$hypre$ Reference Manual

— Version 1.6.0 —
## Contents

1. **Struct System Interface — A structured-grid conceptual interface** ........................................... 3  
   1.1 Struct Grids — ........................................... 3  
   1.2 Struct Stencils — ...................................... 4  
   1.3 Struct Matrices — ..................................... 5  
   1.4 Struct Vectors — ...................................... 7  

2. **SStruct System Interface — A semi-structured-grid conceptual interface**  .................. 9  
   2.1 SStruct Grids — ......................................... 9  
   2.2 SStruct Stencils — ..................................... 13  
   2.3 SStruct Graphs — ...................................... 13  
   2.4 SStruct Matrices — ..................................... 14  
   2.5 SStruct Vectors — ...................................... 19  

3. **IJ System Interface — A linear-algebraic conceptual interface**  .............................. 25  
   3.1 IJ Matrices — ........................................... 25  
   3.2 IJ Vectors — ........................................... 30  

4. **Struct Solvers — Linear solvers for structured grids** ......................................................... 35  
   4.1 Struct Solvers — ........................................ 35  
   4.2 Struct Jacobi Solver — ................................. 35  
   4.3 Struct PFMG Solver — ................................. 37  
   4.4 Struct SMG Solver — .................................. 38  
   4.5 Struct PCG Solver — .................................. 40  
   4.6 Struct GMRES Solver — ............................... 41  

5. **SStruct Solvers — Linear solvers for semi-structured grids** .............................................. 43  
   5.1 SStruct Solvers — ....................................... 43  
   5.2 SStruct PCG Solver — .................................. 43  
   5.3 SStruct GMRES Solver — ............................... 45  
   5.4 SStruct SysPFMG Solver — ............................ 46  

6. **ParCSR Solvers — Linear solvers for sparse matrix systems** ............................................. 49  
   6.1 ParCSR Solvers — ....................................... 49  
   6.2 ParCSR BoomerAMG Solver — .......................... 49  
   6.3 ParCSR ParaSails Preconditioner — ...................... 51  
   6.4 ParCSR Euclid Preconditioner — ........................ 56  
   6.5 ParCSR Pihlt Preconditioner — .......................... 59  
   6.6 ParCSR PCG Solver — ................................... 59  
   6.7 ParCSR GMRES Solver — ................................. 61
Struct System Interface

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

Author: Robert D. Falgout

Names
1.1 Struct Grids .................................................. 3
1.2 Struct Stencils .............................................. 4
1.3 Struct Matrices .............................................. 5
1.4 Struct Vectors ............................................... 7

1.1 Struct Grids

Names
typedef struct hypre_StructGrid_struct * HYPRE_StructGrid
    A grid object is constructed out of several “boxes”, defined on a global
abstract index space
int HYPRE_StructGridCreate (MPI_Comm comm, int ndim,
    HYPRE_StructGrid *grid)
    Create an ndim-dimensional grid object

1.1.1 int HYPRE_StructGridDestroy (HYPRE_StructGrid grid)
    Destroy a grid object ........................................ 4
    int
**HYPRE_StructGridSetExtents** (HYPRE_StructGrid grid, int *lower, int *upper)

Set the extents for a box on the grid

int

**HYPRE_StructGridAssemble** (HYPRE_StructGrid grid)

Finalize the construction of the grid before using

int

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

(Optional) Set periodic

1.1.1

int **HYPRE_StructGridDestroy** (HYPRE_StructGrid grid)

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

1.2

**Struct Stencils**

**Names**

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

int

**HYPRE_StructStencilDestroy** (HYPRE_StructStencil stencil)

Destroy a stencil object

1.2.1

int

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

Set a stencil entry ..........................................................
1.2.1

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

Set a stencil entry.

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

1.3

Struct Matrices

Names

typedef struct  

```
typedef struct  

```

The matrix object

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

Create a matrix object

```c
int HYPRE_StructMatrixDestroy (HYPRE_StructMatrix matrix)
```

Destroy a matrix object

```c
int HYPRE_StructMatrixInitialize (HYPRE_StructMatrix matrix)
```

Prepare a matrix object for setting coefficient values

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

Set matrix coefficients index by index

```c
int HYPRE_StructMatrixSetBoxValues (HYPRE_StructMatrix matrix,
                                   int *lower, int *upper, int *entries,
                                   double *values)
```

Set matrix coefficients a box at a time

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

_Add to matrix coefficients index by index_

int

**HYPRE_StructMatrixAddToBoxValues** (HYPRE_StructMatrix matrix,  
int *lower, int *upper,  
int nentries, int *entries,  
double *values)

_Add to matrix coefficients a box at a time_

int

**HYPRE_StructMatrixAssemble** (HYPRE_StructMatrix matrix)

_Finalize the construction of the matrix before using_

1.3.1

int

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

_(Optional) Define symmetry properties of the matrix ................. 6_

1.3.2

int

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

_Print the matrix to file .................................................... 6_

(Optional) 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.

1.3.2

int

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

Print the matrix to file. This is mainly for debugging purposes.
1.4

Struct Vectors

Names

typedef struct  hpre_StructVector  struct*  HYPRE_StructVector  
    The vector object

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

int
HYPRE_StructVectorDestroy  (HYPRE_StructVector vector)
    Destroy a vector object

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

int
HYPRE_StructVectorSetValues  (HYPRE_StructVector vector,  int *index,  
    double value)
    Set vector coefficients index by index

int
HYPRE_StructVectorSetBoxValues  (HYPRE_StructVector vector,  
    int *lower,  int *upper,  
    double *values)
    Set vector coefficients a box at a time

int
HYPRE_StructVectorAddToValues  (HYPRE_StructVector vector,  
    int *index,  double value)
    Set vector coefficients index by index

int
HYPRE_StructVectorAddToBoxValues  (HYPRE_StructVector vector,  
    int *lower,  int *upper,  
    double *values)
    Set vector coefficients a box at a time

