LoRaSp

Generated by Doxygen 1.6.1

Mon Jun 8 10:15:23 2015

Contents

1 Class Index **1**

1.1 Class Hierarchy 1

2 Class Index **3**

2.1 Class List 3

3 Class Documentation **5**

3.1 blackNode Class Reference 5

3.1.1 Detailed Description 6

3.1.2 Constructor & Destructor Documentation 6

3.1.2.1 blackNode 6

3.1.3 Member Function Documentation 6

3.1.3.1 mergeChildren 6

3.1.3.2 schurComp 6

3.2 diagPC Class Reference 8

3.2.1 Detailed Description 8

3.2.2 Constructor & Destructor Documentation 8

3.2.2.1 diagPC 8

3.3 edge Class Reference 9

3.3.1 Detailed Description 9

3.3.2 Constructor & Destructor Documentation 9

3.3.2.1 edge 9

3.3.3 Member Function Documentation 10

3.3.3.1 compress 10

3.3.3.2 isEliminated 10

3.3.3.3 isWellSeparated 10

3.3.4 Member Data Documentation 10

3.3.4.1 matrix 10

3.4 eyePC Class Reference 11

3.4.1 Detailed Description . . . 11
3.5 gmres< precond > Class Template Reference . . . 12
3.5.1 Detailed Description . . . 12
3.5.2 Constructor & Destructor Documentation . . . 12
3.5.2.1 gmres . . . 12
3.5.3 Member Function Documentation . . . 13
3.5.3.1 solve . . . 13
3.6 node Class Reference . . . 14
3.6.1 Detailed Description . . . 16
3.6.2 Constructor & Destructor Documentation . . . 16
3.6.2.1 node . . . 16
3.6.3 Member Function Documentation . . . 16
3.6.3.1 redParent . . . 16
3.6.3.2 solveL . . . 16
3.6.3.3 solveU . . . 16
3.7 params Class Reference . . . 17
3.7.1 Detailed Description . . . 18
3.7.2 Constructor & Destructor Documentation . . . 18
3.7.2.1 params . . . 18
3.7.2.2 params . . . 18
3.8 redNode Class Reference . . . 19
3.8.1 Detailed Description . . . 19
3.8.2 Constructor & Destructor Documentation . . . 20
3.8.2.1 redNode . . . 20
3.9 RedSVD::RedPCA< _MatrixType > Class Template Reference . . . 21
3.10 RedSVD::RedSVD< _MatrixType > Class Template Reference . . . 22
3.11 RedSVD::RedSymEigen< _MatrixType > Class Template Reference . . . 23
3.12 superNode Class Reference . . . 24
3.12.1 Detailed Description . . . 26
3.12.2 Constructor & Destructor Documentation . . . 26
3.12.2.1 superNode . . . 26
3.12.2.2 superNode . . . 26
3.12.3 Member Function Documentation . . . 26
3.12.3.1 compress . . . 26
3.12.3.2 extendOrthoCols . . . 26
3.12.3.3 extendOrthoCols . . . 26

3.12.3.4 extendOrthoRows . . . 26
3.12.3.5 extendOrthoRows . . . 26
3.12.3.6 schurComp . . . 27
3.12.3.7 schurComp . . . 27
3.13 tree Class Reference . . . 28
3.13.1 Detailed Description . . . 31
3.13.2 Constructor & Destructor Documentation . . . 31
3.13.2.1 tree . . . 31
3.13.2.2 tree . . . 31
3.13.3 Member Function Documentation . . . 31
3.13.3.1 addRedNode . . . 31
3.13.3.2 addSuperNode . . . 32
3.13.3.3 computeSolution . . . 32
3.13.3.4 createAdjList . . . 32
3.13.3.5 createCol2LeafMap . . . 32
3.13.3.6 createLeafEdges . . . 32
3.13.3.7 createSuperNodes . . . 32
3.13.3.8 eliminate . . . 32
3.13.3.9 factorize . . . 32
3.13.3.10 Parameters . . . 32
3.13.3.11 setRanks . . . 33
3.13.3.12 solve . . . 33
3.13.3.13 solve . . . 33
3.13.3.14 solveL . . . 33
3.13.3.15 solveU . . . 33
3.13.4 Member Data Documentation . . . 33
3.13.4.1 frobNorms . . . 33
3.13.4.2 totalSizes . . . 33

