomerAMGSetThreshold** (HYPRE_Solver solver, double threshold)  
(Optional) Defines threshold for ParaSAILS. ............................................ 159

6.2.55

**int**
ParCSR Solvers

6.2.1 \textbf{int HYPRE_BoomerAMGCreate} (HYPRE_Solver* solver)

Create a solver object

6.2.2 \textbf{int HYPRE_BoomerAMGDestroy} (HYPRE_Solver solver)

Destroy a solver object

Parallel unstructured algebraic multigrid solver and preconditioner

\textbf{HYPRE_BoomerAMGSetFilter} (HYPRE_Solver solver, double filter)

\textit{(Optional)} Defines filter for ParaSAILS.

\textbf{HYPRE_BoomerAMGSetDropTol} (HYPRE_Solver solver, double drop_tol)

\textit{(Optional)} Defines drop tolerance for PILUT.

\textbf{HYPRE_BoomerAMGSetMaxNzPerRow} (HYPRE_Solver solver, int max_nz_per_row)

\textit{(Optional)} Defines maximal number of nonzeros for PILUT.

\textbf{HYPRE_BoomerAMGSetEuclidFile} (HYPRE_Solver solver, char* euclidfile)

\textit{(Optional)} Defines name of an input file for Euclid parameters.

\textbf{HYPRE_BoomerAMGSetGSMG} (HYPRE_Solver solver, int gsmg)

\textit{(Optional)} Specifies the use of GSMG - geometrically smooth coarsening and interpolation.

\textbf{HYPRE_BoomerAMGSetNumSamples} (HYPRE_Solver solver, int num_samples)

\textit{(Optional)} Defines the number of sample vectors used in GSMG or LS interpolation.
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. 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

```c
int HYPRE_BoomerAMGSetTol(HYPRE_Solver solver, double tol)
```

(Optional) Set the convergence tolerance, if BoomerAMG is used as a solver. If it is used as a preconditioner, this function has no effect. The default is $1.e-7$.

### 6.2.7

```c
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, this function has no effect. The default is 20.

### 6.2.8

```c
int HYPRE_BoomerAMGSetMaxLevels(HYPRE_Solver solver, int max_levels)
```

(Optional) Sets maximum number of multigrid levels. The default is 25.

### 6.2.9

```c
int HYPRE_BoomerAMGSetStrongThreshold(HYPRE_Solver solver, double 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.

6.2.10

```c
int HYPRE_BoomerAMGSetMaxRowSum (HYPRE_Solver solver, double 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.

6.2.11

```c
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`:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>CLJP-coarsening (a parallel coarsening algorithm using independent sets)</td>
</tr>
<tr>
<td>1</td>
<td>classical Ruge-Stueben coarsening on each processor, no boundary treatment (not recommended!)</td>
</tr>
<tr>
<td>3</td>
<td>classical Ruge-Stueben coarsening on each processor, followed by a third pass, which adds coarse points on the boundaries</td>
</tr>
<tr>
<td>6</td>
<td>Falgout coarsening (uses 1 first, followed by CLJP using the interior coarse points generated by 1 as its first independent set)</td>
</tr>
<tr>
<td>7</td>
<td>CLJP-coarsening (using a fixed random vector, for debugging purposes only)</td>
</tr>
<tr>
<td>8</td>
<td>PMIS-coarsening (a parallel coarsening algorithm using independent sets, generating lower complexities than CLJP, might also lead to slower convergence)</td>
</tr>
<tr>
<td>9</td>
<td>PMIS-coarsening (using a fixed random vector, for debugging purposes only)</td>
</tr>
<tr>
<td>10</td>
<td>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)</td>
</tr>
<tr>
<td>11</td>
<td>one-pass Ruge-Stueben coarsening on each processor, no boundary treatment (not recommended!)</td>
</tr>
</tbody>
</table>

The default is 6.
6.2.12

```c
int HYPRE_BoomerAMGSetMeasureType (HYPRE_Solver solver, int measure_type)
```

(Optional) Defines whether local or global measures are used.

6.2.13

```c
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.

6.2.14

```c
int HYPRE_BoomerAMGSetNumGridSweeps (HYPRE_Solver solver, int* num_grid_sweeps)
```

(Optional) Defines the number of sweeps for the fine and coarse grid, the up and down cycle.

Note: This routine will be phased out!!!! Use HYPRE_BoomerAMGSetNumSweeps or HYPRE_BoomerAMGSetCycleNumSweeps instead.

6.2.15

```c
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.
6.2.16

```c
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:

<table>
<thead>
<tr>
<th>Option</th>
<th>k</th>
</tr>
</thead>
<tbody>
<tr>
<td>the finest level</td>
<td>0</td>
</tr>
<tr>
<td>the down cycle</td>
<td>1</td>
</tr>
<tr>
<td>the up cycle</td>
<td>2</td>
</tr>
<tr>
<td>the coarsest level</td>
<td>3</td>
</tr>
</tbody>
</table>

6.2.17

```c
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.

6.2.18

```c
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`:

<table>
<thead>
<tr>
<th>Option</th>
<th>Relax Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Jacobi</td>
</tr>
<tr>
<td>1</td>
<td>Gauss-Seidel, sequential (very slow!)</td>
</tr>
<tr>
<td>2</td>
<td>Gauss-Seidel, interior points in parallel, boundary sequential (slow!)</td>
</tr>
<tr>
<td>3</td>
<td>hybrid Gauss-Seidel or SOR, forward solve</td>
</tr>
<tr>
<td>4</td>
<td>hybrid Gauss-Seidel or SOR, backward solve</td>
</tr>
<tr>
<td>5</td>
<td>hybrid chaotic Gauss-Seidel (works only with OpenMP)</td>
</tr>
<tr>
<td>6</td>
<td>hybrid symmetric Gauss-Seidel or SSOR</td>
</tr>
<tr>
<td>9</td>
<td>Gaussian elimination (only on coarsest level)</td>
</tr>
</tbody>
</table>
6.2.19

```c
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

<table>
<thead>
<tr>
<th>Level</th>
<th>k</th>
</tr>
</thead>
<tbody>
<tr>
<td>the finest level</td>
<td>0</td>
</tr>
<tr>
<td>the down cycle</td>
<td>1</td>
</tr>
<tr>
<td>the up cycle</td>
<td>2</td>
</tr>
<tr>
<td>the coarsest level</td>
<td>3</td>
</tr>
</tbody>
</table>

6.2.20

```c
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:

<table>
<thead>
<tr>
<th>Order</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>the points are relaxed in natural or lexicographic order on each processor</td>
</tr>
<tr>
<td>1</td>
<td>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.</td>
</tr>
</tbody>
</table>

The default is 1 (CF-relaxation).

6.2.21

```c
int HYPRE_BoomerAMGSetGridRelaxPoints (HYPRE_Solver solver, int** grid_relax_points)
```

(Optional) Defines in which order the points are relaxed.

Note: This routine will be phased out!!!! Use HYPRE_BoomerAMGSetRelaxOrder instead.
6.2.22

```c
int HYPRE_BoomerAMGSetRelaxWeight (HYPRE_Solver solver, double* relax_weight)
```

(Optional) Defines the relaxation weight for smoothed Jacobi and hybrid SOR.

Note: This routine will be phased out!!!! Use HYPRE_BoomerAMGSetRelaxWt or HYPRE_BoomerAMGSetLevelRelaxWt instead.

6.2.23

```c
int HYPRE_BoomerAMGSetRelaxWt (HYPRE_Solver solver, double 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 \( \frac{3}{4} \| D^{-1/2} A D^{-1/2} \| \), 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.

6.2.24

```c
int HYPRE_BoomerAMGSetLevelRelaxWt (HYPRE_Solver solver, double 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.
6.2.25

int HYPRE_BoomerAMGSetOmega (HYPRE_Solver solver, double* omega)

(Optional) Defines the outer relaxation weight for hybrid SOR. Note: This routine will be phased out!!!! Use HYPRE_BoomerAMGSetOuterWt or HYPRE_BoomerAMGSetLevelOuterWt instead.

6.2.26

int HYPRE_BoomerAMGSetOuterWt (HYPRE_Solver solver, double 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.

6.2.27

int HYPRE_BoomerAMGSetLevelOuterWt (HYPRE_Solver solver, double 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.

6.2.28

int HYPRE_BoomerAMGSetDebugFlag (HYPRE_Solver solver, int debug_flag)

(Optional)
6.2.29

```
int HYPRE_BoomerAMGGetResidual (HYPRE_Solver solver,
        HYPRE_ParVector* residual)
```

Returns the residual

6.2.30

```
int HYPRE_BoomerAMGGetNumIterations (HYPRE_Solver solver, int*
        num_iterations)
```

Returns the number of iterations taken

6.2.31

```
int HYPRE_BoomerAMGGetFinalRelativeResidualNorm (HYPRE_Solver
        solver, double* rel_resid_norm)
```

Returns the norm of the final relative residual

6.2.32

```
int HYPRE_BoomerAMGSetTruncFactor (HYPRE_Solver solver, double
        trunc_factor)
```

(Optional) Defines a truncation factor for the interpolation. The default is 0.
6.2.33

\begin{align*}
\text{int} & \quad \text{HYPRE_BoomerAMGSetPMaxElmts} \ (\text{HYPRE_Solver} \ solver, \ \text{int} \ P_{\text{max\_elmts}}) \\
\end{align*}

(Optional) Defines the maximal number of elements per row for the interpolation. The default is 0.

6.2.34

\begin{align*}
\text{int} & \quad \text{HYPRE_BoomerAMGSetSCommPkgSwitch} \ (\text{HYPRE_Solver} \ solver, \ \text{double} \ S_{\text{commpkg\_switch}}) \\
\end{align*}

(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 value, 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.

6.2.35

\begin{align*}
\text{int} & \quad \text{HYPRE_BoomerAMGSetInterpType} \ (\text{HYPRE_Solver} \ solver, \ \text{int} \ \text{interp\_type}) \\
\end{align*}

(Optional) Defines which parallel interpolation operator is used. There are the following options for interp_type:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>classical modified interpolation</td>
</tr>
<tr>
<td>1</td>
<td>LS interpolation (for use with GSMG)</td>
</tr>
<tr>
<td>2</td>
<td>classical modified interpolation for hyperbolic PDEs</td>
</tr>
<tr>
<td>3</td>
<td>direct interpolation (with separation of weights)</td>
</tr>
<tr>
<td>4</td>
<td>multipass interpolation</td>
</tr>
<tr>
<td>5</td>
<td>multipass interpolation (with separation of weights)</td>
</tr>
<tr>
<td>6</td>
<td>extended classical modified interpolation</td>
</tr>
<tr>
<td>7</td>
<td>extended (if no common C neighbor) classical modified interpolation</td>
</tr>
<tr>
<td>8</td>
<td>standard interpolation</td>
</tr>
<tr>
<td>9</td>
<td>standard interpolation (with separation of weights)</td>
</tr>
<tr>
<td>10</td>
<td>classical block interpolation (for use with nodal systems version only)</td>
</tr>
<tr>
<td>11</td>
<td>classical block interpolation (for use with nodal systems version only) with diagonalized diagonal blocks</td>
</tr>
<tr>
<td>12</td>
<td>FF interpolation</td>
</tr>
<tr>
<td>13</td>
<td>FF1 interpolation</td>
</tr>
</tbody>
</table>
The default is 0.

### 6.2.36