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

int
HYPRE_StructVectorGetValues  (HYPRE_StructVector vector,  int *index,  
    double *value)
    Get vector coefficients index by index

int
**HYPRE_StructVectorGetBoxValues** (HYPRE_StructVector vector, 
int *lower, int *upper, 
double *values)

*Get vector coefficients a box at a time*

1.4.1 int

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

*Print the vector to file* ................................. 8

---

1.4.1 int

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

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

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

Author: Robert D. Falgout

Names

2.1 SStruct Grids

2.2 SStruct Stencils

2.3 SStruct Graphs

2.4 SStruct Matrices

2.5 SStruct Vectors

2.1 SStruct Grids

Names

2.1.1 typedef struct  hypre_SStructGrid struct*  HYPRE_SStructGrid
A grid object is constructed out of several structured “parts” and an optional unstructured “part” ................................................................. 10

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 .......................... 11

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

2.1.3 int
### 2.1.4

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

*Describe additional variables that live at a particular index*

### 2.1.5

**HYPRE_SStructGridSetNeighborBox** (HYPRE_SStructGrid grid, int part, int *lower, int *upper, int *nbor_lower, int *nbor_upper, int *index_map)

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

### 2.1.6

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

*Add an unstructured part to the grid*

**HYPRE_SStructGridAssemble** (HYPRE_SStructGrid grid)

*Finalize the construction of the grid before using*

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

*(Optional) Set periodic for a particular part*

### 2.1.1

```c
#define 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

#define 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

```c
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.

2.1.4

```c
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 (→ page 10), and are referenced as such.

2.1.5

```c
int HYPRE_SStructGridSetNeighborBox (HYPRE_SStructGrid grid, int part, int *ilower, int *upper, int nbor_part, int *nbor_ilower, int *nbor_upper, 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 `upper` map directly to the indexes `nbor_ilower` and `nbor_upper`. Although, it is required that indexes increase from `ilower` to `upper`, indexes may increase and/or decrease from `nbor_ilower` to `nbor_upper`.

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) on part `part` map to indexes 1, 2, and 0 on part `nbor_part`, respectively.
2.1.6

```c
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.

2.2

SStruct Stencils

Names

```c
typedef struct hpre_SStructStencil_struct* HYPRE_SStructStencil
    The stencil object
int HYPRE_SStructStencilCreate (int ndim, int size,
        HYPRE_SStructStencil *stencil)
    Create a stencil object for the specified number of spatial dimensions and
    stencil entries
int HYPRE_SStructStencilDestroy (HYPRE_SStructStencil stencil)
    Destroy a stencil object
int HYPRE_SStructStencilSetEntry (HYPRE_SStructStencil stencil, int entry,
        int *offset, int var)
    Set a stencil entry
```

2.3

SStruct Graphs

Names
typedef struct  hypre_SStructGraph_struct*  HYPRE_SStructGraph

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

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

Create a graph object

int  HYPRE_SStructGraphDestroy (HYPRE_SStructGraph graph)

Destroy a graph object

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

2.3.1

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

int  HYPRE_SStructGraphAssemble (HYPRE_SStructGraph graph)

Finalize the construction of the graph before using

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.

2.4

SSStruct Matrices

Names
typedef struct hypre_SStructMatrix_struct* HYPRE_SStructMatrix

The matrix object

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

Create a matrix object

int HYPRE_SStructMatrixDestroy (HYPRE_SStructMatrix matrix)

Destroy a matrix object

int HYPRE_SStructMatrixInitialize (HYPRE_SStructMatrix matrix)

Prepare a matrix object for setting coefficient values

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

Set matrix coefficients index by index ........................................ 16

2.4.2 int
HYPRE_SStructMatrixSetBoxValues (HYPRE_SStructMatrix matrix,
                                  int part, int *ilower, int *ilower,
                                  int var, int nentries, int *entries,
                                  double *values)

Set matrix coefficients a box at a time ..................................... 16

2.4.3 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 ................................ 17

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

Add to matrix coefficients a box at a time ................................ 17

int HYPRE_SStructMatrixAssemble (HYPRE_SStructMatrix matrix)

Finalize the construction of the matrix before using

2.4.5 int
HYPRE_SStructMatrixSetSymmetric (HYPRE_SStructMatrix matrix,
                                  int symmetric)

Define symmetry properties of the matrix ................................. 18

2.4.6 int
HYPRE_SStructMatrixSetObjectType (HYPRE_SStructMatrix matrix,
                                  int type)

Set the storage type of the matrix object to be constructed .......... 18

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

Get a reference to the constructed matrix object .............................. 18

int
HYPRE_SStructMatrixSetComplex (HYPRE_SStructMatrix matrix)

Set the matrix to be complex

2.4.8

int
HYPRE_SStructMatrixPrint (const char *filename,

HYPRE_SStructMatrix matrix, int all)

Print the matrix to file .............................................................. 19

2.4.1

int
HYPRE_SStructMatrixSetValues (HYPRE_SStructMatrix matrix, int part, int *

*index, int *var, int nentries, int *entries, double *values)

Set matrix coefficients index by index.

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 (→ page 16)

2.4.2

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.
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 (→ page 16)

2.4.3

\[
\text{int HYPRE_SStructMatrixAddToValues (HYPRE_SStructMatrix matrix, int part, int *index, int var, int nentries, int *entries, double *values)}
\]

Add to matrix coefficients index by index.

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 (→ page 16)

2.4.4

\[
\text{int HYPRE_SStructMatrixAddToBoxValues (HYPRE_SStructMatrix matrix, int part, int *ilower, int *upper, int var, int nentries, int *entries, double *values)}
\]

Add to matrix 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 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 (→ page 16)

### 2.4.5

```c
int HYPRE_SStructMatrixSetSymmetric (HYPRE_SStructMatrix 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.