Chapter 1

Class Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

diagPC . . . 8
edge . . . 9
eyePC . . . 11
gmres< precond > . . . 12
node . . . 14
  blackNode . . . 5
  redNode . . . 19
  superNode . . . 24
  superNode . . . 24
params . . . 17
RedSVD::RedPCA< _MatrixType > . . . 21
RedSVD::RedSVD< _MatrixType > . . . 22
RedSVD::RedSymEigen< _MatrixType > . . . 23
tree . . . 28

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

blackNode (Class blackNode) 5
diagPC (This is the diagonal preconditioner) 8
edge (Class edge) 9
eyePC (This is the default preconditioner) 11
gmres< precond > (Class gmres) 12
node (Class node) 14
params (Parameters class) 17
redNode (Class redNode) 19
RedSVD::RedPCA< _MatrixType > 21
RedSVD::RedSVD< _MatrixType > 22
RedSVD::RedSymEigen< _MatrixType > 23
superNode (Class superNode) 24
tree 28

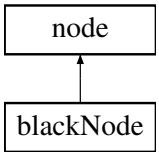
Chapter 3

Class Documentation

3.1 blackNode Class Reference

Class blackNode.

`#include <blackNode.h>`Inheritance diagram for blackNode::



Public Member Functions

- blackNode ()

  Default constructor.

- blackNode (redNode ∗, tree ∗, int, int)

  Constructor:.

- ∼blackNode ()

  Destructor.

- redNode ∗ parent () const

  Returns parent_.

- int ∗ OuterIndex () const

  Returns outer_index_ptr_.

- int ∗ InnerIndex () const

  Returns inner_index_ptr_.

- redNode ∗ leftChild () const

  Returns pointer to the left child.

- redNode ∗ rightChild () const

  Returns pointer to the right child.

- superNode ∗ superChild () const

  Returns pointer to the super child.

- redNode ∗ redParent ()

  Returns pointer to its parent.

- redNode ∗ redParent (int)

  overloaded version for many levels up

- double mergeChildren ()

  Merge left and right redNodes to a superNode.

- void mergeRHS ()

  Merge RHS of children.

- timeTuple2 schurComp ()

  Apply schur-complement.

3.1.1 Detailed Description

Class blackNode. Inherited from the general class node. It is a node correponding to the local variables, and multipole equations.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 blackNode::blackNode (redNode ∗ *P*, tree ∗ *T*, int *first*, int *last*)

Constructor:. input arguments: pointer to the redNode parent, pointer to the tree, range of belonging rows/cols

Array of start indices in edgetab

Adjacency array of every vertex

3.1.3 Member Function Documentation

3.1.3.1 double blackNode::mergeChildren ()

Merge left and right redNodes to a superNode. Output is the time elapsed.

3.1.3.2 timeTuple2 blackNode::schurComp ()

Apply schur-complement. This function is called immediately after its superNode child is eliminated.

The documentation for this class was generated from the following files:

- blackNode.h
- blackNode.cpp

3.2 diagPC Class Reference

This is the diagonal preconditioner.

`#include <diagPC.h>`

Public Member Functions

- diagPC ()
  Default constructor.
- diagPC (spMat ∗A)
  constructor
- ∼diagPC ()
  Destructor.
- VectorXd & solve (VectorXd &b)
  Solve function.

3.2.1 Detailed Description

This is the diagonal preconditioner.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 diagPC::diagPC (spMat ∗ *A*) `[inline]`

constructor Takes pointer to the sparse matrix

The documentation for this class was generated from the following file:

- diagPC.h

3.3 edge Class Reference

Class edge.

`#include <edge.h>`

Public Member Functions

- edge ()
  Default constructor.
- edge (node ∗, node ∗)
  Construcotr.
- node ∗ source () const
  Returns pointer to the source node.
- node ∗ destination () const
  Returns pointer to the destination node.
- bool isWellSeparated ()
  Check if edge is between two spearated nodes.
- void compress ()
  Compress the edge.
- bool isEliminated ()
  Check if the edge is eliminated.
- bool isCompressed () const
  Returns the compressed flag.

Public Attributes

- densMat ∗ matrix
  Pointer to the interaction matrix.

3.3.1 Detailed Description

Class edge. An edge can connect any two nodes (red, black, superNode). It has the data of the corresponding interaction block. Convention: Edge is created by its source node.

