# groupF

Generated by Doxygen 1.6.1

Fri Dec 18 13:54:03 2015

# Contents

# Chapter 1

# Class Index

## 1.1 Class List

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

# Chapter 2

# Class Documentation

## 2.1 blmintegrals Class Reference

### Public Member Functions

- void integral1 (vector< vector< double > > &a_matrix, vector< double > &a_x1, vector< double > &a_x2, vector< double > &a_x3, vector< vector< double > > &Etensor)
- void integral2 (vector< double > &a_vector, vector< double > &a_x1, vector< double > &a_x2, vector< double > &a_x3, vector< double > &a_force)
- void integral3xx (vector< double > &a_vector, vector< double > &a_x1, vector< double > &a_x2, vector< double > &a_x3, vector< double > &a_zvector, vector< double > &a_traction)
- void integral3yy (vector< double > &a_vector, vector< double > &a_x1, vector< double > &a_x2, vector< double > &a_x3, vector< double > &a_zvector, vector< double > &a_traction)
- void integral3zz (vector< double > &a_vector, vector< double > &a_x1, vector< double > &a_x2, vector< double > &a_x3, vector< double > &a_zvector, vector< double > &a_traction)

### 2.1.1 Member Function Documentation

#### 2.1.1.1 void blmintegrals::integral1 (vector< vector< double > > & *a_matrix*, vector< double > & *a_x1*, vector< double > & *a_x2*, vector< double > & *a_x3*, vector< vector< double > > & *Etensor*)

Perform Integral 1 for balance of linear momentum. First argument is matrix to return into. Next 3 are input vectors from node table. Last is input matrix of elasticity tensor.

#### 2.1.1.2 void blmintegrals::integral2 (vector< double > & *a_vector*, vector< double > & *a_x1*, vector< double > & *a_x2*, vector< double > & *a_x3*, vector< double > & *a_force*)

Perform Integral 2 for balance of linear momentum. First argument is vector to return into. Next 3 are input vectors from node table. Last is input vector of force components.

#### 2.1.1.3 void blmintegrals::integral3xx (vector< double > & *a_vector*, vector< double > & *a_x1*, vector< double > & *a_x2*, vector< double > & *a_x3*, vector< double > & *a_zvector*, vector< double > & *a_traction*)

Perform surface Integral 3 for balance of linear momentum on x surface. First argument is vector to return into. Next 3 are input vectors from node table. Last is input vector of traction components.

#### 2.1.1.4 void blmintegrals::integral3yy (vector< double > & *a_vector*, vector< double > & *a_x1*, vector< double > & *a_x2*, vector< double > & *a_x3*, vector< double > & *a_zvector*, vector< double > & *a_traction*)

Perform surface Integral 3 for balance of linear momentum on y surface. First argument is vector to return into. Next 3 are input vectors from node table. Last is input vector of traction components.

#### 2.1.1.5 void blmintegrals::integral3zz (vector< double > & *a_vector*, vector< double > & *a_x1*, vector< double > & *a_x2*, vector< double > & *a_x3*, vector< double > & *a_zvector*, vector< double > & *a_traction*)

Perform surface Integral 3 for balance of linear momentum on z surface. First argument is vector to return into. Next 3 are input vectors from node table. Last is input vector of traction components.

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

- src/blmintegrals.H

## 2.2 CGSolver Class Reference

### Public Member Functions

- double solve (const SparseMatrix &a_A, const vector< double > &a_rhs, double a_tolerance, int a_iter, vector< double > &a_phi)

### 2.2.1 Member Function Documentation

#### 2.2.1.1 double CGSolver::solve (const SparseMatrix & *a_A*, const vector< double > & *a_rhs*, double *a_tolerance*, int *a_iter*, vector< double > & *a_phi*)

Conjugate gradient solver. Solves until max norm of residual is less than tolerance, or reaches passed in a_iter. Returns final residual.

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

- src/CGSolver.H

## 2.3 femfunctions Class Reference

### Public Member Functions

- femfunctions ()

  *Constructor.*

- void mat_mult (vector< vector< double > > &A_matrix, vector< vector< double > > &B_matrix, vector< vector< double > > &C_matrix)

  *Dense Matrix Multiplication.*

- void mat_mult_vec (vector< vector< double > > &A_matrix, vector< double > &B_vector, vector< double > &C_vector)

  *Dense Matrix Multiplication with Vector.*

- void vec_mult_mat (vector< double > &A_vector, vector< vector< double >> &B_matrix, vector< double > &C_vector)

  *Dense Matrix Multiplication with Vector.*

- void mat_transpose (vector< vector< double > > &A_matrix, vector< vector< double > > &A_matrix_trans)

  *Matrix Tranpose.*

- void inverse_mat (vector< vector< double > > &a_matrix, vector< vector< double > > &a_matrix_inv, vector< vector< double > > &a_matrix_inv_trans)

  *Computes the inverse of a matrix m.*

- double Jacobian (vector< vector< double > > &a_matrix)

  *Jacobian of matrix.*

- double phi0 (double z1, double z2, double z3)

  *Mapping Functions.*

- double phi1 (double z1, double z2, double z3)

  *Mapping Functions.*

- double phi2 (double z1, double z2, double z3)

  *Mapping Functions.*

- double phi3 (double z1, double z2, double z3)

  *Mapping Functions.*

- double phi4 (double z1, double z2, double z3)

  *Mapping Functions.*

- double phi5 (double z1, double z2, double z3)

  *Mapping Functions.*

- double phi6 (double z1, double z2, double z3)