### 2.4.6

```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) or HYPRE_PARCSR.

**See Also:**  
HYPRE_SStructMatrixGetObject (→ 2.4.7, page 18)

### 2.4.7

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

See Also: HYPRE_SStructMatrixSetObjectType (→2.4.6, page 18)

```
2.4.8

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

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

2.5

SSStruct Vectors

Names

typedef struct hypre_SStructVector_struct* HYPRE_SStructVector
The vector object

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

int
HYPRE_SStructVectorDestroy (HYPRE_SStructVector vector)
Destroy a vector object

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

2.5.1

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

2.5.2

int
HYPRE_SStructVectorSetBoxValues (HYPRE_SStructVector vector,
    int part, int *ilower, int *ilower,
    int var, double *values)
Set vector coefficients a box at a time ........................................... 21

2.5.3
int
HYPRE_SStructVectorAddToValues (HYPRE_SStructVector vector,
    int part, int *index, int var,
    double *value)
Set vector coefficients index by index ............................................. 21

2.5.4
int
HYPRE_SStructVectorAddToBoxValues (HYPRE_SStructVector vector,
    int part, int *ilower, int *ilower,
    int var, double *values)
Set vector coefficients a box at a time ............................................. 22

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

int
HYPRE_SStructVectorGather (HYPRE_SStructVector vector)
Gather vector data so that efficient GetValues can be done

2.5.5
int
HYPRE_SStructVectorGetValues (HYPRE_SStructVector vector, int part,
    int *index, int var, double *value)
Get vector coefficients index by index ............................................. 22

2.5.6
int
HYPRE_SStructVectorGetBoxValues (HYPRE_SStructVector vector,
    int part, int *ilower, int *ilower,
    int var, double *values)
Get vector coefficients a box at a time ............................................. 23

2.5.7
int
HYPRE_SStructVectorSetObjectType (HYPRE_SStructVector vector,
    int type)
Set the storage type of the vector object to be constructed .................. 23

2.5.8
int
HYPRE_SStructVectorGetObject (HYPRE_SStructVector vector,
    void **object)
Get a reference to the constructed vector object ............................... 23

int
HYPRE_SStructVectorSetComplex (HYPRE_SStructVector vector)
Set the vector to be complex

2.5.9
int
HYPRE_SStructVectorPrint (const char *filename,
    HYPRE_SStructVector vector, int all)
Print the vector to file ................................................................. 24
2.5.1

```c
int HYPRE_SStructVectorSetValue (HYPRE_SStructVector vector, int part, int *index, int var, double *value)
```

Set vector coefficients index by index.

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 value consists of a pair of doubles representing the real and imaginary parts of the complex value.

See Also: HYPRE_SStructVectorSetComplex (→ page 20)

2.5.2

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

Set vector 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 values consists of pairs of doubles representing the real and imaginary parts of each complex value.

See Also: HYPRE_SStructVectorSetComplex (→ page 20)

2.5.3

```c
int HYPRE_SStructVectorAddToValues (HYPRE_SStructVector vector, int part, int *index, int var, double *value)
```
Set vector coefficients index by index.

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 value consists of a pair of doubles representing the real and imaginary parts of the complex value.

See Also: HYPRE_SStructVectorSetComplex (→ page 20)

2.5.4

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

Set vector 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 values consists of pairs of doubles representing the real and imaginary parts of each complex value.

See Also: HYPRE_SStructVectorSetComplex (→ page 20)

2.5.5

```c
int HYPRE_SStructVectorGetValues (HYPRE_SStructVector vector, int part, int *index, int var, double *value)
```

Get vector coefficients index by index.

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 (→ page 20)

#### 2.5.6

```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.

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 (→ page 20)

#### 2.5.7

```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) or HYPRE_PARCSR.

**See Also:** HYPRE_SStructVectorGetObject (→ 2.5.8, page 23)

#### 2.5.8

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

**See Also:** HYPRE_SStructVectorSetObjectType (→2.5.7, page 23)

```c
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 'T' and 'J' in the name are meant to be mnemonic for the traditional matrix notation A[I,J].

Names
3.1   IJ Matrices

3.2   IJ Vectors

3.1   IJ Matrices

Names
typedef struct  hpreIJMatrix_struct*  HYPRE_IJMatrix
The matrix object

3.1.1   int
HYPRE_IJMatrixCreate (MPI_Comm comm, int ilower, int iupper,
    int jlower, int jupper, HYPRE_IJMatrix *matrix)
Create a matrix object ............................... 26

3.1.2   int
HYPRE_IJMatrixDestroy (HYPRE_IJMatrix matrix)
Destroy a matrix object ............................... 27

3.1.3   int
HYPRE_IJMatrixInitialize (HYPRE_IJMatrix matrix)
Prepare a matrix object for setting coefficient values ............................... 27

3.1.4   int
HYPRE_IJMatrixSetValues (HYPRE_IJMatrix matrix, int nrows, int *ncols,
    const int *rows, const int *cols,
    const double *values)
Sets values for n rows of the matrix ............................... 27

3.1.5   int
3.1.1

int

**HYPRE_JJMatrixCreate** (MPI_Comm comm, int ilower, int iupper, int jlower,
int jupper, HYPRE_JJMatrix *matrix)
Create a matrix object. Each process owns some unique consecutive range of rows, indicated by the global row indices $i\text{lower}$ and $i\text{upper}$. The row data is required to be such that the value of $i\text{lower}$ on any process $p$ be exactly one more than the value of $i\text{upper}$ 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, $j\text{lower}$ and $j\text{upper}$ typically should match $i\text{lower}$ and $i\text{upper}$, respectively. For rectangular matrices, $j\text{lower}$ and $j\text{upper}$ 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 $j\text{lower}$ and $j\text{upper}$ to store the diagonal blocks (rectangular in general) of the matrix separately from the rest of the matrix.

Collective.

### 3.1.2

```c
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.