3.3.2 Constructor & Destructor Documentation

3.3.2.1 edge::edge (node ∗ *s*, node ∗ *d*)

Construcotr. Constructor inputs are: pointer to the source node, pointer to the destination node,

3.3.3 Member Function Documentation

3.3.3.1 void edge::compress ()

Compress the edge. This function should be called after an edge is compressed, and moved to the parent level. As a result the following steps happen:

- Set the compressed_ flag to [true]
- Free the matrix memory

3.3.3.2 bool edge::isEliminated ()

Check if the edge is eliminated. An edge is eliminated during the elimination process iff either its source or destination node is eliminated

3.3.3.3 bool edge::isWellSeparated ()

Check if edge is between two spearated nodes. This function check if the edges source and separation are well separated. The adjacency list of nodes are used to determine that. Note that source and dest. can be different node types at different levels.

3.3.4 Member Data Documentation

3.3.4.1 densMat* edge::matrix

Pointer to the interaction matrix. a pointer to a dense m by n matrix note: n is the number of columns (variables at source node), and m is the number of rows (equations at destination node).

The documentation for this class was generated from the following files:

- edge.h
- edge.cpp

3.4 eyePC Class Reference

This is the default preconditioner.

`#include <eyePC.h>`

Public Member Functions

- eyePC ()
  Default constructor.
- ~eyePC ()
  Destructor.
- VectorXd & solve (VectorXd &b)
  Solve function.

3.4.1 Detailed Description

This is the default preconditioner. Essentialy, this precondiotner = no preconditioner!

The documentation for this class was generated from the following file:

- eyePC.h

3.5 gmres< precond > Class Template Reference

Class gmres.

`#include <gmres.h>`

Public Member Functions

- gmres ()

  Default constructor.

- ~gmres ()

  Destructor.

- gmres (spMat *, precond *, VectorXd *, VectorXd *, int, double, bool)

  Constructor with input parameters.

- void solve ()

  solve: start iteration until:

- VectorXd * residuals ()

  Returns vector of residuals.

- VectorXd * retVal ()

  Returns solution.

- int totalIters () const

  Returns total number of itertions.

- double totalTime () const

  Returns total time to solve.

3.5.1 Detailed Description

template<class precond> class gmres< precond >

Class gmres. This class get a sparse matrix, and a rhs, apply the GMRES method until convergence.

3.5.2 Constructor & Destructor Documentation

3.5.2.1 template<class precond > gmres< precond >::gmres (spMat * *A*, precond * *P*, VectorXd * *rhs*, VectorXd * *x0*, int *m*, double *eps*, bool *verb*) [inline]

Constructor with input parameters. Pointer to the sparse matrix. Pointer to the preconditioner. Pointer to the RHS vector. Pointer to the initial gauess vector. Maximum number of iterations. Accuracy threshold.

Allocate memory to matrices and vectors

3.5.3 Member Function Documentation

3.5.3.1 template<class precond > void gmres< precond >::solve () [inline]

solve: start iteration until: 1) Krylov sub-space stops growing or 2) Reach maximum number of iteration or 3) Reach desired accuracy

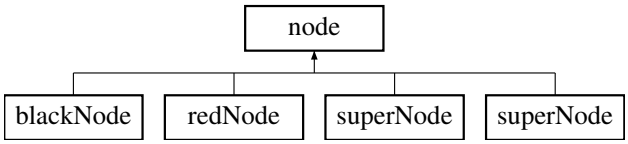
The documentation for this class was generated from the following file:

- gmres.h

3.6 node Class Reference

Class node.

`#include <node.h>`Inheritance diagram for node::



Public Member Functions

- node ()
  Default constructor.
- node (node *, tree *, int, int)
  Constructor:.
- virtual ~node ()
  Destructor.
- node * parent () const
  Returns parent_.
- int IndexFirst () const
  Returns index_first_.
- int IndexLast () const
  Returns index_last_.
- tree * Tree () const
  Returns tree_ptr_.
- bool isEmpty ()
  True if this node contains no column of the matrix.
- bool isEliminated () const
  True if this node is eliminated.
- void eliminate ()
  Set the elimination flag to [true].
- void deEliminate ()
  Set the elimination flag to [false].
- int m () const
  Returns m_.

- int n () const

  Returns n_.

- void m (int val)

  Set m_.

- void n (int val)

  Set n_.

- virtual redNode * redParent ()=0

  Returns pointer to the redParent (pure virtual).

- virtual redNode * redParent (int)=0

  Overload: int is the lvel of grand parent.

- void eraseCompressedEdges ()

  Erase removed edges from the list of incoming/outgoing edges.

- VectorXd * RHS ()

  Access to the pointer of the RHS vector.

- void RHS (VectorXd *in)

  Set the pointer of the RHS vector.

- VectorXd * VAR ()

  Access to the pointer of the variables vector.

- void VAR (VectorXd *in)

  Set the pointer of the VAR vector.