```c
int HYPRE_BoomerAMGSetMinIter (HYPRE_Solver solver, int min_iter)
```

(Optional)

### 6.2.37

```c
int HYPRE_BoomerAMGInitGridRelaxation (int** num_grid_sweeps_ptr, int**
grid_relax_type_ptr, int*** grid_relax_points_ptr, int coarsen_type, double**
relax_weights_ptr, int max_levels)
```

(Optional) This routine will be eliminated in the future

### 6.2.38

```c
int HYPRE_BoomerAMGSetSmoothType (HYPRE_Solver solver, int
smooth_type)
```

(Optional) Enables the use of more complex smoothers. The following options exist for smooth_type:

<table>
<thead>
<tr>
<th>value</th>
<th>smoother</th>
<th>routines needed to set smoother parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>Schwarz smoother</td>
<td>HYPRE_BoomerAMGSetDomainType, HYPRE_BoomerAMGSetOverlap, HYPRE_BoomerAMGSetSchwarzRlxWeight</td>
</tr>
<tr>
<td>7</td>
<td>Pilut</td>
<td>HYPRE_BoomerAMGSetDropTol, HYPRE_BoomerAMGSetMaxNzPerRow</td>
</tr>
<tr>
<td>8</td>
<td>ParaSails</td>
<td>HYPRE_BoomerAMGSetSym, HYPRE_BoomerAMGSetLevel, HYPRE_BoomerAMGSetFilter, HYPRE_BoomerAMGSetThreshold</td>
</tr>
<tr>
<td>9</td>
<td>Euclid</td>
<td>HYPRE_BoomerAMGSetEuclidFile</td>
</tr>
</tbody>
</table>

The default is 6. Also, if no smoother parameters are set via the routines mentioned in the table above, default values are used.
6.2.39

```c
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.

6.2.40

```c
int HYPRE_BoomerAMGSetSmoothNumSweeps (HYPRE_Solver solver, int smooth_num_sweeps)
```

(Optional) Sets the number of sweeps for more complex smoothers. The default is 1.

6.2.41

```c
int HYPRE_BoomerAMGSetPrintLevel (HYPRE_Solver solver, int print_level)
```

(Optional) Requests automatic printing of setup and solve information.

<table>
<thead>
<tr>
<th>print_level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>no printout (default)</td>
</tr>
<tr>
<td>1</td>
<td>print setup information</td>
</tr>
<tr>
<td>2</td>
<td>print solve information</td>
</tr>
<tr>
<td>3</td>
<td>print both setup and solve information</td>
</tr>
</tbody>
</table>

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.

6.2.42

```c
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.

### 6.2.43

```c
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.

### 6.2.44

```c
int HYPRE_BoomerAMGSetNodal (HYPRE_Solver solver, int nodal)
```

(Optional) Sets whether to use the nodal systems version. The default is 0.

### 6.2.45

```c
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 \((0,1,...,k-1,0,1,...,k-1,...)\).

### 6.2.46

```c
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.
6.2.47  

```c
int HYPRE_BoomerAMGSetNumPaths (HYPRE_Solver solver, int num_paths)
```

(Optional) Defines the degree of aggressive coarsening. The default is 1.

6.2.48  

```c
int HYPRE_BoomerAMGSetVariant (HYPRE_Solver solver, int variant)
```

(Optional) Defines which variant of the Schwarz method is used. The following options exist for variant:

<table>
<thead>
<tr>
<th>Variant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>hybrid multiplicative Schwarz method (no overlap across processor boundaries)</td>
</tr>
<tr>
<td>1</td>
<td>hybrid additive Schwarz method (no overlap across processor boundaries)</td>
</tr>
<tr>
<td>2</td>
<td>additive Schwarz method</td>
</tr>
<tr>
<td>3</td>
<td>hybrid multiplicative Schwarz method (with overlap across processor boundaries)</td>
</tr>
</tbody>
</table>

The default is 0.

6.2.49  

```c
int HYPRE_BoomerAMGSetOverlap (HYPRE_Solver solver, int overlap)
```

(Optional) Defines the overlap for the Schwarz method. The following options exist for overlap:

<table>
<thead>
<tr>
<th>Overlap</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>no overlap</td>
</tr>
<tr>
<td>1</td>
<td>minimal overlap (default)</td>
</tr>
<tr>
<td>2</td>
<td>overlap generated by including all neighbors of domain boundaries</td>
</tr>
</tbody>
</table>

6.2.50  

```c
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:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>each point is a domain</td>
</tr>
<tr>
<td>1</td>
<td>each node is a domain (only of interest in &quot;systems&quot; AMG)</td>
</tr>
<tr>
<td>2</td>
<td>each domain is generated by agglomeration (default)</td>
</tr>
</tbody>
</table>

6.2.51

```c
int HYPRE_BoomerAMGSetSchwarzRlxWeight (HYPRE_Solver solver, double schwarz_rlx_weight)
```

(Optional) Defines a smoothing parameter for the additive Schwarz method

6.2.52

```c
int HYPRE_BoomerAMGSetSym (HYPRE_Solver solver, int sym)
```

(Optional) Defines symmetry for ParaSAILS. For further explanation see description of ParaSAILS.

6.2.53

```c
int HYPRE_BoomerAMGSetLevel (HYPRE_Solver solver, int level)
```

(Optional) Defines number of levels for ParaSAILS. For further explanation see description of ParaSAILS.

6.2.54

```c
int HYPRE_BoomerAMGSetThreshold (HYPRE_Solver solver, double threshold)
```

(Optional) Defines threshold for ParaSAILS. For further explanation see description of ParaSAILS.
6.2.55

```
int HYPRE_BoomerAMGSetFilter (HYPRE_Solver solver, double filter)
```

(Optional) Defines filter for ParaSAILS. For further explanation see description of ParaSAILS.

6.2.56

```
int HYPRE_BoomerAMGSetDropTol (HYPRE_Solver solver, double drop_tol)
```

(Optional) Defines drop tolerance for PILUT. For further explanation see description of PILUT.

6.2.57

```
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.

6.2.58

```
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.
6.2.59

\textbf{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)

6.2.60

\textbf{int HYPRE\_BoomerAMGSetNumSamples} (HYPRE\_Solver solver, int num\_samples)

(Optional) Defines the number of sample vectors used in GSMG or LS interpolation

6.3

\textbf{ParCSR ParaSails Preconditioner}

\textbf{Names}

6.3.1 \textbf{int HYPRE\_ParaSailsCreate} (MPI\_Comm comm, HYPRE\_Solver* solver)

\textit{Create a ParaSails preconditioner} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} 162

6.3.2 \textbf{int HYPRE\_ParaSailsDestroy} (HYPRE\_Solver solver)

\textit{Destroy a ParaSails preconditioner} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} 162

6.3.3 \textbf{int HYPRE\_ParaSailsSetup} (HYPRE\_Solver solver, HYPRE\_ParCSRMatrix A, HYPRE\_ParVector b, HYPRE\_ParVector x)

\textit{Set up the ParaSails preconditioner}. \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} 162

6.3.4 \textbf{int HYPRE\_ParaSailsSolve} (HYPRE\_Solver solver, HYPRE\_ParCSRMatrix A, HYPRE\_ParVector b, HYPRE\_ParVector x)

\textit{Apply the ParaSails preconditioner}. \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} 163

6.3.5 \textbf{int HYPRE\_ParaSailsSetParams} (HYPRE\_Solver solver, double thresh, int nlevels)

\textit{Set the threshold and levels parameter for the ParaSails preconditioner}. \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} 163

6.3.6 \textbf{int}
Parallel sparse approximate inverse preconditioner for the ParCSR matrix format.

### 6.3.1

```c
int HYPRE_ParaSailsCreate (MPI_Comm comm, HYPRE_Solver* solver)
```

Create a ParaSails preconditioner

### 6.3.2

```c
int HYPRE_ParaSailsDestroy (HYPRE_Solver solver)
```

Destroy a ParaSails preconditioner

### 6.3.3

```c
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

```c
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

```c
int HYPRE_ParaSailsSetParams (HYPRE_Solver solver, double 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 ≤ thresh ≤ 1. The default value is 0.1.
- **nlevels** [IN] Value of levels parameter, 0 ≤ nlevels. The default value is 1.
6.3.6

**int HYPRE_ParaSailsSetFilter** (HYPRE_Solver solver, double 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 is used 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:
  - 0: nonsymmetric and/or indefinite problem, and nonsymmetric preconditioner
  - 1: SPD problem, and SPD (factored) preconditioner
  - 2: nonsymmetric, definite problem, and SPD (factored) preconditioner

6.3.8

**int HYPRE_ParaSailsSetLoadbal** (HYPRE_Solver solver, double loadbal)

Set the load balance parameter for the ParaSails preconditioner.
ParCSR Solvers

Parameters:

- **solver** [IN] Preconditioner object for which to set the load balance parameter.

- **loadbal** [IN] Value of the load balance parameter, 0 ≤ loadbal ≤ 1. A zero value indicates that no load balance is attempted; a value of unity indicates that perfect load balance will be attempted. The recommended value is 0.9 to balance the overhead of data exchanges for load balancing. No load balancing is needed if the preconditioner is very sparse and fast 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 value indicates that the pattern of the preconditioner should be 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 ............................................. 167

6.4.2 int HYPRE_EuclidDestroy (HYPRE_Solver solver)
Destroy a Euclid object ............................................. 167

6.4.3 int HYPRE_EuclidSetup (HYPRE_Solver solver, HYPRE_ParCSRMatrix A,
                                  HYPRE_ParVector b, HYPRE_ParVector x)
Set up the Euclid preconditioner. ................................. 167

6.4.4 int HYPRE_EuclidSolve (HYPRE_Solver solver, HYPRE_ParCSRMatrix A,
                                  HYPRE_ParVector b, HYPRE_ParVector x)
Apply the Euclid preconditioner. ................................. 168

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). ......................... 168

6.4.6 int HYPRE_EuclidSetParamsFromFile (HYPRE_Solver solver, char* filename)
Insert (name, value) pairs in Euclid’s options database. ......................... 168

MPI Parallel ILU preconditioner

Options summary:
<table>
<thead>
<tr>
<th>Option</th>
<th>Default</th>
<th>Synopsis</th>
</tr>
</thead>
<tbody>
<tr>
<td>-level</td>
<td>1</td>
<td>ILU((k)) factorization level</td>
</tr>
<tr>
<td>-bj</td>
<td>0 (false)</td>
<td>Use Block Jacobi ILU instead of PILU</td>
</tr>
<tr>
<td>-eu_stats</td>
<td>0 (false)</td>
<td>Print internal timing and statistics</td>
</tr>
<tr>
<td>-eu_mem</td>
<td>0 (false)</td>
<td>Print internal memory usage</td>
</tr>
</tbody>
</table>