### 3.1.3

```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.

### 3.1.4

```c
int HYPRE_IJMatrixSetValue (HYPRE_IJMatrix matrix, int nrows, int *ncols,
                          const int *rows, const int *cols, const double *values)
```
Sets values for \texttt{mrows} of the matrix. The arrays \texttt{ncols} and \texttt{rows} are of dimension \texttt{mrows} and contain the number of columns in each row and the row indices, respectively. The array \texttt{cols} contains the column indices for each of the \texttt{rows}, and is ordered by rows. The data in the \texttt{values} array corresponds directly to the column entries in \texttt{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.

### 3.1.5

\begin{verbatim}
int HYPRE_IJMatrixAddToValues (HYPRE_IJMatrix matrix, int mrows, int *
ncols, const int *rows, const int *cols, const double *values)
\end{verbatim}

Adds to values for \texttt{mrows} of the matrix. Usage details are analogous to \texttt{HYPRE_IJMatrixSetValues} (\textit{\ref{3.1.4, page 27}}). Adds to any previous values at the specified locations, or, if there was no value there before, inserts a new one.

Not collective.

### 3.1.6

\begin{verbatim}
int HYPRE_IJMatrixGetValues (HYPRE_IJMatrix matrix, int mrows, int *
ncols, int *rows, int *cols, double *values)
\end{verbatim}

Gets values for \texttt{mrows} of the matrix. Usage details are analogous to \texttt{HYPRE_IJMatrixSetValues} (\textit{\ref{3.1.4, page 27}}).

### 3.1.7

\begin{verbatim}
int HYPRE_IJMatrixSetObjectType (HYPRE_IJMatrix matrix, int type)
\end{verbatim}

Set the storage type of the matrix object to be constructed. Currently, \texttt{type} can only be \texttt{HYPRE_PARCSR}.
Not collective, but must be the same on all processes.

See Also:  
HYPRE_IJMatrixGetObject (→3.1.8, page 29)

3.1.8

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

Get a reference to the constructed matrix object.

See Also:  
HYPRE_IJMatrixSetObjectType (→3.1.7, page 28)

3.1.9

```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.

3.1.10

```c
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.

3.1.11

```c
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.

3.1.12

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

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

3.2

**IJ Vectors**

Names

typedef struct  

`HYPRE_IJVector`  

*The vector object*

3.2.1

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

*Create a vector object* ................................................. 31

3.2.2

```c
int HYPRE_IJVectorDestroy (HYPRE_IJVector vector)
```

*Destroy a vector object* ............................................. 32

3.2.3

```c
int
```
3.2.1

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

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

Collective.

3.2.2

```c
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.

3.2.3

```c
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.

3.2.4

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

Sets values in vector. The arrays `values` and `indices` are of dimension `nvalues` and contain the vector values to be set and the corresponding global vector indices, respectively. Erases any previous values at the specified locations and replaces them with new ones.

Not collective.
3.2.5

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

Adds to values in vector. Usage details are analogous to HYPRE_IJVectorSetValues (→3.2.4, page 32).
Not collective.

3.2.6

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

Gets values in vector. Usage details are analogous to HYPRE_IJVectorSetValues (→3.2.4, page 32).
Not collective.

3.2.7

```c
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 (→3.2.8, page 34)
3.2.8

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

Get a reference to the constructed vector object.

**See Also:** HYPRE_IJVectorSetObjectType (→3.2.7, page 33)

3.2.9

```c
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.

3.2.10

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

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

4.2 Struct Jacobi Solver
Names

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

Create a solver object
```

4.2.1

```c
int HYPRE_StructJacobiDestroy (HYPRE_StructSolver solver)

Destroy a solver object
```

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

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

Solve the system
```

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

(Optional) Set the convergence tolerance
```

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

(Optional) Set maximum number of iterations
```

```c
int HYPRE_StructJacobiSetZeroGuess (HYPRE_StructSolver solver)

(Optional) Use a zero initial guess
```

```c
int HYPRE_StructJacobiSetNonZeroGuess (HYPRE_StructSolver solver)

(Optional) Use a nonzero initial guess
```

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

Return the number of iterations taken
```

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

Return the norm of the final relative residual
```

4.2.1

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

4.3 Struct PFMG Solver

Names

int HYPRE_StructPFMGCreate (MPI_Comm comm,
                              HYPRE_StructSolver *solver)

Create a solver object

int HYPRE_StructPFMGDestroy (HYPRE_StructSolver solver)

Destroy a solver object

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

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

Solve the system

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

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

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

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

int

HYPRE_StructPFMGSetNonZeroGuess (HYPRE_StructSolver solver)
(Optional) Use a non-zero initial guess

int

HYPRE_StructPFMGSetRelaxType (HYPRE_StructSolver solver, int relaxType)
(Optional) Set relaxation type

int

HYPRE_StructPFMGSetNumPreRelax (HYPRE_StructSolver solver, int numPreRelax)
(Optional) Set number of pre-relaxation sweeps

int

HYPRE_StructPFMGSetNumPostRelax (HYPRE_StructSolver solver, int numPostRelax)
(Optional) Set number of post-relaxation sweeps

int

HYPRE_StructPFMGSkipRelax (HYPRE_StructSolver solver, int skipRelax)
(Optional) Skip relaxation on certain grids for isotropic problems

int

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

int

HYPRE_StructPFMGetNumIterations (HYPRE_StructSolver solver, int *numIterations)
Return the number of iterations taken

int

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

4.4

Struct SMG Solver

Names

int

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

int
\textbf{HYPRE_StructSMGDestroy} (HYPRE_StructSolver solver)

Destroy a solver object

\textbf{int HYPRE_StructSMGSetup} (HYPRE_StructSolver solver, 
\hspace{1em} HYPRE_StructMatrix A, 
\hspace{1em} HYPRE_StructVector b, HYPRE_StructVector x)

\textbf{int HYPRE_StructSMGSolve} (HYPRE_StructSolver solver, 
\hspace{1em} HYPRE_StructMatrix A, HYPRE_StructVector b, 
\hspace{1em} HYPRE_StructVector x)

\textit{Solve the system}

\textbf{int HYPRE_StructSMGSetTol} (HYPRE_StructSolver solver, double tol)

(Optional) Set the convergence tolerance

\textbf{int HYPRE_StructSMGSetMaxIter} (HYPRE_StructSolver solver, int max_iter)