- void solveL ()

  solve L z = b

- void solveU ()

  solve U VAR = RHS

Public Attributes

- std::vector< edge * > inEdges

  List of incoming edges.

- std::vector< edge * > outEdges

  List of outgoing edges.

- bool eliminated_

  A boolean flag to keep track of elimination.

- densMat * invPivot

The inevrse of the selfEdge matrix (i.e., pivot).

- bool rhsUpdated_

  A boolean flag to keep track of updated RHS.

- int order

  The order of elimination.

3.6.1 Detailed Description

Class node. This (virtual) class is an interface for the three inherited classes blackNode, redNode, and superNode

3.6.2 Constructor & Destructor Documentation

3.6.2.1 node::node (node * *P*, tree * *T*, int *first*, int *last*)

Constructor:. input arguments: pointer to parent, pointer to the tree, range of belonging rows/cols Constructor by passing the ptr to the tree, and range of belonging columns

3.6.3 Member Function Documentation

3.6.3.1 virtual redNode* node::redParent () [pure virtual]

Returns pointer to the redParent (pure virtual). redNode -> returns itself blackNode -> returns parent() superNode -> returns parent() of paranet()

Implemented in blackNode, redNode, superNode, and superNode.

3.6.3.2 void node::solveL ()

solve L z = b This function updates RHS, which is solve for z in L z = b It uses the order of elimination.

3.6.3.3 void node::solveU ()

solve U VAR = RHS Solve for unknowns of this cluster. It uses the order of elimination.

The documentation for this class was generated from the following files:

- node.h
- node.cpp

3.7 params Class Reference

Parameters class.

`#include <params.h>`

Public Member Functions

- std::string Input_Matrix_File () const
  Public function to access to input matrix file.
- params ()
  Default constructor.
- params (char *)
  Constructor.
- params (char *, int, int, int, double, int, double, double, int, double, int, double, int, bool)
  Constructor.
- ~params ()
  Destructor.
- unsigned int treeLevelThreshold () const
  Returns subdividing_threshold_.
- double epsilon () const
  Returns epsilon_.
- int lowRankMethod () const
  Returns lowRankMethod_.
- int cutOffMethod () const
  Returns cutOffMethod_.
- double aPrioriRank () const
  Returns aPrioryRank_.
- double rankCapFactor () const
  Returns rankCapFactor_.
- double deployFactor () const
  Returns deployFactor_.
- int gmresMaxIters () const
  Returns GMRES maximum number of iterations.
- double gmresEpsilon () const
  Returns GMRES residual threshold.

- int gmresPC () const
  Returns GMRES preconditioner.
- bool gmresVerbose () const
  Returns GMRES verbose.
- double ILUDropTol () const
  Returns ILU drop tol.
- int ILUFill () const
  Returns ILU Fill.
- int normCols () const
  Returns normCols_.

3.7.1 Detailed Description

Parameters class. This class loads an input parameter file, and store different parameters. All other classes can access to the parameters using this class.

3.7.2 Constructor & Destructor Documentation

3.7.2.1 params::params (char * *address*)

Constructor. Should construct a parameter object with the char* of \ the path to the input parameter file

In the input parameter file, empty line and lines starting \ with # will be ignored

3.7.2.2 params::params (char * *matrixFile*, int *depth*, int *lrmeth*, int *cometh*, double *eps*, int *aprank*, double *rankCap*, double *depFac*, int *gmresMI*, double *gmresEps*, int *gmresPrec*, double *ILUDT*, int *ILUFil*, bool *normCols*)

Constructor. With this constructor we directly provide parameters

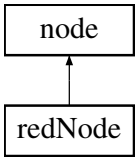
The documentation for this class was generated from the following files:

- params.h
- params.cpp

3.8 redNode Class Reference

Class redNode.

`#include <redNode.h>`Inheritance diagram for redNode::



Public Member Functions

- redNode ()
  Default constructor.
- redNode (blackNode ∗, tree ∗, bool, int, int)
  Constructor:.
- ∼redNode ()
  Destructor.
- blackNode ∗ child () const
  Returns child_.
- bool IsLeaf () const
  True if this is leaf node.
- blackNode ∗ parent () const
  Return parent_.
- unsigned int level () const
  Returns level_.
- bool which () const
  Returns which_.
- void **createBlackNode** ()
- std::set< redNode ∗ > ∗ **AdjList** ()
- redNode ∗ redParent ()
  Returns pointer to it self.
- redNode ∗ redParent (int)
  overloaded version for many levels up

3.8.1 Detailed Description

Class redNode. Inherited from the general class node. It is a node correponding to the multipole variables, and local equations.