6.4.1

```c
int HYPRE_EuclidCreate (MPI_Comm comm, HYPRE_Solver* solver)
```

Create a Euclid object

6.4.2

```c
int HYPRE_EuclidDestroy (HYPRE_Solver solver)
```

Destroy a Euclid object

6.4.3

```c
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

```c
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

```c
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

```c
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:

```bash
> 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

---

### ParCSR Pilut Preconditioner

**Names**

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)

Precondition the system

6.5.4 int

HYPRE_ParCSRPIlutSolve (HYPRE_Solver solver,
HYPRE_ParCSRMatrix A,
HYPRE_ParVector b, HYPRE_ParVector x)

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, double tol)

(Optional)

6.5.7 int

HYPRE_ParCSRPIlutSetFactorRowSize (HYPRE_Solver solver, int size)

(Optional)
Create a preconditioner object

6.5.2

int HYPRE_ParCSRPrepilutDestroy (HYPRE_Solver solver)

Destroy a preconditioner object

6.5.3

int HYPRE_ParCSRPrepilutSetup (HYPRE_Solver solver, HYPRE_ParCSRMatrix A, HYPRE_ParVector b, HYPRE_ParVector x)

Precondition the system

6.5.4

int HYPRE_ParCSRPrepilutSolve (HYPRE_Solver solver, HYPRE_ParCSRMatrix A, HYPRE_ParVector b, HYPRE_ParVector x)

(Optional) Set maximum number of iterations

6.5.5

int HYPRE_ParCSRPrepilutSetMaxIter (HYPRE_Solver solver, int max_iter)
6.5.6

int
HYPRE_ParCSRPreSetDropTolerance (HYPRE_Solver solver, double tol)

(Optional)

6.5.7

int HYPRE_ParCSRPreSetFactorRowSize (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 ............................... 173

6.6.2 int
HYPRE_AMSDestroy (HYPRE_Solver solver)
Destroy an AMS solver object ............................... 173

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. ................. 174

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. ......... 174

6.6.5 int
HYPRE_AMSSetDimension (HYPRE_Solver solver, int dim)
(Optional) Sets the problem dimension (2 or 3). ............. 174

6.6.6 int
HYPRE_AMSSetDiscreteGradient (HYPRE_Solver solver, HYPRE_ParCSRMatrix G)
Sets the discrete gradient matrix G. ....................... 175

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.

**HYPRE_AMSSetEdgeConstantVectors** (HYPRE_Solver solver,  
HYPRE_ParVector Gx,  
HYPRE_ParVector Gy,  
HYPRE_ParVector Gz)

Sets the vectors $G_x$, $G_y$ and $G_z$ which give the representations of the constant vector fields $(1, 0, 0)$, $(0, 1, 0)$ and $(0, 0, 1)$ in the edge element basis.

**HYPRE_AMSSetAlphaPoissonMatrix** (HYPRE_Solver solver,  
HYPRE_ParCSRMatrix A_alpha)

(Optional) Sets the matrix $A_\alpha$ corresponding to the Poisson problem with coefficient $\alpha$ (the curl-curl term coefficient in the Maxwell problem).

**HYPRE_AMSSetBetaPoissonMatrix** (HYPRE_Solver solver,  
HYPRE_ParCSRMatrix A_beta)

(Optional) Sets the matrix $A_\beta$ corresponding to the Poisson problem with coefficient $\beta$ (the mass term coefficient in the Maxwell problem).

**HYPRE_AMSSetMaxIter** (HYPRE_Solver solver, int maxit)

(Optional) Sets maximum number of iterations, if AMS is used as a solver.

**HYPRE_AMSSetCycleType** (HYPRE_Solver solver, int cycle_type)

(Optional) Choose which three-level solver to use.

**HYPRE_AMSSetPrintLevel** (HYPRE_Solver solver, int print_level)

(Optional) Control how much information is printed during the solution iterations.

**HYPRE_AMSSetSmoothingOptions** (HYPRE_Solver solver, int relax_type,  
int relax_times, double relax_weight,  
double omega)

(Optional) Sets relaxation parameters for $A$. 
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

```c
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:**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>solver</code></td>
<td>[IN] object to be set up.</td>
</tr>
<tr>
<td><code>A</code></td>
<td>[IN] ParCSR matrix used to construct the solver/preconditioner.</td>
</tr>
<tr>
<td><code>b</code></td>
<td>Ignored by this function.</td>
</tr>
<tr>
<td><code>x</code></td>
<td>Ignored by this function.</td>
</tr>
</tbody>
</table>

6.6.4

```c
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:**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>solver</code></td>
<td>[IN] solver or preconditioner object to be applied.</td>
</tr>
<tr>
<td><code>A</code></td>
<td>[IN] ParCSR matrix, matrix of the linear system to be solved</td>
</tr>
<tr>
<td><code>b</code></td>
<td>[IN] right hand side of the linear system to be solved</td>
</tr>
<tr>
<td><code>x</code></td>
<td>[OUT] approximated solution of the linear system to be solved</td>
</tr>
</tbody>
</table>

6.6.5

```c
int HYPRE_AMSSetDimension (HYPRE_Solver solver, int dim)
```

(Optional) Sets the problem dimension (2 or 3). The default is 3.
6.6.6

```c
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

```c
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

```c
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

```c
int HYPRE_AMSSetAlphaPoissonMatrix (HYPRE_Solver solver,
                                   HYPRE_ParCSRMatrix A_alpha)
```
(Optional) Sets the matrix $A_\alpha$ corresponding to the Poisson problem with coefficient $\alpha$ (the curl-curl term coefficient in the Maxwell problem).

If this function is called, the coarse space solver on the range of $\Pi^T$ is a block-diagonal version of $A_\Pi$. If this function is not called, the coarse space solver on the range of $\Pi^T$ is constructed as $\Pi^T A \Pi$ in HYPRE_AMSSetup(). See the user’s manual for more details.

### 6.6.10

```c
int HYPRE_AMSSetBetaPoissonMatrix (HYPRE_Solver solver, HYPRE_ParCSRMatrix A_beta)
```

(Optional) Sets the matrix $A_\beta$ corresponding to the Poisson problem with coefficient $\beta$ (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 $\beta$ is identically 0 and use two-level (instead of three-level) methods. See the user’s manual for more details.

### 6.6.11

```c
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.

### 6.6.12

```c
int HYPRE_AMSSetTol (HYPRE_Solver solver, double 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 $10^{-6}$.
6.6.13

\texttt{int HYPRE\_AMSSetCycleType} (HYPRE\_Solver solver, int cycle\_type)

(Optional) Choose which three-level solver to use. Possible values are:

<table>
<thead>
<tr>
<th>int</th>
<th>3-level multiplicative solver (01210)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>3-level multiplicative solver (02120)</td>
</tr>
<tr>
<td>5</td>
<td>3-level multiplicative solver (0102010)</td>
</tr>
<tr>
<td>7</td>
<td>3-level multiplicative solver (0201020)</td>
</tr>
<tr>
<td>2</td>
<td>3-level additive solver (0+1+2)</td>
</tr>
<tr>
<td>4</td>
<td>3-level additive solver (010+2)</td>
</tr>
<tr>
<td>6</td>
<td>3-level additive solver (1+020)</td>
</tr>
<tr>
<td>8</td>
<td>3-level additive solver (010+020)</td>
</tr>
</tbody>
</table>

The default is 1. See the user’s manual for more details.

6.6.14

\texttt{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).

6.6.15

\texttt{int HYPRE\_AMSSetSmoothingOptions} (HYPRE\_Solver solver, int relax\_type, int relax\_times, double relax\_weight, double omega)

(Optional) Sets relaxation parameters for $A$. The defaults are 2, 1, 1.0, 1.0.

6.6.16

\texttt{int HYPRE\_AMSSetAlphaAMGOptions} (HYPRE\_Solver solver, int alpha\_coarsen\_type, int alpha\_agg\_levels, int alpha\_relax\_type, double alpha\_strength\_threshold)
(Optional) Sets AMG parameters for $B_{II}$. The defaults are 10, 1, 3, 0.25. See the user’s manual for more details.

### 6.6.17

```c
int HYPRE_AMSSetBetaAMGOptions (HYPRE_Solver solver, int beta_coarsen_type, int beta_agg_levels, int beta_relax_type, double beta_strength_threshold)
```

(Optional) Sets AMG parameters for $B_{G}$. The defaults are 10, 1, 3, 0.25. See the user’s manual for more details.

### 6.6.18

```c
int HYPRE_AMSGetNumIterations (HYPRE_Solver solver, int* num_iterations)
```

Returns the number of iterations taken

### 6.6.19

```c
int HYPRE_AMSGetFinalRelativeResidualNorm (HYPRE_Solver solver, double* rel_resid_norm)
```

Returns the norm of the final relative residual

### 6.6.20

```c
int HYPRE_AMSConstructDiscreteGradient (HYPRE_ParCSRMatrix A, HYPRE_ParVector x_coord, int* edge_vertex, HYPRE_ParCSRMatrix* G)
```
Construct and return the 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 edge $i$ depends only on the sign of edge vertex$[2*i+1]$ - edge vertex$[2*i]$.

## 6.7 ParCSR Hybrid Solver

### Names

6.7.1  
```c
int HYPRE_ParCSRHybridCreate ( HYPRE_Solver* solver)
        Create solver object
```

6.7.2  
```c
int HYPRE_ParCSRHybridDestroy (HYPRE_Solver solver)
        Destroy solver object
```

6.7.3  
```c
int HYPRE_ParCSRHybridSetup (HYPRE_Solver solver, 
                                HYPRE_ParCSRMatrix A, 
                                HYPRE_ParVector b,  HYPRE_ParVector x)
        Setup the hybrid solver
```

6.7.4  
```c
int HYPRE_ParCSRHybridSolve ( HYPRE_Solver solver, 
                               HYPRE_ParCSRMatrix A, 
                               HYPRE_ParVector b,  HYPRE_ParVector x)
        Solve linear system
```

6.7.5  
```c
int HYPRE_ParCSRHybridSetTol (HYPRE_Solver solver, double tol)
        Set the convergence tolerance for the Krylov solver.
```

6.7.6  
```c
int HYPRE_ParCSRHybridSetConvergenceTol (HYPRE_Solver solver, 
                                         double cf_tol)
        Set the desired convergence factor
```

6.7.7  
```c
int HYPRE_ParCSRHybridSetDSCGMaxIter (HYPRE_Solver solver, 
                                       int dscg_max_its)
        Set the maximal number of iterations for the diagonally preconditioned solver
```

6.7.8  
```c
int HYPRE_ParCSRHybridSetPCGMaxIter (HYPRE_Solver solver, 
                                      int pcg_max_its)
        Set the maximal number of iterations for the AMG preconditioned solver
```

6.7.9  
```c
int HYPRE_ParCSRHybridSetSolverType (HYPRE_Solver solver, 
                                      int solver_type)
        Set the desired solver type
```

6.7.10 int
6 ParCSR Solvers

**HYPRE_ParCSRHybridSetKDim** (HYPRE_Solver solver, int k_dim)

Set the Krylov dimension for restarted GMRES.