(Optional) Set maximum number of iterations

\textbf{int HYPRE_StructSMGSetRelChange} (HYPRE_StructSolver solver, 
\hspace{1em} int rel_change)

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

\textbf{int HYPRE_StructSMGSetZeroGuess} (HYPRE_StructSolver solver)

(Optional) Use a zero initial guess

\textbf{int HYPRE_StructSMGSetNonZeroGuess} (HYPRE_StructSolver solver)

(Optional) Use a nonzero initial guess

\textbf{int HYPRE_StructSMGSetNumPreRelax} (HYPRE_StructSolver solver, 
\hspace{1em} int num_pre_relax)

(Optional) Set number of pre-relaxation sweeps

\textbf{int HYPRE_StructSMGSetNumPostRelax} (HYPRE_StructSolver solver, 
\hspace{1em} int num_post_relax)

(Optional) Set number of post-relaxation sweeps

\textbf{int HYPRE_StructSMGSetLogging} (HYPRE_StructSolver solver, int logging)

(Optional) Set the amount of logging to do

\textbf{int HYPRE_StructSMGGetNumIterations} (HYPRE_StructSolver solver, 
\hspace{1em} int *num_iterations)

Return the number of iterations taken

\textbf{int HYPRE_StructSMGGetFinalRelativeResidualNorm} (HYPRE_StructSolver solver, 
\hspace{1em} double *norm)

Return the norm of the final relative residual
4.5 Struct PCG Solver

Names

int
HYPRE_StructPCGCreate (MPLComm comm, 
                     HYPRE_StructSolver *solver)

Create a solver object

int
HYPRE_StructPCGDestroy (HYPRE_StructSolver solver)

Destroy a solver object

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

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

Solve the system

int
HYPRE_StructPCGSetTol (HYPRE_StructSolver solver, double tol)

(Optional) Set the convergence tolerance

int
HYPRE_StructPCGSetMaxIter (HYPRE_StructSolver solver, int max_iter)

(Optional) Set maximum number of iterations

int
HYPRE_StructPCGSetTwoNorm (HYPRE_StructSolver solver, 
                            int two_norm)

(Optional) Use the two-norm in stopping criteria

int
HYPRE_StructPCGSetRelChange (HYPRE_StructSolver solver, 
                             int rel_change)

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

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

(Optional) Set the preconditioner to use

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

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

int

**HYPRE_StructPCGGetNumIterations** (HYPRE_StructSolver solver, int *num_iterations)

*Return the number of iterations taken*

int

**HYPRE_StructPCGGetFinalRelativeResidualNorm** (HYPRE_StructSolver solver, double *norm)

*Return the norm of the final relative residual*

int

**HYPRE_StructDiagScaleSetup** (HYPRE_StructSolver solver,

HYPRE_StructMatrix A,

HYPRE_StructVector y,

HYPRE_StructVector x)

*Setup routine for diagonal preconditioning*

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

int

**HYPRE_StructGMRESCreate** (MPI_Comm comm,

HYPRE_StructSolver *solver)

*Create a solver object*

int

**HYPRE_StructGMRESDestroy** (HYPRE_StructSolver solver)

*Destroy a solver object*

int

**HYPRE_StructGMRESSetup** (HYPRE_StructSolver solver,

HYPRE_StructMatrix A,

HYPRE_StructVector b,

HYPRE_StructVector x)

*set up*

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

Solve the system

int
HYPRE_StructGMRESSetTol ( HYPRE_StructSolver solver, double tol )

(Optional) Set the convergence tolerance

int
HYPRE_StructGMRESSetMaxIter ( HYPRE_StructSolver solver,
                             int max_iter )

(Optional) Set maximum number of iterations

int
HYPRE_StructGMRESSetPrecond ( HYPRE_StructSolver solver,
                               HYPRE_PtrToStructSolverFcn preconditioner,
                               HYPRE_PtrToStructSolverFcn preconditionerSetup,
                               HYPRE_StructSolver preconditioner )

(Optional) Set the preconditioner to use

int
HYPRE_StructGMRESSetLogging ( HYPRE_StructSolver solver,
                               int logging )

(Optional) Set the amount of logging to do

int
HYPRE_StructGMRESSetNumIterations ( HYPRE_StructSolver solver,
                                     int *numIterations )

Return the number of iterations taken

int
HYPRE_StructGMRESSetFinalRelativeResidualNorm ( HYPRE_StructSolver solver,
                                               double *norm )

Return the norm of the final relative residual
These solvers use matrix/vector storage schemes that are tailored to semi-structured grid problems.

Names

5.1 SStruct Solvers

5.2 SStruct PCG Solver

5.3 SStruct GMRES Solver

5.4 SStruct SysPFMG Solver

5.1 SStruct Solvers

define struct  hypre_SStructSolver struct*  HYPRE_SStructSolver

The solver object

5.2 SStruct PCG Solver

Names

int  HYPRE_SStructPCGCreate (MPI_Comm comm,
HYPRE_SStructSolver *solver)

Create a solver object

5.2.1 int
**HYPER_SStructPCGDestroy** (HYPER_SStructSolver solver)

*Destroy a solver object* .......................................................... 45

int

**HYPER_SStructPCGSetup** (HYPER_SStructSolver solver,
HYPER_SStructMatrix A,
HYPER_SStructVector b,
HYPER_SStructVector x)

*Solve the system*

int

**HYPER_SStructPCGSetTol** (HYPER_SStructSolver solver, double tol)

*(Optional) Set the convergence tolerance*

int

**HYPER_SStructPCGSetMaxIter** (HYPER_SStructSolver solver,
int max_iter)

*(Optional) Set maximum number of iterations*

int

**HYPER_SStructPCGSetTwoNorm** (HYPER_SStructSolver solver,
int two_norm)

*(Optional) Set type of norm to use in stopping criteria*

int

**HYPER_SStructPCGSetRelChange** (HYPER_SStructSolver solver,
int rel_change)

*(Optional) Set to use additional relative-change convergence test*

int

**HYPER_SStructPCGSetPrecond** (HYPER_SStructSolver solver,
HYPER_PtrToSStructSolverFctn precond,
HYPER_PtrToSStructSolverFctn precond_setup, void *precond_solver)