3.8.2 Constructor & Destructor Documentation

3.8.2.1 redNode::redNode (blackNode * *P*, tree * *T*, bool *W*, int *first*, int *last*)

Constructor:. input arguments: pointer to the blackNode parent, pointer to the tree, which child, range of belonging rows/cols

The documentation for this class was generated from the following files:

- redNode.h
- redNode.cpp

3.9 RedSVD::RedPCA< _MatrixType > Class Template Reference

Public Types

- typedef _MatrixType **MatrixType**
- typedef MatrixType::Scalar **Scalar**
- typedef MatrixType::Index **Index**
- typedef Eigen::Matrix< Scalar, Eigen::Dynamic, Eigen::Dynamic > **DenseMatrix**
- typedef Eigen::Matrix< Scalar, Eigen::Dynamic, 1 > **ScalarVector**

Public Member Functions

- **RedPCA** (const MatrixType &A)
- **RedPCA** (const MatrixType &A, const Index rank)
- void **compute** (const DenseMatrix &A, const Index rank)
- DenseMatrix **components** () const
- DenseMatrix **scores** () const

template<typename _MatrixType> class RedSVD::RedPCA< _MatrixType >

The documentation for this class was generated from the following file:

- rsvd.h

3.10 RedSVD::RedSVD< _MatrixType > Class Template Reference

Public Types

- typedef _MatrixType **MatrixType**
- typedef MatrixType::Scalar **Scalar**
- typedef MatrixType::Index **Index**
- typedef Eigen::Matrix< Scalar, Eigen::Dynamic, Eigen::Dynamic > **DenseMatrix**
- typedef Eigen::Matrix< Scalar, Eigen::Dynamic, 1 > **ScalarVector**

Public Member Functions

- **RedSVD** (const MatrixType &A)
- **RedSVD** (const MatrixType &A, const Index rank)
- void **compute** (const MatrixType &A, const Index rank)
- DenseMatrix **matrixU** () const
- ScalarVector **singularValues** () const
- DenseMatrix **matrixV** () const

template<typename _MatrixType> class RedSVD::RedSVD< _MatrixType >

The documentation for this class was generated from the following file:

- rsvd.h

3.11 RedSVD::RedSymEigen< _MatrixType > Class Template Reference

Public Types

- typedef _MatrixType **MatrixType**
- typedef MatrixType::Scalar **Scalar**
- typedef MatrixType::Index **Index**
- typedef Eigen::Matrix< Scalar, Eigen::Dynamic, Eigen::Dynamic > **DenseMatrix**
- typedef Eigen::Matrix< Scalar, Eigen::Dynamic, 1 > **ScalarVector**

Public Member Functions

- **RedSymEigen** (const MatrixType &A)
- **RedSymEigen** (const MatrixType &A, const Index rank)
- void **compute** (const MatrixType &A, const Index rank)
- ScalarVector **eigenvalues** () const
- DenseMatrix **eigenvectors** () const

template<typename _MatrixType> class RedSVD::RedSymEigen< _MatrixType >

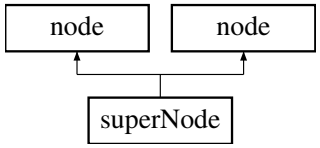
The documentation for this class was generated from the following file:

- rsvd.h

3.12 superNode Class Reference

Class superNode.

`#include <superNode.h>`Inheritance diagram for superNode::



Public Member Functions

- superNode ()
  Default constructor.
- superNode (blackNode ∗, tree ∗, int, int)
  Constructor:.
- ∼superNode ()
  Destructor.
- blackNode ∗ parent () const
  Returns parent_.
- redNode ∗ redParent ()
  Returns pointer to its parent.
- redNode ∗ redParent (int)
  overloaded version for many levels up
- unsigned int level () const
  Returns level_.
- timeTuple2 compress ()
  Compress all well separated interactions.
- timeTuple2 schurComp ()
  Actually eliminate the node, and its parent.
- void splitVAR ()
  Split the solution to left/right redNodes.
- void addRandomCols (densMat &, int)
  Pad some random columns to the matrix (on the right).
- void addRandomRows (densMat &, int)
  Pad some random rows to the matrix (at the bottom).