6.7.11

**HYPRE_ParCSRHybridSetTwoNorm** (HYPRE_Solver solver, int two_norm)

Set the type of norm for PCG

6.7.12

**HYPRE_ParCSRHybridSetStopCrit** (HYPRE_Solver solver, int stop_crit)

Set the choice of stopping criterion for PCG

6.7.13

**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

6.7.14

**HYPRE_ParCSRHybridSetLogging** (HYPRE_Solver solver, int logging)

Set logging parameter (default: 0, no logging)

6.7.15

**HYPRE_ParCSRHybridSetPrintLevel** (HYPRE_Solver solver,

int print_level)

Set print level (default: 0, no printing)

6.7.16

**HYPRE_ParCSRHybridSetMaxRowSum** (HYPRE_Solver solver,

double max_row_sum)

(Optional) Sets a parameter to modify the definition of strength for diagonal dominant portions of the matrix.

6.7.17

**HYPRE_ParCSRHybridSetTruncFactor** (HYPRE_Solver solver,

double trunc_factor)

(Optional) Defines a truncation factor for the interpolation.

6.7.18

**HYPRE_ParCSRHybridSetMaxLevels** (HYPRE_Solver solver,

int max_levels)

(Optional) Defines the maximal number of levels used for AMG.

6.7.19

**HYPRE_ParCSRHybridSetMeasureType** (HYPRE_Solver solver,

int measure_type)

(Optional) Defines whether local or global measures are used

6.7.20

**HYPRE_ParCSRHybridSetCoarsenType** (HYPRE_Solver solver,

int coarsen_type)

(Optional) Defines which parallel coarsening algorithm is used

6.7.21

**HYPRE_ParCSRHybridSetStrongThreshold** (HYPRE_Solver solver,

double strong_threshold)

(Optional) Sets AMG strength threshold.

6.7.16

**HYPRE_ParCSRHybridSetMaxRowSum** (HYPRE_Solver solver,

double max_row_sum)

(Optional) Sets a parameter to modify the definition of strength for diagonal dominant portions of the matrix.

6.7.17

**HYPRE_ParCSRHybridSetTruncFactor** (HYPRE_Solver solver,

double trunc_factor)

(Optional) Defines a truncation factor for the interpolation.

6.7.18

**HYPRE_ParCSRHybridSetMaxLevels** (HYPRE_Solver solver,

int max_levels)

(Optional) Defines the maximal number of levels used for AMG.

6.7.19

**HYPRE_ParCSRHybridSetMeasureType** (HYPRE_Solver solver,

int measure_type)

(Optional) Defines whether local or global measures are used

6.7.20

**HYPRE_ParCSRHybridSetCoarsenType** (HYPRE_Solver solver,

int coarsen_type)

(Optional) Defines which parallel coarsening algorithm is used

6.7.21
6.7.23 int HYPRE_ParCSRHybridSetCycleType ( HYPRE_Solver solver, int cycle_type )

(Optional) Defines the type of cycle.

6.7.24 int HYPRE_ParCSRHybridSetNumSweeps ( HYPRE_Solver solver, int num_sweeps )

(Optional) Sets the number of sweeps.

6.7.25 int HYPRE_ParCSRHybridSetRelaxType ( HYPRE_Solver solver, int relax_type )

(Optional) Defines the smoother to be used.

6.7.26 int HYPRE_ParCSRHybridSetCycleRelaxOrder ( HYPRE_Solver solver, int relax_order )

(Optional) Defines in which order the points are relaxed.

6.7.27 int HYPRE_ParCSRHybridSetRelaxWt ( HYPRE_Solver solver, double relax_wt )

(Optional) Defines the relaxation weight for smoothed Jacobi and hybrid SOR on all levels.

6.7.28 int HYPRE_ParCSRHybridSetLevelRelaxWt ( HYPRE_Solver solver, double relax_wt, int level )

(Optional) Defines the relaxation weight for smoothed Jacobi and hybrid SOR on the user defined level.

6.7.29 int HYPRE_ParCSRHybridSetOuterWt ( HYPRE_Solver solver, double outer_wt )

(Optional) Defines the outer relaxation weight for hybrid SOR and SSOR on all levels.

6.7.30 int HYPRE_ParCSRHybridSetLevelOuterWt ( HYPRE_Solver solver, double outer_wt, int level )

(Optional) Defines the outer relaxation weight for hybrid SOR or SSOR on the user defined level.

6.7.31 int HYPRE_ParCSRHybridSetAggNumLevels ( HYPRE_Solver solver, int agg_num_levels )

(Optional) Defines the number of levels of aggressive coarsening, starting with the finest level.

6.7.32 int
6.7.1

```
int HYPRE_ParCSRHybridCreate ( HYPRE_Solver* solver)
```

Create solver object

HYPRE_ParCSRHybridSetNumPaths ( HYPRE_Solver solver, int num_paths )

(Optional) Defines the degree of aggressive coarsening.

(192)

HYPRE_ParCSRHybridSetNumberOfFunctions ( HYPRE_Solver solver, int num_functions )

(Optional) Sets the size of the system of PDEs, if using the systems version.

(192)

HYPRE_ParCSRHybridSetDofFunc ( HYPRE_Solver solver, int* dof_func )

(Optional) Sets the mapping that assigns the function to each variable, if using the systems version.

(192)

HYPRE_ParCSRHybridSetNodal ( HYPRE_Solver solver, int nodal )

(Optional) Sets whether to use the nodal systems version.

(192)

HYPRE_ParCSRHybridGetNumIterations (HYPRE_Solver solver, int* num_its)

Retrieves the total number of iterations

(193)

HYPRE_ParCSRHybridGetDSCGNumIterations (HYPRE_Solver solver, int* dscg_num_its)

Retrieves the number of iterations used by the diagonally scaled solver

(193)

HYPRE_ParCSRHybridGetPCGNumIterations (HYPRE_Solver solver, int* pcg_num_its)

Retrieves the number of iterations used by the AMG preconditioned solver

(193)

HYPRE_ParCSRHybridGetFinalRelativeResidualNorm (HYPRE_Solver solver, double* norm)

Retrieves the final relative residual norm

(193)
6.7.2

`int HYPRE_ParCSRHybridDestroy (HYPRE_Solver solver)`

Destroy solver object

6.7.3

`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.

6.7.4

`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
6.7.5

int HYPRE_ParCSRHybridSetTol (HYPRE_Solver solver, double tol)

Set the convergence tolerance for the Krylov solver. The default is 1.e-7.

6.7.6

int HYPRE_ParCSRHybridSetConvergenceTol (HYPRE_Solver solver, double cf_tol)

Set the desired convergence factor

6.7.7

int HYPRE_ParCSRHybridSetDSCGMaxIter (HYPRE_Solver solver, int dscg_max_its)

Set the maximal number of iterations for the diagonally preconditioned solver

6.7.8

int HYPRE_ParCSRHybridSetPCGMaxIter (HYPRE_Solver solver, int pcg_max_its)

Set the maximal number of iterations for the AMG preconditioned solver
6.7.9

```c
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

6.7.10

```c
int HYPRE_ParCSRHybridSetKDim (HYPRE_Solver solver, int k_dim)
```

Set the Krylov dimension for restarted GMRES. The default is 5.

6.7.11

```c
int HYPRE_ParCSRHybridSetTwoNorm (HYPRE_Solver solver, int two_norm)
```

Set the type of norm for PCG

6.7.12

```c
int HYPRE_ParCSRHybridSetStopCrit (HYPRE_Solver solver, int stop_crit)
```

Set the choice of stopping criterion for PCG
6.7.13

```c
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

6.7.14

```c
int HYPRE_ParCSRHybridSetLogging (HYPRE_Solver solver, int logging)
```

Set logging parameter (default: 0, no logging)

6.7.15

```c
int HYPRE_ParCSRHybridSetPrintLevel (HYPRE_Solver solver, int print_level)
```

Set print level (default: 0, no printing)

6.7.16

```c
int HYPRE_ParCSRHybridSetStrongThreshold ( HYPRE_Solver solver, double 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.
6.7.17

```c
int HYPRE_ParCSRHybridSetMaxRowSum ( HYPRE_Solver solver, double 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.

6.7.18

```c
int HYPRE_ParCSRHybridSetTruncFactor ( HYPRE_Solver solver, double trunc_factor )
```

(Optional) Defines a truncation factor for the interpolation. The default is 0.

6.7.19

```c
int HYPRE_ParCSRHybridSetMaxLevels ( HYPRE_Solver solver, int max_levels )
```

(Optional) Defines the maximal number of levels used for AMG. The default is 25.

6.7.20

```c
int HYPRE_ParCSRHybridSetMeasureType ( HYPRE_Solver solver, int measure_type )
```

(Optional) Defines whether local or global measures are used
6.7.21

```c
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`:

```
<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>CLJP-coarsening (a parallel coarsening algorithm using independent sets).</td>
</tr>
<tr>
<td>1</td>
<td>classical Ruge-Stueben coarsening on each processor, no boundary treatment</td>
</tr>
<tr>
<td>2</td>
<td>classical Ruge-Stueben coarsening on each processor, followed by a third pass, which adds coarse points on the boundaries</td>
</tr>
<tr>
<td>3</td>
<td>Falgout coarsening (uses 1 first, followed by CLJP using the interior coarse points generated by 1 as its first independent set)</td>
</tr>
<tr>
<td>4</td>
<td>CLJP-coarsening (using a fixed random vector, for debugging purposes only)</td>
</tr>
<tr>
<td>5</td>
<td>PMIS-coarsening (a parallel coarsening algorithm using independent sets with lower complexities than CLJP, might also lead to slower convergence)</td>
</tr>
<tr>
<td>6</td>
<td>PMIS-coarsening (using a fixed random vector, for debugging purposes only)</td>
</tr>
<tr>
<td>7</td>
<td>Falgout coarsening (uses one pass Ruge-Stueben on each processor independently, followed by PMIS using the interior C-points as its first independent set)</td>
</tr>
<tr>
<td>8</td>
<td>one-pass Ruge-Stueben coarsening on each processor, no boundary treatment</td>
</tr>
</tbody>
</table>
```

The default is 6.

6.7.22

```c
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.