*(Optional) Set the preconditioner to use*

int

**HYPER_SStructPCGSetLogging** (HYPER_SStructSolver solver, int logging)

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

int

**HYPER_SStructPCGGetNumIterations** (HYPER_SStructSolver solver,
int *num_iterations)

*Return the number of iterations taken*

int

**HYPER_SStructPCGGetFinalRelativeResidualNorm** (HYPER_SStructSolver solver,
double *norm)

*Return the norm of the final relative residual*
5.2.1

```c
int HYPRE_SStructPCGDestroy (HYPRE_SStructSolver solver)
```

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

5.3

**SSStruct GMRES Solver**

### Names

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

*Create a solver object*

5.3.1

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

*Destroy a solver object*

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

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

*Solve the system*

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

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

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

*(Optional) Set the convergence tolerance*
**HYPRE_SStructGMRESSetMaxIter** (HYPRE_SStructSolver solver, int max_iter)

*(Optional) Set maximum number of iterations*

**HYPRE_SStructGMRESSetPrecond** (HYPRE_SStructSolver solver, HYPRE_PtrToSStructSolverFcn preconditioner, HYPRE_PtrToSStructSolverFcn preconditioner_setup, void *preconditioner)

*(Optional) Set the preconditioner to use*

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

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

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

*Return the number of iterations taken*

**HYPRE_SStructGMRESGetFinalRelativeResidualNorm** (HYPRE_SStructSolver solver, double *norm)

*Return the norm of the final relative residual*

### 5.3.1

**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.

### 5.4

**SStruct SysPFMG Solver**

**Names**
int HYPRE_SStructSysPFMGCreate ( MPI_Comm comm,
                                  HYPRE_SStructSolver *solver )

Create a solver object

int HYPRE_SStructSysPFMGDestroy ( HYPRE_SStructSolver solver )

Destroy a solver object

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

int HYPRE_SStructSysPFMG Solve ( HYPRE_SStructSolver solver,
                                 HYPRE_SStructMatrix A,
                                 HYPRE_SStructVector b,
                                 HYPRE_SStructVector x)

Solve the system

int HYPRE_SStructSysPFMGSetTol ( HYPRE_SStructSolver solver,
                                 double tol )

(Optional) Set the convergence tolerance

int HYPRE_SStructSysPFMGSetMaxIter ( HYPRE_SStructSolver solver,
                                      int maxIter )

(Optional) Set maximum number of iterations

int HYPRE_SStructSysPFMGSetRelChange ( HYPRE_SStructSolver solver,
                                       int relChange )

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

int HYPRE_SStructSysPFMGSetZeroGuess ( HYPRE_SStructSolver solver )

(Optional) Use a zero initial guess

int HYPRE_SStructSysPFMGSetNonZeroGuess ( HYPRE_SStructSolver solver )

(Optional) Use a non-zero initial guess

int HYPRE_SStructSysPFMGSetRelaxType ( HYPRE_SStructSolver solver,
                                       int relaxType )

(Optional) Set relaxation type

int HYPRE_SStructSysPFMGSetNumPreRelax ( HYPRE_SStructSolver solver,
                                         int numPreRelax )

(Optional) Set number of pre-relaxation sweeps

int
**HYPRE_SStructSysPFMGSNumPostRelax** (HYPRE_SStructSolver solver, int numPostRelax)

*(Optional) Set number of post-relaxation sweeps*

int

**HYPRE_SStructSysPFMGSkipRelax** (HYPRE_SStructSolver solver, int skipRelax)

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

int

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

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

int

**HYPRE_SStructSysPFMGSNumIterations** (HYPRE_SStructSolver solver, int *numIterations)

*Return the number of iterations taken*

int

**HYPRE_SStructSysPFMGSFinalRelativeResidualNorm** (HYPRE_SStructSolver solver, double *norm)

*Return the norm of the final relative residual*
ParCSR Solvers

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

Names

6.1 ParCSR Solvers

6.2 ParCSR BoomerAMG Solver

6.3 ParCSR ParaSails Preconditioner

6.4 ParCSR Euclid Preconditioner

6.5 ParCSR Pilut Preconditioner

6.6 ParCSR PCG Solver

6.7 ParCSR GMRES Solver

---

# define HYPRE_SOLVER_STRUCT

The solver object

---

6.2

ParCSR BoomerAMG Solver
Names

int HYPRE_BoomerAMGCreate (HYPRE_Solver *solver)
   \textit{Create a solver object}

int HYPRE_BoomerAMGDestroy (HYPRE_Solver solver)
   \textit{Destroy a solver object}

int HYPRE_BoomerAMGSetup (HYPRE_Solver solver,
                           HYPRE_ParCSRMatrix A,
                           HYPRE_ParVector b, HYPRE_ParVector x)

int HYPRE_BoomerAMGSolve (HYPRE_Solver solver,
                           HYPRE_ParCSRMatrix A,
                           HYPRE_ParVector b, HYPRE_ParVector x)
   \textit{Solve the system}

int HYPRE_BoomerAMGSetTol (HYPRE_Solver solver, double tol)
   (Optional) Set the convergence tolerance

int HYPRE_BoomerAMGSetMaxIter (HYPRE_Solver solver, int max_iter)
   (Optional) Set maximum number of iterations

int HYPRE_BoomerAMGSetMaxLevels (HYPRE_Solver solver, int max_levels)
   (Optional) Set maximum number of multigrid levels

int HYPRE_BoomerAMGSetStrongThreshold (HYPRE_Solver solver,
                                       double strong_threshold)
   (Optional) Set AMG strength threshold

int HYPRE_BoomerAMGSetMaxRowSum (HYPRE_Solver solver,
                                 double max_row_sum)
   (Optional)