  *Mapping Functions.*

- double phi7 (double z1, double z2, double z3)

  *Mapping Functions.*

- double dphi0dz1 (double z1, double z2, double z3)

  *Derivatives of Mapping Functions with respect to z1.*

- double dphi1dz1 (double z1, double z2, double z3)

  *Derivatives of Mapping Functions with respect to z1.*

- double dphi2dz1 (double z1, double z2, double z3)

  *Derivatives of Mapping Functions with respect to z1.*

- double dphi3dz1 (double z1, double z2, double z3)

  *Derivatives of Mapping Functions with respect to z1.*

- double dphi4dz1 (double z1, double z2, double z3)

  *Derivatives of Mapping Functions with respect to z1.*

- double dphi5dz1 (double z1, double z2, double z3)

  *Derivatives of Mapping Functions with respect to z1.*

- double dphi6dz1 (double z1, double z2, double z3)

  *Derivatives of Mapping Functions with respect to z1.*

- double dphi7dz1 (double z1, double z2, double z3)

  *Derivatives of Mapping Functions with respect to z1.*

- double dphi0dz2 (double z1, double z2, double z3)

  *Derivatives of Mapping Functions with respect to z2.*

- double dphi1dz2 (double z1, double z2, double z3)

  *Derivatives of Mapping Functions with respect to z2.*

- double dphi2dz2 (double z1, double z2, double z3)

  *Derivatives of Mapping Functions with respect to z2.*

- double dphi3dz2 (double z1, double z2, double z3)

  *Derivatives of Mapping Functions with respect to z2.*

- double dphi4dz2 (double z1, double z2, double z3)

  *Derivatives of Mapping Functions with respect to z2.*

- double dphi5dz2 (double z1, double z2, double z3)

  *Derivatives of Mapping Functions with respect to z2.*

- double dphi6dz2 (double z1, double z2, double z3)

  *Derivatives of Mapping Functions with respect to z2.*

- double dphi7dz2 (double z1, double z2, double z3)

*Derivatives of Mapping Functions with respect to z2.*

- double dphi0dz3 (double z1, double z2, double z3)

  *Derivatives of Mapping Functions with respect to z3.*
- double dphi1dz3 (double z1, double z2, double z3)

  *Derivatives of Mapping Functions with respect to z3.*
- double dphi2dz3 (double z1, double z2, double z3)

  *Derivatives of Mapping Functions with respect to z3.*
- double dphi3dz3 (double z1, double z2, double z3)

  *Derivatives of Mapping Functions with respect to z3.*
- double dphi4dz3 (double z1, double z2, double z3)

  *Derivatives of Mapping Functions with respect to z3.*
- double dphi5dz3 (double z1, double z2, double z3)

  *Derivatives of Mapping Functions with respect to z3.*
- double dphi6dz3 (double z1, double z2, double z3)

  *Derivatives of Mapping Functions with respect to z3.*
- double dphi7dz3 (double z1, double z2, double z3)

  *Derivatives of Mapping Functions with respect to z3.*

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

- src/femfunctions.H

## 2.4 hsbounds Class Reference

### Public Member Functions

- void HS_bounds (double &k1, double &k2, double &u1, double &u2, double &v2, double &kstar, double &ustar)

### 2.4.1 Member Function Documentation

#### 2.4.1.1 void hsbounds::HS_bounds (double & *k1*, double & *k2*, double & *u1*, double & *u2*, double & *v2*, double & *kstar*, double & *ustar*)

Determines effective bulk (kstar) and shear (ustar) moduli from input bulk (k1,k2) and shear (u1,u2) moduli and proportion (v2)

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

- src/hsbounds.H

## 2.5 JacobiSolver Class Reference

### Public Member Functions

- double solve (const SparseMatrix &a_A, const vector< double > &a_rhs, double a_tolerance, int a_iter, vector< double > &a_phi)

  *solves until max norm of residual is less than tolerance, or reaches passed in a_iter. returns final residual*

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

- src/JacobiSolver.H

## 2.6 Mesher Class Reference

### Public Member Functions

- Mesher ()

  *Default Constructor.*

- Mesher (const int &a_N)

  *Constructor: just require int number of nodes in each direction.*

- void createmesh ()

  *Mesh Creator.*

- vector< vector< double > > getNode ()

  *Get node table (vector of vector of double).*

- vector< vector< int > > getConn ()

  *Get connectivity table (vector of vector of double).*

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

- src/mesher.H

## 2.7 SparseMatrix Class Reference

### Public Member Functions

- SparseMatrix ()

  *Set up an M rows and N columns sparse matrix with all values of zero (no non-zero elements).*

- **SparseMatrix** (int a_M, int a_N)
- vector< double > operator* (const vector< double > &a_v) const

  *Matrix Vector multiply. a_v.size()==N, returns vector of size M.*

- double & operator[ ] (array< int, 2 > a_index)

  *Accessor functions for get and set operations of matrix elements.*

- const double & operator[ ] (array< int, 2 > a_index) const

  *Accessor function just to get a value.*

- void zero ()

  *Zero out all the elements, but leave the sparse structure in place.*

- SparseMatrix transpose () const

  *Build and return a new SparseMatrix that is the transpose of the input matrix.*

- unsigned int M () const

  *Get first dimension size.*

- unsigned int N () const

  *Get second dimension size.*

- bool symmetric () const

  *Return true if symmetric.*

- void print () const

  *Print function for SparseMatrix type.*

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

- src/SparseMatrix.H