6.7.23

```c
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.
6.7.24

\begin{verbatim}
int HYPRE_ParCSRHybridSetCycleNumSweeps ( HYPRE_Solver solver, int num_sweeps, int k )
\end{verbatim}

(Optional) Sets the number of sweeps at a specified cycle. There are the following options for \( k \):

\begin{itemize}
  \item the down cycle if \( k = 1 \)
  \item the up cycle if \( k = 2 \)
  \item the coarsest level if \( k = 3 \).
\end{itemize}

6.7.25

\begin{verbatim}
int HYPRE_ParCSRHybridSetRelaxType ( HYPRE_Solver solver, int relax_type )
\end{verbatim}

(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 \( \text{relax\_type} \):

\begin{itemize}
  \item 0 Jacobi
  \item 1 Gauss-Seidel, sequential (very slow!)
  \item 2 Gauss-Seidel, interior points in parallel, boundary sequential (slow!)
  \item 3 hybrid Gauss-Seidel or SOR, forward solve
  \item 4 hybrid Gauss-Seidel or SOR, backward solve
  \item 5 hybrid chaotic Gauss-Seidel (works only with OpenMP)
  \item 6 hybrid symmetric Gauss-Seidel or SSOR
  \item 9 Gaussian elimination (only on coarsest level)
\end{itemize}

6.7.26

\begin{verbatim}
int HYPRE_ParCSRHybridSetCycleRelaxType ( HYPRE_Solver solver, int relax_type, int k )
\end{verbatim}

(Optional) Defines the smoother at a given cycle. For options of \( \text{relax\_type} \) see description of HYPRE_BoomerAMGSetRelaxType). Options for \( k \) are
6.7.27

<table>
<thead>
<tr>
<th>Relax Order</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Natural or lexicographic order on each processor</td>
</tr>
<tr>
<td>1</td>
<td>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.</td>
</tr>
</tbody>
</table>

The default is 1 (CF-relaxation).

6.7.28

<table>
<thead>
<tr>
<th>Relax Weight</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt; 0</td>
<td>This assigns the given relaxation weight on all levels</td>
</tr>
<tr>
<td>= 0</td>
<td>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)</td>
</tr>
<tr>
<td>= -k</td>
<td>The relaxation weight is determined with at most $k$ CG steps on each level (this should only be used for symmetric positive definite problems)</td>
</tr>
</tbody>
</table>

The default is 1.

6.7.29

int HYPRE_ParCSRHybridSetLevelRelaxWt (HYPRE_Solver solver, double 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.

### 6.7.30

```c
int HYPRE_ParCSRHybridSetOuterWt ( HYPRE_Solver solver, double 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.

### 6.7.31

```c
int HYPRE_ParCSRHybridSetLevelOuterWt ( HYPRE_Solver solver, double 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.

### 6.7.32

```c
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.
6.7.33

```c
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.*

6.7.34

```c
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.

6.7.35

```c
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,...).

6.7.36

```c
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).
6.7.37

```c
int HYPRE_ParCSRHybridGetNumIterations (HYPRE_Solver solver, int* num_its)
```

Retrieves the total number of iterations

6.7.38

```c
int HYPRE_ParCSRHybridGetDSCGNumIterations (HYPRE_Solver solver, int* dscg_num_its)
```

Retrieves the number of iterations used by the diagonally scaled solver

6.7.39

```c
int HYPRE_ParCSRHybridGetPCGNumIterations (HYPRE_Solver solver, int* pcg_num_its)
```

Retrieves the number of iterations used by the AMG preconditioned solver

6.7.40

```c
int HYPRE_ParCSRHybridGetFinalRelativeResidualNorm (HYPRE_Solver solver, double* norm)
```

Retrieves the final relative residual norm
6.8 ParCSR PCG Solver

Names

6.8.1 int HYPRE_ParCSRPCGCreate (MPI_Comm comm, HYPRE_Solver* solver)
Create a solver object ................................................. 195

6.8.2 int HYPRE_ParCSRPCGDestroy (HYPRE_Solver solver)
Destroy a solver object .................................................... 140

6.8.3 int HYPRE_ParCSRPCGSetup (HYPRE_Solver solver,
HYPRE_ParCSRMatrix A,
HYPRE_ParVector b, HYPRE_ParVector x)
....................................................................................... 196

6.8.4 int HYPRE_ParCSRPCGSolve (HYPRE_Solver solver,
HYPRE_ParCSRMatrix A,
HYPRE_ParVector b, HYPRE_ParVector x)
Solve the system ............................................................ 196

6.8.5 int HYPRE_ParCSRPCGSetTol (HYPRE_Solver solver, double tol)
(Optional) Set the convergence tolerance .............................. 196

6.8.6 int HYPRE_ParCSRPCGSetMaxIter (HYPRE_Solver solver, int max_iter)
(Optional) Set maximum number of iterations .......................... 196

6.8.7 int HYPRE_ParCSRPCGSetTwoNorm (HYPRE_Solver solver, int two_norm)
(Optional) Use the two-norm in stopping criteria ........................ 196

6.8.8 int HYPRE_ParCSRPCGSetRelChange (HYPRE_Solver solver, int rel_change)
(Optional) Additionally require that the relative difference in successive it-
erates be small ................................................................ 197

6.8.9 int HYPRE_ParCSRPCGSetPrecond (HYPRE_Solver solver,
HYPRE_PtrToParSolverFcn precond,
HYPRE_PtrToParSolverFcn
precond_setup,
HYPRE_Solver precond_solver)
(Optional) Set the preconditioner to use ................................ 197

6.8.10 int HYPRE_ParCSRPCGGetPrecond (HYPRE_Solver solver,
HYPRE_Solver* precond_data)
....................................................................................... 197

6.8.11 int
6.8.12 \textbf{int HYPRE\_ParCSRPCGSetPrintLevel} (HYPRE\_Solver solver, \hspace{1em} \text{int print\_level})

\textbf{(Optional) Set the print level} ............................................. \hspace{1em} 198

6.8.13 \textbf{int HYPRE\_ParCSRPCGGetNumIterations} (HYPRE\_Solver solver, \hspace{1em} \text{int* num\_iterations})

\textit{Return the number of iterations taken} ................................. \hspace{1em} 198

6.8.14 \textbf{int HYPRE\_ParCSRPCGGetFinalRelativeResidualNorm} (HYPRE\_Solver solver, \hspace{1em} \text{double* norm})

\textit{Return the norm of the final relative residual} ......................... \hspace{1em} 198

6.8.15 \textbf{int HYPRE\_ParCSRDiagScaleSetup} (HYPRE\_Solver solver, \hspace{1em} \text{HYPRE\_ParCSRMatrix A,} \hspace{1em} \text{HYPRE\_ParVector y,} \hspace{1em} \text{HYPRE\_ParVector x})

\textit{Setup routine for diagonal preconditioning} .......................... \hspace{1em} 198

6.8.16 \textbf{int HYPRE\_ParCSRDiagScale} (HYPRE\_Solver solver, \hspace{1em} \text{HYPRE\_ParCSRMatrix HA,} \hspace{1em} \text{HYPRE\_ParVector Hy,} \hspace{1em} \text{HYPRE\_ParVector Hx})

\textit{Solve routine for diagonal preconditioning} .......................... \hspace{1em} 199

---

**6.8.1** \textbf{int HYPRE\_ParCSRPCGCreate} (MPI\_Comm comm, HYPRE\_Solver* solver)

Create a solver object

**6.8.2** \textbf{int HYPRE\_ParCSRPCGDestroy} (HYPRE\_Solver solver)

Destroy a solver object
6.8.3

```c
int HYPRE_ParCSRPCGSetup (HYPRE_Solver solver, HYPRE_ParCSRMatrix A, HYPRE_ParVector b, HYPRE_ParVector x)
```

6.8.4

```c
int HYPRE_ParCSRPCGSolve (HYPRE_Solver solver, HYPRE_ParCSRMatrix A, HYPRE_ParVector b, HYPRE_ParVector x)
```

Solve the system

6.8.5

```c
int HYPRE_ParCSRPCGSetTol (HYPRE_Solver solver, double tol)
```

(Optional) Set the convergence tolerance

6.8.6

```c
int HYPRE_ParCSRPCGSetMaxIter (HYPRE_Solver solver, int max_iter)
```

(Optional) Set maximum number of iterations

6.8.7

```c
int HYPRE_ParCSRPCGSetTwoNorm (HYPRE_Solver solver, int two_norm)
```

(Optional) Use the two-norm in stopping criteria
6.8.8

int HYPRE_ParCSRPCGSetRelChange (HYPRE_Solver solver, int rel_change)

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

6.8.9

int HYPRE_ParCSRPCGSetPrecond (HYPRE_Solver solver,
HYPRE_PtrToParSolverFcn precond, HYPRE_PtrToParSolverFcn precond_setup,
HYPRE_Solver precond_solver)

(Optional) Set the preconditioner to use

6.8.10

int HYPRE_ParCSRPCGGetPrecond (HYPRE_Solver solver, HYPRE_Solver* precond_data)

6.8.11

int HYPRE_ParCSRPCGSetLogging (HYPRE_Solver solver, int logging)

(Optional) Set the amount of logging to do
6.8.12

\begin{verbatim}
int HYPRE_ParCSRPCGSetPrintLevel (HYPRE_Solver solver, int print_level)
\end{verbatim}

(Optional) Set the print level

6.8.13

\begin{verbatim}
int HYPRE_ParCSRPCGGetNumIterations (HYPRE_Solver solver, int* num_iterations)
\end{verbatim}

Return the number of iterations taken

6.8.14

\begin{verbatim}
int HYPRE_ParCSRPCGGetFinalRelativeResidualNorm (HYPRE_Solver solver, double* norm)
\end{verbatim}

Return the norm of the final relative residual

6.8.15

\begin{verbatim}
int HYPRE_ParCSRDiagScaleSetup (HYPRE_Solver solver, HYPRE_ParCSRMatrix A, HYPRE_ParVector y, HYPRE_ParVector x)
\end{verbatim}

Setup routine for diagonal preconditioning
6.8.16

```c
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

```c
HYPRE_ParCSRGMRESCreate (MPI_Comm comm, HYPRE_Solver* solver)
```

Create a solver object .......................... 200

6.9.2 int

```c
HYPRE_ParCSRGMRESDestroy (HYPRE_Solver solver)
```

Destroy a solver object .......................... 200

6.9.3 int

```c
HYPRE_ParCSRGMRESSetup (HYPRE_Solver solver,
                         HYPRE_ParCSRMatrix A,
                         HYPRE_ParVector b,
                         HYPRE_ParVector x)
```

.......................................................................................... 201

6.9.4 int

```c
HYPRE_ParCSRGMRESSolve (HYPRE_Solver solver,
                         HYPRE_ParCSRMatrix A,
                         HYPRE_ParVector b, HYPRE_ParVector x)
```

Solve the system .................................................. 201

6.9.5 int

```c
HYPRE_ParCSRGMRESSetKDim (HYPRE_Solver solver, int k_dim)
```

(Optional) Set the maximum size of the Krylov space .......................... 201

6.9.6 int

```c
HYPRE_ParCSRGMRESSetTol (HYPRE_Solver solver, double tol)
```

(Optional) Set the convergence tolerance .......................... 201

6.9.7 int

```c
HYPRE_ParCSRGMRESSetMaxIter (HYPRE_Solver solver, int max_iter)
```

(Optional) Set maximum number of iterations .......................... 201

6.9.8 int
HYPRE_ParCSRGMRESSetPrecond (HYPRE_Solver solver,
    HYPRE_PtrToParSolverFcn precond,
    HYPRE_PtrToParSolverFcn precond_setup,
    HYPRE_Solver precond_solver)

(Optional) Set the preconditioner to use ........................................ 202

6.9.9 int HYPRE_ParCSRGMRESGetPrecond (HYPRE_Solver solver,

    HYPRE_Solver* precond_data)

............................................................... 202

6.9.10 int HYPRE_ParCSRGMRESSetLogging (HYPRE_Solver solver, int logging)

(Optional) Set the amount of logging to do ............................... 202

6.9.11 int HYPRE_ParCSRGMRESSetPrintLevel (HYPRE_Solver solver,

    int print_level)

(Optional) Set print level ......................................................... 202

6.9.12 int HYPRE_ParCSRGMRESGetNumIterations (HYPRE_Solver solver,

    int* num_iterations)

Return the number of iterations taken ........................................ 203

6.9.13 int HYPRE_ParCSRGMRESGetFinalRelativeResidualNorm (HYPRE_Solver solver,

    double* norm)

Return the norm of the final relative residual .................................. 203

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
6.9.3