- void extendOrthoCols (densMat &, int)
  Pad some random columns and apply QR (on the right).
- void extendOrthoRows (densMat &, int)
  Pad some random rows and apply QR (at the bottom).
- bool criterionCheck (int, double, RedSVD::RedSVD< densMat > &, densMat &, int)
  check if the approximation is fine
- superNode ()
  Default constructor.
- superNode (blackNode *, tree *, int, int)
  Constructor:.
- ~superNode ()
  Destructor.
- blackNode * parent () const
  Returns parent_.
- redNode * redParent ()
  Returns pointer to its parent.
- redNode * redParent (int)
  overloaded version for many levels up
- unsigned int level () const
  Returns level_.
- void compress ()
  Compress all well separated interactions.
- void schurComp ()
  Actually eliminate the node, and its parent.
- void splitVAR ()
  Split the solution to left/right redNodes.
- void addRandomCols (densMat &, int)
  Pad some random columns to the matrix (on the right).
- void addRandomRows (densMat &, int)
  Pad some random rows to the matrix (at the bottom).
- void extendOrthoCols (densMat &, int)
  Pad some random columns and apply QR (on the right).
- void extendOrthoRows (densMat &, int)
  Pad some random rows and apply QR (at the bottom).

- bool criterionCheck (int, double, RedSVD::RedSVD< densMat > &, densMat &, int)

  check if the approximation is fine

3.12.1 Detailed Description

Class superNode. Inherited from the general class node. It is a node correponding to particles

3.12.2 Constructor & Destructor Documentation

3.12.2.1 superNode::superNode (blackNode * *P*, tree * *T*, int *first*, int *last*)

Constructor:. input arguments: pointer to the blackNode parent, pointer to the tree, range of belonging rows/cols

3.12.2.2 superNode::superNode (blackNode *, tree *, int, int)

Constructor:. input arguments: pointer to the blackNode parent, pointer to the tree, range of belonging rows/cols

3.12.3 Member Function Documentation

3.12.3.1 timeTuple2 superNode::compress ()

Compress all well separated interactions. Exit is a timeTuple: (lowRank time, everything else time)

3.12.3.2 void superNode::extendOrthoCols (densMat &, int)

Pad some random columns and apply QR (on the right). We assume the input matrix consists of orthonormal columns.

3.12.3.3 void superNode::extendOrthoCols (densMat & *A*, int *p*)

Pad some random columns and apply QR (on the right). We assume the input matrix consists of orthonormal columns.

3.12.3.4 void superNode::extendOrthoRows (densMat &, int)

Pad some random rows and apply QR (at the bottom). We assume the input matrix consists of orthonormal rows.

3.12.3.5 void superNode::extendOrthoRows (densMat & *A*, int *p*)

Pad some random rows and apply QR (at the bottom). We assume the input matrix consists of orthonormal rows.

3.12.3.6 void superNode::schurComp ()

Actually eliminate the node, and its parent. Eliminating a node invloves going through all edges, and create new edges based on schur complement

3.12.3.7 timeTuple2 superNode::schurComp ()

Actually eliminate the node, and its parent. Eliminating a node invloves going through all edges, and create new edges based on schur complement

The documentation for this class was generated from the following files:

- superNode.h
- superNode_rsvd.h
- superNode.cpp
- superNode_general.cpp
- superNode_QR.cpp
- superNode_rsvd.cpp
- superNode_svd.cpp

3.13 tree Class Reference

`#include <tree.h>`

Public Member Functions

- tree ()
  Constructor (for root node).
- tree (params ∗)
  Constructor with parameters as input.
- ∼tree ()
  Destructor.
- params ∗ Parameters () const
  Provide a submatrix with given range of columns.
- spMat ∗ Matrix () const
  Returns matrix_.
- spMatBool ∗ SymbMatrix () const
  Returns symb_matrix_.
- int **n** () const
- std::vector< redNodeStrList > ∗ redNodeList ()
  Returns pointer to the list of redNodes.
- std::vector< superNodeStrList > ∗ superNodeList ()
  Returns pointer to the list of superNodes.
- std::vector< double > ∗ meanRanks ()
  Returns pointer to mean rank list.
- std::vector< int > ∗ minRanks ()
  Returns pointer to min rank list.
- std::vector< int > ∗ maxRanks ()
  Returns pointer to max rank list.
- unsigned int maxLevel ()
  Returns the maximum level number.
- void addRedNode (unsigned int, redNode ∗)
  Add a new redNode to the list.
- void addSuperNode (unsigned int, superNode ∗)
  Add a new superNode to the list.