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

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

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

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

int
HYPRE_BoomerAMGSetGridRelaxType (HYPRE_Solver solver,
   int *grid_relax_type)
   (Optional)

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

int
HYPRE_BoomerAMGSetRelaxWeight (HYPRE_Solver solver,
   double *relax_weight)
   (Optional)

int
HYPRE_BoomerAMGSetOutDat (HYPRE_Solver solver, int ioutdat)
   (Optional)

int
HYPRE_BoomerAMGSetDebugFlag (HYPRE_Solver solver, int debug_flag)
   (Optional)

int
HYPRE_BoomerAMGGetNumIterations (HYPRE_Solver solver,
   int *num_iterations)

Return the number of iterations taken

int
HYPRE_BoomerAMGGetFinalRelativeResidualNorm (HYPRE_Solver
   solver, double
   *rel_resid_norm)

Return the norm of the final relative residual

6.3

ParCSR ParaSails Preconditioner

Parallel sparse approximate inverse preconditioner for the ParCSR matrix format.

Names

int
HYPRE_ParaSailsCreate (MPI_Comm comm, HYPRE_Solver *solver)

Create a ParaSails preconditioner

int

HYPRE_ParaSailsDestroy (HYPRE_Solver solver)

Destroy a ParaSails preconditioner

6.3.1

int

HYPRE_ParaSailsSetup (HYPRE_Solver solver, HYPRE_ParCSRMatrix A,
HYPRE_ParVector b, HYPRE_ParVector x)

Set up the ParaSails preconditioner

52

6.3.2

int

HYPRE_ParaSailsSolve (HYPRE_Solver solver, HYPRE_ParCSRMatrix A,
HYPRE_ParVector b, HYPRE_ParVector x)

Apply the ParaSails preconditioner

53

6.3.3

int

HYPRE_ParaSailsSetParams (HYPRE_Solver solver, double thresh,
int levels)

Set the threshold and levels parameter for the ParaSails preconditioner

53

6.3.4

int

HYPRE_ParaSailsSetFilter (HYPRE_Solver solver, double filter)

Set the filter parameter for the ParaSails preconditioner

54

6.3.5

int

HYPRE_ParaSailsSetSym (HYPRE_Solver solver, int sym)

Set the symmetry parameter for the ParaSails preconditioner

54

6.3.6

int

HYPRE_ParaSailsSetLoadbal (HYPRE_Solver solver, double loadbal)

Set the load balance parameter for the ParaSails preconditioner

54

6.3.7

int

HYPRE_ParaSailsSetReuse (HYPRE_Solver solver, int reuse)

Set the pattern reuse parameter for the ParaSails preconditioner

55

6.3.8

int

HYPRE_ParaSailsSetLogging (HYPRE_Solver solver, int logging)

Set the logging parameter for the ParaSails preconditioner

55

6.3.1

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.

6.3.2

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.

6.3.3

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.
levels — [IN] Value of levels parameter, 0 ≤ levels. The default value is 1.
6.3.4

```c
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.

6.3.5

```c
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:

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>nonsymmetric and/or indefinite problem, and nonsymmetric preconditioner</td>
</tr>
<tr>
<td>1</td>
<td>SPD problem, and SPD (factored) preconditioner</td>
</tr>
<tr>
<td>2</td>
<td>nonsymmetric, definite problem, and SPD (factored) preconditioner</td>
</tr>
</tbody>
</table>

6.3.6

```c
int HYPRE_ParaSailsSetLoadbal (HYPRE_Solver solver, double loadbal)
```

Set the load balance parameter for the ParaSails preconditioner.
Parameters:

- **solver** — [IN] Preconditioner object for which to set the load balance parameter.
- **loadbal** — [IN] Value of the load balance parameter, $0 \leq \text{loadbal} \leq 1$. 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.

### 6.3.7

```c
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.

### 6.3.8

```c
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.
6.4

ParCSR Euclid Preconditioner

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>

Names

```c
int HYPRE_EuclidCreate (MPI_Comm comm, HYPRE_Solver *solver)
Create a Euclid object

int HYPRE_EuclidDestroy (HYPRE_Solver solver)
Destroy a Euclid object
```

6.4.1

```c
int HYPRE_EuclidSetup (HYPRE_Solver solver, HYPRE_ParCSRMatrix A, HYPRE_ParVector b, HYPRE_ParVector x)
Set up the Euclid preconditioner ........................................... 57
```

6.4.2

```c
int HYPRE_EuclidSolve (HYPRE_Solver solver, HYPRE_ParCSRMatrix A, HYPRE_ParVector b, HYPRE_ParVector x)
Apply the Euclid preconditioner ........................................... 57
```

6.4.3

```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) .................................. 57
```

6.4.4

```c
int HYPRE_EuclidSetParam (HYPRE_Solver solver, char *name, char *value)
Insert a single (name, value) pair in Euclid's options database ........... 58
```

6.4.5

```c
int HYPRE_EuclidSetParamsFromFile (HYPRE_Solver solver, char *filename)
Insert (name, value) pairs in Euclid's options database .................. 58
```
6.4.1

```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.

6.4.2

```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.

6.4.3

```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_EuclidSetParam`, `HYPRE_EuclidSetParamsFromFile`. 
6.4.4

int HYPRE_EuclidSetParam (HYPRE_Solver solver, char *name, char *value)

Insert a single (name, value) pair in Euclid's options database. If the (name, value) pair already exists, this call updates the value. See also: HYPRE_EuclidSetParams, HYPRE_EuclidSetParamsFromFile.

6.4.5

int HYPRE_EuclidSetParamsFromFile (HYPRE_Solver solver, char *filename)

Insert (name, value) pairs in Euclid's options database. Each line of the file should either begin with a "#," indicating a comment line, or contain a (name value) pair, e.g:

```
>cat optionsFile
#sample runtime parameter file
-blockJacobi 3
-matFile /home/hysom/myfile.euclid
-doSomething true
-xx_coeff -1.0
```

See also: HYPRE_EuclidSetParams, HYPRE_EuclidSetParams.

Parameters: filename [IN] — Pathname/filename to read
6.5

ParCSR Pilut Preconditioner

Names

int 
HYPRE_ParCSR_Pilut_Create (MPI_Comm comm, HYPRE_Solver *solver)

   Create a preconditioner object

int 
HYPRE_ParCSR_Pilut_Destroy (HYPRE_Solver solver)

   Destroy a preconditioner object

int 
HYPRE_ParCSR_Pilut_Setup (HYPRE_Solver solver,
   HYPRE_ParCSR_Matrix A,
   HYPRE_ParVector b, HYPRE_ParVector x)

int 
HYPRE_ParCSR_Pilut_Solve (HYPRE_Solver solver,
   HYPRE_ParCSR_Matrix A,
   HYPRE_ParVector b, HYPRE_ParVector x)

   Precondition the system

int 
HYPRE_ParCSR_Pilut_Set_Max_It (HYPRE_Solver solver, int max_iter)

   (Optional) Set maximum number of iterations

int 
HYPRE_ParCSR_Pilut_Set_Drop_Tolerance (HYPRE_Solver solver, double tol)

   (Optional)

int 
HYPRE_ParCSR_Pilut_Set_Factor_Row_Size (HYPRE_Solver solver, int size)

   (Optional)

6.6

ParCSR PCG Solver

Names

int 
HYPRE_ParCSR_PCG_Create (MPI_Comm comm, HYPRE_Solver *solver)

   Create a solver object

int
HYPRE\_ParCSRPCG\_Destroy (HYPRE\_Solver solver)

Destroy a solver object

int

HYPRE\_ParCSRPCG\_Setup (HYPRE\_Solver solver,
HYPRE\_ParCSRMatrix A,
HYPRE\_ParVector b, HYPRE\_ParVector x)

int

HYPRE\_ParCSRPCG\_Solve (HYPRE\_Solver solver,
HYPRE\_ParCSRMatrix A,
HYPRE\_ParVector b, HYPRE\_ParVector x)

Solve the system

int

HYPRE\_ParCSRPCG\_SetTol (HYPRE\_Solver solver, double tol)

(Optional) Set the convergence tolerance

int

HYPRE\_ParCSRPCG\_SetMaxIter (HYPRE\_Solver solver, int max\_iter)

(Optional) Set maximum number of iterations

int

HYPRE\_ParCSRPCG\_SetTwoNorm (HYPRE\_Solver solver, int two\_norm)

(Optional) Use the two-norm in stopping criteria

int

HYPRE\_ParCSRPCG\_SetRelChange (HYPRE\_Solver solver, int rel\_change)

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

int

HYPRE\_ParCSRPCG\_SetPrecond (HYPRE\_Solver solver,
HYPRE\_PtrToParSolverFcn precond,
HYPRE\_PtrToParSolverFcn precond\_setup,
HYPRE\_Solver precond\_solver)

(Optional) Set the preconditioner to use

int

HYPRE\_ParCSRPCG\_GetPrecond (HYPRE\_Solver solver,
HYPRE\_Solver \*precond\_data)

int

HYPRE\_ParCSRPCG\_SetLogging (HYPRE\_Solver solver, int logging)

(Optional) Set the amount of logging to do

int

HYPRE\_ParCSRPCG\_GetNumIterations (HYPRE\_Solver solver,
int \*num\_iterations)

Return the number of iterations taken

int

HYPRE\_ParCSRPCG\_GetFinalRelativeResidualNorm (HYPRE\_Solver solver,
double \*norm)

Return the norm of the final relative residual

int
HYPRE_ParCSRDiagScaleSetup (HYPRE_Solver solver,
HYPRE_ParCSRMatrix A,
HYPRE_ParVector y,
HYPRE_ParVector x)

Setup routine for diagonal preconditioning

int
HYPRE_ParCSRDiagScale (HYPRE_Solver solver,
HYPRE_ParCSRMatrix HA,
HYPRE_ParVector Hy, HYPRE_ParVector Hx)

Solve routine for diagonal preconditioning

6.7

ParCSR GMRES Solver

Names

int
HYPRE_ParCSRGMRESCreate (MPI_Comm comm,
HYPRE_Solver *solver)

Create a solver object

int
HYPRE_ParCSRGMRESDestroy (HYPRE_Solver solver)

Destroy a solver object

int
HYPRE_ParCSRGMRESSetup (HYPRE_Solver solver,
HYPRE_ParCSRMatrix A,
HYPRE_ParVector b,
HYPRE_ParVector x)

int
HYPRE_ParCSRGMRESSolve (HYPRE_Solver solver,
HYPRE_ParCSRMatrix A,
HYPRE_ParVector b, HYPRE_ParVector x)

Solve the system

int
HYPRE_ParCSRGMRESSetKDim (HYPRE_Solver solver, int kdim)

(Optional) Set the maximum size of the Krylov space

int
HYPRE_ParCSRGMRESSetTol (HYPRE_Solver solver, double tol)

(Optional) Set the convergence tolerance

int
HYPRE_ParCSRGMRESSetMaxIter (HYPRE_Solver solver, int maxIter)

(Optional) Set maximum number of iterations

int
**HYPRE_ParCSRGMRESSetPrecond** (HYPRE_Solver solver,
    HYPRE_PtrToParSolverFcn precond,
    HYPRE_PtrToParSolverFcn
    precond_setup,
    HYPRE_Solver precond_solver)

*(Optional) Set the preconditioner to use*

int
**HYPRE_ParCSRGMRESSetPrecond** (HYPRE_Solver solver,
    HYPRE_Solver *precond_data)

int
**HYPRE_ParCSRGMRESSetLogging** (HYPRE_Solver solver, int logging)

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

int
**HYPRE_ParCSRGMRESSetNumIterations** (HYPRE_Solver solver,
    int *num_iterations)

*Return the number of iterations taken*

int
**HYPRE_ParCSRGMRESSetFinalRelativeResidualNorm** (HYPRE_Solver solver,
    double *norm)

*Return the norm of the final relative residual*