```c
int HYPRE_ParCSRGMRESSetup (HYPRE_Solver solver,
                           HYPRE_ParCSRMatrix A, HYPRE_ParVector b, HYPRE_ParVector x)
```

6.9.4

```c
int HYPRE_ParCSRGMRESSolve (HYPRE_Solver solver, HYPRE_ParCSRMatrix
                           A, HYPRE_ParVector b, HYPRE_ParVector x)
```

Solve the system

6.9.5

```c
int HYPRE_ParCSRGMRESSetKDim (HYPRE_Solver solver, int k_dim)
```

(Optional) Set the maximum size of the Krylov space

6.9.6

```c
int HYPRE_ParCSRGMRESSetTol (HYPRE_Solver solver, double tol)
```

(Optional) Set the convergence tolerance

6.9.7

```c
int HYPRE_ParCSRGMRESSetMaxIter (HYPRE_Solver solver, int max_iter)
```

(Optional) Set maximum number of iterations
6.9.8

int HYPRE_ParCSRGMRESSetPrecond (HYPRE_Solver solver,
HYPRE_PtrToParSolverFcn precond, HYPRE_PtrToParSolverFcn precond_setup,
HYPRE_Solver precond_solver)

(Optional) Set the preconditioner to use

6.9.9

int HYPRE_ParCSRGMRESGetPrecond (HYPRE_Solver solver, HYPRE_Solver*
precond_data)

6.9.10

int HYPRE_ParCSRGMRESSetLogging (HYPRE_Solver solver, int logging)

(Optional) Set the amount of logging to do

6.9.11

int HYPRE_ParCSRGMRESSetPrintLevel (HYPRE_Solver solver, int print_level)

(Optional) Set print level
6.9.12

int HYPRE_ParCSRGMRESGetNumIterations (HYPRE_Solver solver, int* num_iterations)

Return the number of iterations taken

6.9.13

int HYPRE_ParCSRGMRESGetFinalRelativeResidualNorm (HYPRE_Solver solver, double* norm)

Return the norm of the final relative residual

6.10

ParCSR BiCGSTAB Solver

Names

6.10.1 int HYPRE_ParCSRBiCGSTABCreate (MPI_Comm comm,
                                          HYPRE_Solver* solver)
          Create a solver object ......................................... 205

6.10.2 int HYPRE_ParCSRBiCGSTABDestroy (HYPRE_Solver solver)
          Destroy a solver object ......................................... 205

6.10.3 int HYPRE_ParCSRBiCGSTABSetup (HYPRE_Solver solver,
                                    HYPRE_ParCSRMatrix A,
                                    HYPRE_ParVector b,
                                    HYPRE_ParVector x)
          Set up BiCGSTAB solver ......................................... 205

6.10.4 int HYPRE_ParCSRBiCGSTABSolve (HYPRE_Solver solver,
                                     HYPRE_ParCSRMatrix A,
                                     HYPRE_ParVector b,
                                     HYPRE_ParVector x)
          Solve the linear system ...................................... 205

6.10.5 int
6.10.6 int HYPRE_ParCSRBiCGSTABSetTol (HYPRE_Solver solver, double tol)

(Optional) Set the convergence tolerance (default is 1e-6) .................

6.10.7 int HYPRE_ParCSRBiCGSTABSetMinIter (HYPRE_Solver solver, int min_iter)

(Optional) Set the minimal number of iterations (default: 0) ...........

6.10.8 int HYPRE_ParCSRBiCGSTABSetMaxIter (HYPRE_Solver solver, int max_iter)

(Optional) Set the maximal number of iterations allowed (default: 1000) .

6.10.9 int HYPRE_ParCSRBiCGSTABSetStopCrit (HYPRE_Solver solver, int stop_crit)

(Optional) If stop_crit = 1, the absolute residual norm is used for the stopping criterion. .........................................................

6.10.10 int HYPRE_ParCSRBiCGSTABSetPrecond (HYPRE_Solver solver, HYPRE_PtrToParSolverFcn precond, HYPRE_PtrToParSolverFcn precond_setup, HYPRE_Solver precond_solver)

(Optional) Set the preconditioner. ...........................................

6.10.11 int HYPRE_ParCSRBiCGSTABGetPrecond (HYPRE_Solver solver, HYPRE_Solver* precond_data)

Get the preconditioner object ..................................................

6.10.12 int HYPRE_ParCSRBiCGSTABSetPrintLevel (HYPRE_Solver solver, int print_level)

(Optional) Set the desired print level. ....................................

6.10.13 int HYPRE_ParCSRBiCGSTABGetNumIterations (HYPRE_Solver solver, int* num_iterations)

Retrieve the number of iterations taken ...................................

6.10.14 int HYPRE_ParCSRBiCGSTABGetFinalRelativeResidualNorm (HYPRE_Solver solver, double* norm)

Retrieve the final relative residual norm .................................
6.10.1

```c
int HYPRE_ParCSRBiCGSTABCreate (MPI_Comm comm, HYPRE_Solver* solver)
```

Create a solver object

6.10.2

```c
int HYPRE_ParCSRBiCGSTABDestroy (HYPRE_Solver solver)
```

Destroy a solver object

6.10.3

```c
int HYPRE_ParCSRBiCGSTABSetup (HYPRE_Solver solver, HYPRE_ParCSRMatrix A, HYPRE_ParVector b, HYPRE_ParVector x)
```

Set up BiCGSTAB solver

6.10.4

```c
int HYPRE_ParCSRBiCGSTABSolve (HYPRE_Solver solver, HYPRE_ParCSRMatrix A, HYPRE_ParVector b, HYPRE_ParVector x)
```

Solve the linear system
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>int HYPRE_ParCSRBiCGSTABSetTol</code> (HYPRE_Solver solver, double tol)</td>
<td>(Optional) Set the convergence tolerance (default is 1e-6)</td>
</tr>
<tr>
<td><code>int HYPRE_ParCSRBiCGSTABSetMinIter</code> (HYPRE_Solver solver, int min_iter)</td>
<td>(Optional) Set the minimal number of iterations (default: 0)</td>
</tr>
<tr>
<td><code>int HYPRE_ParCSRBiCGSTABSetMaxIter</code> (HYPRE_Solver solver, int max_iter)</td>
<td>(Optional) Set the maximal number of iterations allowed (default: 1000)</td>
</tr>
<tr>
<td><code>int HYPRE_ParCSRBiCGSTABSetStopCrit</code> (HYPRE_Solver solver, int stop_crit)</td>
<td>(Optional) If stop_crit = 1, the absolute residual norm is used for the stopping criterion. The default is the relative residual norm (stop_crit = 0).</td>
</tr>
</tbody>
</table>
6.10.9

```c
int HYPRE_ParCSRBiCGSTABSetPrecond (HYPRE_Solver solver,
        HYPRE_PtrToParSolverFcn precond, HYPRE_PtrToParSolverFcn precond_setup,
        HYPRE_Solver precond_solver)
```

(Optional) Set the preconditioner.

6.10.10

```c
int HYPRE_ParCSRBiCGSTABGetPrecond (HYPRE_Solver solver,
        HYPRE_Solver* precond_data)
```

Get the preconditioner object

6.10.11

```c
int HYPRE_ParCSRBiCGSTABSetLogging (HYPRE_Solver solver, int logging)
```

(Optional) Set the amount of logging to be done. The default is 0, i.e. no logging.

6.10.12

```c
int HYPRE_ParCSRBiCGSTABSetPrintLevel (HYPRE_Solver solver, int print_level)
```

(Optional) Set the desired print level. The default is 0, i.e. no printing.
6.10.13

```c
int HYPRE_ParCSRBiCGSTABGetNumIterations (HYPRE_Solver solver, int* num_iterations)
```

Retrieve the number of iterations taken

6.10.14

```c
int HYPRE_ParCSRBiCGSTABGetFinalRelativeResidualNorm (HYPRE_Solver solver, double* norm)
```

Retrieve the final relative residual norm
Krylov Solvers

7.1 Krylov Solvers

7.2 PCG Solver

7.3 GMRES Solver

7.4 BiCGSTAB Solver

7.5 CGNR Solver

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

7.1.1 typedef struct hypre_Solver_struct *HYPRE_Solver

7.1.2 typedef struct hypre_Matrix_struct *HYPRE_Matrix

7.1.3 typedef struct hypre_Vector_struct *HYPRE_Vector

typedef struct hypre_Solver_struct *HYPRE_Solver
Krylov Solvers

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

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, double tol)

(Optional) Set the convergence tolerance

7.2.4 int HYPRE_PCGSetMaxIter (HYPRE_Solver solver, int max_iter)

(Optional) Set maximum number of iterations

7.2.5 int HYPRE_PCGSetTwoNorm (HYPRE_Solver solver, int two_norm)

(Optional) Use the two-norm in stopping criteria

7.2.6 int
7.2.7 int HYPRE_PCGSetPreconditioner (HYPRE_Solver solver, HYPRE_PtrToSolverFcn precond, HYPRE_PtrToSolverFcn precond_setup, HYPRE_Solver precond_solver)

(Optional) Set the preconditioner to use

7.2.8 int HYPRE_PCGSetLogging (HYPRE_Solver solver, int logging)

(Optional) Set the amount of logging to do

7.2.9 int HYPRE_PCGSetPrintLevel (HYPRE_Solver solver, int level)

Return the number of iterations taken

7.2.10 int HYPRE_PCGGetFinalRelativeResidualNorm (HYPRE_Solver solver, double* norm)

Return the norm of the final relative residual

7.2.11 int HYPRE_PCGGetResidual (HYPRE_Solver solver, void** residual)

Return the residual

7.2.12 int HYPRE_PCGGetTol (HYPRE_Solver solver, double* tol)

7.2.13 int HYPRE_PCGGetMaxIter (HYPRE_Solver solver, int* max_iter)

7.2.14 int HYPRE_PCGGetTwoNorm (HYPRE_Solver solver, int* two_norm)

7.2.15 int HYPRE_PCGGetRelativeChange (HYPRE_Solver solver, int* rel_change)

7.2.16 int HYPRE_PCGGetPreconditioner (HYPRE_Solver solver, HYPRE_Solver* precond_data_ptr)

7.2.17 int HYPRE_PCGGetConverged (HYPRE_Solver solver, int* converged)
7.2.1

\textbf{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

\textbf{int HYPRE_PCGSolve} (HYPRE_Solver solver, HYPRE_Matrix A, HYPRE_Vector b, HYPRE_Vector x)

Solve the system

7.2.3

\textbf{int HYPRE_PCGSetTol} (HYPRE_Solver solver, double tol)

(Optional) Set the convergence tolerance

7.2.4

\textbf{int HYPRE_PCGSetMaxIter} (HYPRE_Solver solver, int max_iter)

(Optional) Set maximum number of iterations
### 7.2.5