- void createAdjList ()
  Form the adjacency list for all redNodes.
- void createCol2LeafMap ()
  Using a Map we create key-value pairs for all leaf nodes.
- void createLeafEdges ()
  Load the data to edges between tree leaves.
- double createSuperNodes (unsigned int)
  Creating superNodes.
- timeTuple4 eliminate (unsigned int)
  Eliminate variables at level [l] in the tree.
- void setRHS (VectorXd &)
  Set the RHS for all nodes.
- void setLeafRHS (VectorXd &)
  Dedicate memory, and set the RHS for leaf nodes.
- void setLeafRHSVAR ()
  Dedicate memory, and set the VAR for leaf nodes.
- void solveL (unsigned int)
  SolveL for black, super, and red Nodes in a level.
- void solveU (unsigned int)
  SolveU for black, super, and red Nodes in a level.
- VectorXd & solve ()
  Top to bottom traverse to solve for a given RHS.
- VectorXd & solve (VectorXd &)
- void factorize ()
  Bottom to Top traverse to decompose A = L U.
- VectorXd & computeSolution ()
  Compute solution.
- VectorXd ∗ VAR ()
  Access to the pointer of the variables vector.
- void setRanks ()
  set ranks
- void log (std::string)
  store everything in a log file
- void computeFrobNorm (unsigned int)

Compute the Frob. norm at level l.

- VectorXd * retVal ()

  Returns a pointer to the solution.

- void normalize_cols (spMat *)

  Divide each col of the matrix by its maximum value.

Public Attributes

- VectorXi * permutationVector

  The vecotr contains required permutation of the last level.

- double assembleTime

  Tree assembling time.

- double precondFactTime

  Time for preconditioner factorization time (solve).

- double accuracy

  GMRES final relative accuracy.

- double residual

  GMRES final relative residual.

- VectorXd * residuals

  GMRES residuals.

- int gmresTotalIters

  GMRES total iterations.

- double gmresTotalTime

  GMRES total time.

- bool padding

  Padding happend?

- bool largePivot

  Large pivots happend?

- std::vector< double > frobNorms

  Store Forb. of each level.

- std::vector< double > totalSizes

  Totoal size of each level.

- double globFrobNorm

  global frob norm = sum(frobNorms)

- double globSize

  global size = sum(totalSizes)

- int count

  COunt the number of eliminated nodes (so far).

3.13.1 Detailed Description

Class tree This class represents a tree of nodes, and provide necessary information/routines for tree The full matrix is also stored as a member of this class, only one version that is globally accessible.

3.13.2 Constructor & Destructor Documentation

3.13.2.1 tree::tree () `[inline]`

Constructor (for root node). In the constructor we read the matrix data from input files (in market-matrix format), and store it as EigenSparse matrix. Also, the nodes of the tree will be generated in the constructor Default constructor

3.13.2.2 tree::tree (params * *par*)

Constructor with parameters as input.

Constructing the root

The root of the tree is a redNode. We need to pass the range of columns that belong to each node. The tree will be created in a BFS order. The order of things need to be done:

- create the root
- When a redNode born it will be added to the redNodelist
- for the last created level do the permuation
- for the last created level decide if further subdividing is required (based on threshold_level)
- if yes for all redNodes in the last level call createBlacknode()

set the global frob norm = 0 initially

Initially no node is eliminated

Initially no padding and no large pivot is assumed

3.13.3 Member Function Documentation

3.13.3.1 void tree::addRedNode (unsigned int *l*, redNode * *ptr*)

Add a new redNode to the list. The blackNode will use this function to create its red children. The first argument is the level of the new red node and the second is the pointer to it

3.13.3.2 void tree::addSuperNode (unsigned int *l*, superNode * *ptr*)

Add a new superNode to the list. The blackNode will use this function to create its superNode child. The first argument is the level of the new superNode and the second is the pointer to it

3.13.3.3 VectorXd & tree::computeSolution ()

Compute solution. i.e., collect the solution of all leaves

3.13.3.4 void tree::createAdjList ()

Form the adjacency list for all redNodes. Note that this is the list of original interactions induced by children. Later on during the elimination, nodes may have new interactions, and will be stored through edges. Only symbolic matrix will be used here.

3.13.3.5 void tree::createCol2LeafMap ()

Using a Map we create key-value pairs for all leaf nodes. For each non-empty leaf the key is the index of the first columns it posses, and the value is the pointer to the leaf.

3.13.3.6 void tree::createLeafEdges ()

Load the data to edges between tree leaves. After the tree, and adjacency lists are created, load the sub block of matrix to the edges between leaves.