```c
int HYPRE_PCGSetTwoNorm (HYPRE_Solver solver, int two_norm)
```

(Optional) Use the two-norm in stopping criteria

### 7.2.6

```c
int HYPRE_PCGSetRelChange (HYPRE_Solver solver, int rel_change)
```

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

### 7.2.7

```c
int HYPRE_PCGSetPrecond (HYPRE_Solver solver, HYPRE_PtrToSolverFcn precond, HYPRE_PtrToSolverFcn precond_setup, HYPRE_Solver precond_solver)
```

(Optional) Set the preconditioner to use

### 7.2.8

```c
int HYPRE_PCGSetLogging (HYPRE_Solver solver, int logging)
```

(Optional) Set the amount of logging to do

### 7.2.9

```c
int HYPRE_PCGSetPrintLevel (HYPRE_Solver solver, int level)
```

(Optional) Set the amount of printing to do to the screen
7.2.10

\begin{verbatim}
int HYPRE_PCGGetNumIterations (HYPRE_Solver solver, int* num_iterations)
\end{verbatim}

Return the number of iterations taken

7.2.11

\begin{verbatim}
int HYPRE_PCGGetFinalRelativeResidualNorm (HYPRE_Solver solver, double* norm)
\end{verbatim}

Return the norm of the final relative residual

7.2.12

\begin{verbatim}
int HYPRE_PCGGetResidual (HYPRE_Solver solver, void** residual)
\end{verbatim}

Return the residual

7.2.13

\begin{verbatim}
int HYPRE_PCGGetTol (HYPRE_Solver solver, double* tol)
\end{verbatim}

7.2.14

\begin{verbatim}
int HYPRE_PCGGetMaxIter (HYPRE_Solver solver, int* max_iter)
\end{verbatim}
7.2.15

```c
int HYPRE_PCGGetTwoNorm (HYPRE_Solver solver, int* two_norm)
```

7.2.16

```c
int HYPRE_PCGGetRelChange (HYPRE_Solver solver, int* rel_change)
```

7.2.17

```c
int HYPRE_PCGGetPrecond (HYPRE_Solver solver, HYPRE_Solver* precond_data_ptr)
```

7.2.18

```c
int HYPRE_PCGGetLogging (HYPRE_Solver solver, int* level)
```

7.2.20

```c
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. ........................................ 217

7.3.2 int HYPRE_GMRESsolve (HYPRE_Solver solver, HYPRE_Matrix A, HYPRE_Vector b, HYPRE_Vector x)

Solve the system .................................................. 217

7.3.3 int HYPRE_GMRESsetTol (HYPRE_Solver solver, double tol)

(Optional) Set the convergence tolerance ......................... 218

7.3.4 int HYPRE_GMRESsetMaxIter (HYPRE_Solver solver, int max_iter)

(Optional) Set maximum number of iterations ...................... 218

7.3.5 int HYPRE_GMRESsetKDim (HYPRE_Solver solver, int k_dim)

(Optional) Set the maximum size of the Krylov space ............. 218

7.3.6 int HYPRE_GMRESsetRelChange (HYPRE_Solver solver, int rel_change)

(Optional) Additionally require that the relative difference in successive iterates be small .............................................. 218

7.3.7 int HYPRE_GMRESsetPrecond (HYPRE_Solver solver, HYPRE_PtrToSolverFcn precond, HYPRE_PtrToSolverFcn precond_setup, HYPRE_Solver precond_solver)

(Optional) Set the preconditioner to use ............................ 218

7.3.8 int HYPRE_GMRESsetLogging (HYPRE_Solver solver, int logging)

(Optional) Set the amount of logging to do ....................... 219

7.3.9 int HYPRE_GMRESsetPrintLevel (HYPRE_Solver solver, int level)

(Optional) Set the amount of printing to do to the screen ...... 219

7.3.10 int HYPRE_GMRESgetNumIterations (HYPRE_Solver solver, int* num_iterations)

Return the number of iterations taken ............................ 219

7.3.11 int HYPRE_GMRESgetFinalRelativeResidualNorm (HYPRE_Solver solver, double* norm)

Return the norm of the final relative residual ..................... 219

7.3.12 int
7.3.13 \texttt{HYPRE\_GMRES\_GetResidual (HYPRE\_Solver solver, void** residual)}

Return the residual

7.3.14 \texttt{HYPRE\_GMRES\_GetTol (HYPRE\_Solver solver, double* tol)}

7.3.15 \texttt{HYPRE\_GMRES\_GetKDim (HYPRE\_Solver solver, int* k_dim)}

7.3.16 \texttt{HYPRE\_GMRES\_GetRelChange (HYPRE\_Solver solver, int* rel_change)}

7.3.17 \texttt{HYPRE\_GMRES\_GetPrecond (HYPRE\_Solver solver, HYPRE\_Solver*-precond\_data\_ptr)}

7.3.18 \texttt{HYPRE\_GMRES\_GetLogging (HYPRE\_Solver solver, int* level)}

7.3.19 \texttt{HYPRE\_GMRES\_GetPrintLevel (HYPRE\_Solver solver, int* level)}

7.3.20 \texttt{HYPRE\_GMRES\_GetConverged (HYPRE\_Solver solver, int* converged)}

7.3.1

\begin{verbatim}
int \texttt{HYPRE\_GMRESSetup (HYPRE\_Solver solver, HYPRE\_Matrix A,  
HYPRE\_Vector b, HYPRE\_Vector x)}
\end{verbatim}

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

\begin{verbatim}
int \texttt{HYPRE\_GMRESSolve (HYPRE\_Solver solver, HYPRE\_Matrix A,  
HYPRE\_Vector b, HYPRE\_Vector x)}
\end{verbatim}

Solve the system
### 7.3.3

**int HYPRE_GMRESSetTol** (HYPRE_Solver solver, double tol)

(Optional) Set the convergence tolerance

### 7.3.4

**int HYPRE_GMRESSetMaxIter** (HYPRE_Solver solver, int max_iter)

(Optional) Set maximum number of iterations

### 7.3.5

**int HYPRE_GMRESSetKDim** (HYPRE_Solver solver, int k_dim)

(Optional) Set the maximum size of the Krylov space

### 7.3.6

**int HYPRE_GMRESSetRelChange** (HYPRE_Solver solver, int rel_change)

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

### 7.3.7

**int HYPRE_GMRESSetPrecond** (HYPRE_Solver solver, HYPRE_PtrToSolverFcn precond, HYPRE_PtrToSolverFcn precond_setup, HYPRE_Solver precond_solver)

(Optional) Set the preconditioner to use
7.3.8

int HYPRE_GMRESSetLogging (HYPRE_Solver solver, int logging)

(Optional) Set the amount of logging to do

7.3.9

int HYPRE_GMRESSetPrintLevel (HYPRE_Solver solver, int level)

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

7.3.10

int HYPRE_GMRESGetNumIterations (HYPRE_Solver solver, int* num_iterations)

Return the number of iterations taken

7.3.11

int HYPRE_GMRESGetFinalRelativeResidualNorm (HYPRE_Solver solver, double* norm)

Return the norm of the final relative residual
Return the residual

7.3.13

int HYPRE_GMRESGetTol (HYPRE_Solver solver, double* tol)

7.3.14

int HYPRE_GMRESGetMaxIter (HYPRE_Solver solver, int* max_iter)

7.3.15

int HYPRE_GMRESGetKDim (HYPRE_Solver solver, int* k_dim)

7.3.16

int HYPRE_GMRESGetRelChange (HYPRE_Solver solver, int* rel_change)
7.3.17

```c
int HYPRE_GMRESGetPrecond (HYPRE_Solver solver, HYPRE_Solver* precond_data_ptr)
```

7.3.18

```c
int HYPRE_GMRESGetLogging (HYPRE_Solver solver, int* level)
```

7.3.19

```c
int HYPRE_GMRESGetPrintLevel (HYPRE_Solver solver, int* level)
```

7.3.20

```c
int HYPRE_GMRESGetConverged (HYPRE_Solver solver, int* converged)
```

7.4

### BiCGSTAB Solver

**Names**

7.4.1  
```c
int HYPRE_BiCGSTABSetup (HYPRE_Solver solver, HYPRE_Matrix A, HYPRE_Vector b, HYPRE_Vector x)
```

Prepare to solve the system. ..............................................

7.4.2  
```c
```
7.4.1 int HYPRE_BiCGSTABSetup (HYPRE_Solver solver, HYPRE_Matrix A, HYPRE_Vector b, HYPRE_Vector x)

7.4.3 int HYPRE_BiCGSTABSetTol (HYPRE_Solver solver, double tol)

(Optimal) Set the convergence tolerance

7.4.4 int HYPRE_BiCGSTABSetMaxIter (HYPRE_Solver solver, int max_iter)

(Optional) Set maximum number of iterations

7.4.5 int HYPRE_BiCGSTABSetPrecond (HYPRE_Solver solver, HYPRE_PtrToSolverFcn precond, HYPRE_PtrToSolverFcn precond_setup, HYPRE_Solver precond_solver)

(Optional) Set the preconditioner to use

7.4.6 int HYPRE_BiCGSTABSetLogging (HYPRE_Solver solver, int logging)

(Optional) Set the amount of logging to do

7.4.7 int HYPRE_BiCGSTABSetPrintLevel (HYPRE_Solver solver, int level)

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

7.4.8 int HYPRE_BiCGSTABGetNumIterations (HYPRE_Solver solver, int* num_iterations)

Return the number of iterations taken

7.4.9 int HYPRE_BiCGSTABGetFinalRelativeResidualNorm (HYPRE_Solver solver, double* norm)

Return the norm of the final relative residual

7.4.10 int HYPRE_BiCGSTABGetResidual (HYPRE_Solver solver, void** residual)

Return the residual

7.4.11 int HYPRE_BiCGSTABGetPrecond (HYPRE_Solver solver, HYPRE_Solver* precond_data_ptr)
Prepare to solve the system. The coefficient data in \( \mathbf{b} \) and \( \mathbf{x} \) is ignored here, but information about the layout of the data may be used.

### 7.4.2

```c
int HYPRE_BiCGSTABSolve (HYPRE_Solver solver, HYPRE_Matrix A, HYPRE_Vector b, HYPRE_Vector x)
```

Solve the system

### 7.4.3

```c
int HYPRE_BiCGSTABSetTol (HYPRE_Solver solver, double tol)
```

(Optional) Set the convergence tolerance

### 7.4.4

```c
int HYPRE_BiCGSTABSetMaxIter (HYPRE_Solver solver, int max_iter)
```

(Optional) Set maximum number of iterations

### 7.4.5

```c
int HYPRE_BiCGSTABSetPrecond (HYPRE_Solver solver, HYPRE_PtrToSolverFcn precond, HYPRE_PtrToSolverFcn precond_setup, HYPRE_Solver precond_solver)
```

(Optional) Set the preconditioner to use
### 7.4.6

```c
int HYPRE_BiCGSTABSetLogging (HYPRE_Solver solver, int logging)
```

(Optional) Set the amount of logging to do

### 7.4.7

```c
int HYPRE_BiCGSTABSetPrintLevel (HYPRE_Solver solver, int level)
```

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

### 7.4.8

```c
int HYPRE_BiCGSTABGetNumIterations (HYPRE_Solver solver, int* num_iterations)
```

Return the number of iterations taken

### 7.4.9

```c
int HYPRE_BiCGSTABGetFinalRelativeResidualNorm (HYPRE_Solver solver, double* norm)
```

Return the norm of the final relative residual
### 7.4.10

```c
int HYPRE_BiCGSTABGetResidual (HYPRE_Solver solver, void** residual)
```

Return the residual.

### 7.4.11

```c
int HYPRE_BiCGSTABGetPrecond (HYPRE_Solver solver, HYPRE_Solver* precond_data_ptr)
```

### 7.5

**CGNR Solver**

#### Names

**7.5.1**

```c
int HYPRE_CGNRSetup (HYPRE_Solver solver, HYPRE_Matrix A, 
                     HYPRE_Vector b, HYPRE_Vector x)
```

Prepare to solve the system.

**7.5.2**

```c
int HYPRE_CGNRSolve (HYPRE_Solver solver, HYPRE_Matrix A, 
                     HYPRE_Vector b, HYPRE_Vector x)
```

Solve the system.

**7.5.3**

```c
int HYPRE_CGNRSetTol (HYPRE_Solver solver, double tol)
```

(Optional) Set the convergence tolerance.

**7.5.4**

```c
int HYPRE_CGNRSetMaxIter (HYPRE_Solver solver, int max_iter)
```

(Optional) Set maximum number of iterations.

**7.5.5**

```c
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.
7.5.1

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.

7.5.2

int **HYPRE_CGNRSolve** (HYPRE_Solver solver, HYPRE_Matrix A, HYPRE_Vector b, HYPRE_Vector x)

Solve the system

7.5.3

int **HYPRE_CGNRSetTol** (HYPRE_Solver solver, double tol)
(Optional) Set the convergence tolerance

7.5.4

int HYPRE_CGNRSetMaxIter (HYPRE_Solver solver, int max_iter)

(Optional) Set maximum number of iterations

7.5.5

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

7.5.6

int HYPRE_CGNRSetLogging (HYPRE_Solver solver, int logging)

(Optional) Set the amount of logging to do

7.5.7

int HYPRE_CGNRGetNumIterations (HYPRE_Solver solver, int* num_iterations)

Return the number of iterations taken
7.5.8  

int HYPRE_CGNRGetFinalRelativeResidualNorm (HYPRE_Solver solver, double* norm)

Return the norm of the final relative residual

7.5.9  

int HYPRE_CGNRGetPrecond (HYPRE_Solver solver, HYPRE_Solver* precond_data_ptr)
8 Finite Element Interface

8.1 FEI Functions

8.1.1 LLNL_FEI_Impl (MPI_Comm comm)

Finite element interface constructor: this function creates an instantiation of the HYPRE fei class.

8.1.2 ~LLNL_FEI_Impl ()

Finite element interface destructor: this function destroys the object as well as its internal memory allocations.

8.1.3 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.

8.1.4 int initFields (int numFields, int* fieldSizes, int* fieldIDs)

Each node or element variable has one or more fields.

8.1.5 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.

8.1.6 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.

8.1.7 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. ........................................... 233

8.1.8 int 
initCRMult (int CRListLen, int* CRNodeList, int* CRFieldList, int* CRID)
   This function initializes the Lagrange multiplier constraints ............. 233

8.1.9 int 
initComplete ()
   This function signals to the FEI that the initialization step has been 
   completed. .............................................................. 234

8.1.10 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. .................................. 234

8.1.11 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. .................................. 234

8.1.12 int 
resetRHSVector (double s)
   This function resets the right hand side vector to s. .......................... 235

8.1.13 int 
resetInitialGuess (double s)
   This function resets the solution vector to s. ................................ 235

8.1.14 int 
loadNodeBCs (int nNodes, int* nodeIDs, int fieldID, double** alpha, 
   double** beta, double** gamma)
   This function loads the nodal boundary conditions. .......................... 236

8.1.15 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 ........................ 236

8.1.16 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. ............................................... 236

8.1.17 int 
sumInElemRHS (int elemBlock, int elemID, int* elemConn, 
   double* elemLoad)
   This function adds the element load to the right hand side vector .......... 236

8.1.18 int 
loadComplete ()
   This function signals to the FEI that the loading phase has been completed. 
   ................................................................. 237

8.1.19 int
getNumBlockActNodes (int elemBlockID, int* nNodes)
   This function returns the number of nodes given the element block. ......  

getNumBlockActEqns (int elemBlockID, int* nEqns)
   This function returns the number of unknowns given the element block. .  

getBlockNodeIDList (int elemBlockID, int numNodes, int* nodeIDList)
   This function returns the node identifiers given the element block. ......  

getBlockNodeSolution (int elemBlockID, int numNodes, int* nodeIDList,
   int* solnOffsets, double* solnValues)
   This function returns the nodal solutions given the element block number.  

loadCRMult (int CRID, int CRListLen, int* CRNodeList, int* CRFieldList,
   double* CRWeightList, double CRValue)
   This function loads the Lagrange multiplier constraints  

8.1.1

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

8.1.2

~LLNL_FEI_Impl ()

Finite element interface destructor: this function destroys the object as well as its internal memory allocations.

Parameters: 

   - no parameter needed
### 8.1.3 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

### 8.1.4 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

### 8.1.5 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
- interleaveStrategy - indicates how unknowns are ordered

This page has been automatically generated with DOC++

DOC++ is ©1995 by Roland Wunderling

Malte Zöckler
8.1.6

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

8.1.7

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

8.1.8

int initCRMult (int CRListLen, int* CRNodeList, int* CRFieldList, int* CRID)

This function initializes the Lagrange multiplier constraints
Parameters:

<table>
<thead>
<tr>
<th>CRListLen</th>
<th>the number of constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRNodeList</td>
<td>node identifiers where constraints are applied</td>
</tr>
<tr>
<td>CRFieldList</td>
<td>field identifiers within nodes where constraints are applied</td>
</tr>
<tr>
<td>CRID</td>
<td>the constraint identifier</td>
</tr>
</tbody>
</table>

8.1.9

```c
int initComplete ()
```

This function signals to the FEI that the initialization step has been completed. The loading step will follow.

Parameters:
- no parameter needed

8.1.10

```c
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.

8.1.11

```c
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.
8.1.12

int resetRHSVector (double s)

This function resets the right hand side vector to s.

Parameters:

\textbf{s} - the value each right hand side vector entry is set to.

8.1.13

int resetInitialGuess (double s)

This function resets the solution vector to s.

Parameters:

\textbf{s} - the value each solution vector entry is set to.

8.1.14

int loadNodeBCs (int nNodes, int* nodeIDs, int fieldID, double** alpha, double** beta, double** gamma)

This function loads the nodal boundary conditions. The boundary conditions allowed are of the robin type.

Parameters:

\begin{itemize}
\item \textbf{nNodes} - number of nodes boundary conditions are imposed
\item \textbf{nodeIDs} - nodal identifiers
\item \textbf{fieldID} - field identifier with nodes where BC are imposed
\item \textbf{alpha} - the multipliers for the field
\item \textbf{beta} - the multipliers for the normal derivative of the field
\item \textbf{gamma} - the boundary values on the right hand side of the equations
\end{itemize}
8.1.15

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

8.1.16

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

8.1.17

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

8.1.18

int loadComplete ()

This function signals to the FEI that the loading phase has been completed.

Parameters: - no parameter needed

8.1.19

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

8.1.20

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
### 8.1.21

**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

---

### 8.1.22

**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

---

### 8.1.23

**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)
8.2

FEI Solver Parameters

8.2.1 Preconditioners and Solvers

Here the various options for solvers and preconditioners are defined. .......

8.2.2 BoomerAMG

Parameter options for the algebraic multigrid preconditioner BoomerAMG. ........................................... 239

8.2.3 MLI

Parameter options for the smoothed aggregation preconditioner MLI. .......

8.2.4 Various

Parameter options for ILUT, ParaSails and polynomial preconditioners are defined. ........................................... 240

8.2.5 Matrix Reduction

Parameters which define different reduction modes. ......................... 241

8.2.6 Performance Tuning and Diagnostics

Parameters control diagnostic information, memory use, etc. ............. 242

8.2.7 Miscellaneous

Parameters that are helpful for finite element information. ............... 243

Preconditioners and Solvers

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.

8.2.2 BoomerAMG

Parameter options for the algebraic multigrid preconditioner BoomerAMG.

\textbf{amgMaxLevels} \( xxx \) where \( xxx \) is an integer specifying the maximum number of levels to be used for the grid hierarchy.

\textbf{amgCoarsenType} \( xxx \) where \( xxx \) specifies one of falgout or ruge, or default (CLJP) coarsening for BoomerAMG.

\textbf{amgMeasureType} \( xxx \) where \( xxx \) specifies one of local or global. This parameter affects how coarsening is performed in parallel.

\textbf{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.

\textbf{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.

\textbf{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.

\textbf{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.

\textbf{amgStrongThreshold} \( xxx \) where \( xxx \) is a floating point number between 0 and 1 specifying the threshold used to determine strong coupling in BoomerAMG’s coarsening. The default value is 0.25.

\textbf{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.

---

### 8.2.3 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 'AMGSAce', the parameter 'haveSFEI' has to be sent into the FEI using the parameters function (this option is valid only for the Sandia FEI implementation).

### 8.2.4 Various

Parameter options for ILUT, ParaSails and polynomial preconditioners are defined.

- **euclidNlevels $xxx$** where $xxx$ is a 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.
8.2.5  

Matrix Reduction

Parameters which define different reduction modes.

\textbf{schurReduction} turns on the Schur reduction mode.

\textbf{slideReduction} turns on the slide reduction mode.

\textbf{slideReduction2} turns on the slide reduction mode version 2 (see section 2).

\textbf{slideReduction3} turns on the slide reduction mode version 3 (see section 2).

8.2.6  

Performance Tuning and Diagnostics

Parameters control diagnostic information, memory use, etc.

\textbf{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.

\textbf{setDebug} xxx where xxx is one of \textit{slideReduction1}, \textit{slideReduction2}, \textit{slideReduction3} (level 1,2,3 diagnostics in the slide surface reduction code), \textit{printMat} (print the original matrix into a file), \textit{printReducedMat} (print the reduced matrix into a file), \textit{printSol} (print the solution into a file), \textit{ddilut} (output diagnostic information for DDIlut preconditioner setup), and \textit{amgDebug} (output diagnostic information for AMG).

\textbf{optimizeMemory} cleans up the matrix sparsity pattern after the matrix has been loaded. (It has been kept to allow matrix reuse.)

\textbf{imposeNoBC} turns off the boundary condition to allow diagnosing the matrix (for example, checking the null space.)
Parameters that are helpful for finite element information.

**AConjugateProjection** \( \text{xxx} \) where \( \text{xxx} \) specifies the number of previous solution vectors to keep for the \( A \)-conjugate projection. The default is 0 (the projection is off).

**minResProjection** \( \text{xxx} \) where \( \text{xxx} \) 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.