3.13.3.7 double tree::createSuperNodes (unsigned int *l*)

Creating superNodes. This function combine sibling redNodes at level [l] of the tree, and create a new superNode. By construction, the parent of the resulted superNode is a blackNode. Note that in order to acess pairs of sibling redNodes at level [l] of the tree, we go through the redNodes at level [l-1] in the tree, and look at their grand red children. Output is the time elapsed: tuple(compression, schurComp).

3.13.3.8 timeTuple4 tree::eliminate (unsigned int *l*)

Eliminate variables at level [l] in the tree. Output is the time elapsed.

3.13.3.9 void tree::factorize ()

Bottom to Top traverse to decompose A = L U. Bottom to top traverse.

At each level, first create the superNodes (i.e., combine two redNodes), and then eliminate (and compress) all superNodes, as well as their black node parents.

3.13.3.10 params* tree::Parameters () const `[inline]`

Provide a submatrix with given range of columns. The order of input arguments: first column, last column, InnerIndex list, OuterIndex liset Note: The indices in the in/out-er lists always start from 0! Returns 0 if successful, returns 1 otherwise Returns param_

3.13.3.11 void tree::setRanks ()

set ranks At each level compute mean, min, and max size of the redNodes

3.13.3.12 VectorXd & tree::solve (VectorXd & *RHS*)

bottom to top traverse to solve L z = b

top to bottom traverse to solve U x = z

3.13.3.13 VectorXd & tree::solve ()

Top to bottom traverse to solve for a given RHS. Top to bottom traverse.

If no RHS is provided it uses the RHS from input params, and automatically permute it. For the overloaded version, it takes a permuted RHS as an input.

Start from the top, solve the first set of equations directly (e.g., direct LU), Then back-propagate toword leaves, and solve all unknowns. This version uses the RHS provided in the input params.

3.13.3.14 void tree::solveL (unsigned int *l*)

SolveL for black, super, and red Nodes in a level. This functions work from bottom to top

3.13.3.15 void tree::solveU (unsigned int *l*)

SolveU for black, super, and red Nodes in a level. This functions work from top to the bottom

3.13.4 Member Data Documentation

3.13.4.1 std::vector<double> tree::frobNorms

Store Forb. of each level. The frob. norm of level l is defined as: sqrt(sum_{e is edge between two superNodes} e.matrix.frob^2)

3.13.4.2 std::vector<double> tree::totalSizes

Totoal size of each level. For each level, l, it computes: sum of sizes (= m∗n for a m-by-n matrix) of all blocks in that level

The documentation for this class was generated from the following files:

- tree.h
- tree.cpp

Index

addRedNode
    tree, 31
addSuperNode
    tree, 31

blackNode, 5
    blackNode, 6
    mergeChildren, 6
    schurComp, 6

compress
    edge, 10
    superNode, 26
computeSolution
    tree, 32
createAdjList
    tree, 32
createCol2LeafMap
    tree, 32
createLeafEdges
    tree, 32
createSuperNodes
    tree, 32

diagPC, 8
    diagPC, 8

edge, 9
    compress, 10
    edge, 9
    isEliminated, 10
    isWellSeparated, 10
    matrix, 10
eliminate
    tree, 32
extendOrthoCols
    superNode, 26
extendOrthoRows
    superNode, 26
eyePC, 11

factorize
    tree, 32
frobNorms
    tree, 33

gmres, 12
    gmres, 12
    solve, 13

isEliminated
    edge, 10
isWellSeparated
    edge, 10

matrix
    edge, 10
mergeChildren
    blackNode, 6

node, 14
    node, 16
    redParent, 16
    solveL, 16
    solveU, 16

Parameters
    tree, 32
params, 17
    params, 18

redNode, 19
    redNode, 20
redParent
    node, 16
RedSVD::RedPCA, 21
RedSVD::RedSVD, 22
RedSVD::RedSymEigen, 23

schurComp
    blackNode, 6
    superNode, 26, 27
setRanks
    tree, 32
solve
    gmres, 13
    tree, 33
solveL
    node, 16
    tree, 33
solveU
    node, 16

tree, 33
superNode, 24
compress, 26
extendOrthoCols, 26
extendOrthoRows, 26
schurComp, 26, 27
superNode, 26

totalSizes
tree, 33
tree, 28
addRedNode, 31
addSuperNode, 31
computeSolution, 32
createAdjList, 32
createCol2LeafMap, 32
createLeafEdges, 32
createSuperNodes, 32
eliminate, 32
factorize, 32
frobNorms, 33
Parameters, 32
setRanks, 32
solve, 33
solveL, 33
solveU, 33
totalSizes, 33
tree, 31