NL-PartSol

Generated by Doxygen 1.8.18

Chapter 1

My Personal Index Page

1.1 Introduction

This is the introduction.

1.2 Installation

1.2.1 Step 1: Opening the box

etc...

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

Data Structure Documentation

4.1 Boundaries Struct Reference

`#include <Variables.h>`

Data Fields

- int NumBounds

  Number of boundaries of the domain.
- Load * BCC_i

  Table with all the boundaries and its values.

4.1.1 Detailed Description

Boundary conditions definition

Definition at line 283 of file Variables.h.

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

- /home/migmolper2/NL-PartSol/nl-partsol/include/Variables.h

4.2 Chain Struct Reference

`#include <Variables.h>`

Data Fields

- int I

  Index of a node in the set.
- struct Chain * next

  Pointer to the next node in the set.

4.2.1 Detailed Description

This structure is devoted to define each component of a set

Definition at line 55 of file Variables.h.

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

- /home/migmolper2/NL-PartSol/nl-partsol/include/Variables.h

4.3 ChainPtr Struct Reference

`#include <Variables.h>`

4.3.1 Detailed Description

Pointer to the header of the set

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

- /home/migmolper2/NL-PartSol/nl-partsol/include/Variables.h

4.4 Curve Struct Reference

`#include <Variables.h>`

Data Fields

- int Num
  Number of items in the curve.
- double * Fx
  Values for each time.
- char Info [100]
  Aditional information.

4.4.1 Detailed Description

Curve definition

Definition at line 119 of file Variables.h.

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

- /home/migmolper2/NL-PartSol/nl-partsol/include/Variables.h

4.5 Element Struct Reference

`#include <Variables.h>`

Data Fields

- int i_GP
  Index of the particle.
- int NumberNodes
  Number of nodes close to the particle.
- int * Connectivity
  List of tributary nodes for the particle.

4.5.1 Detailed Description

Structure with the current "element" of the particle

Definition at line 567 of file Variables.h.

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

- /home/migmolper2/NL-PartSol/nl-partsol/include/Variables.h

4.6 Fields Struct Reference

`#include <Variables.h>`

Data Fields

- Matrix rho
  Density field.
- Matrix mass
  Mass field.
- Matrix x_GC
  Position in global coordinates.
- Matrix x_EC
  Position in element coordiantes.
- Matrix dis
  Displacement field.
- Matrix vel
  Velocity field.
- Matrix acc
  Acceleration field.
- Matrix Stress
  Stress field.
- Matrix Strain
  Strain field.
- Matrix StrainF
  Strain during crack.
- Matrix W
  Deformation Energy.
- Matrix ji
  Damage parameter (Fracture)

4.6.1 Detailed Description

Variable devoted to store in the memory the physicl information of each particle

Definition at line 168 of file Variables.h.

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

- /home/migmolper2/NL-PartSol/nl-partsol/include/Variables.h

4.7 GaussPoint Struct Reference

`#include <Variables.h>`

Data Fields

- int NumGP
  Number of particles
- int * I0
  Index with the closest node to each particle.
- int * NumberNodes
  Tributary nodes variables.
- ChainPtr * **ListNodes**
- ChainPtr * Beps
  Set of particles close to each particle.
- Fields Phi
  Store the values of each field in the current time step.
- Fields Phi_n0
  Values from the previous step.
- int NumberMaterials
  Number of materials.
- int * MatIdx
  Index of the material for each particle.
- Material * Mat
  Library of materials.
- int NumNeumannBC
  Number of Neumann boundary conditions.
- Load * F
  Load case of Neumann boundary conditions.
- int NumberBodyForces
  Number of body forces.
- Load * B
  Load case for the body forces.
- Matrix lp
  Size of the voxel for each particle.
- Matrix lambda
  Lagrange multiplier for the LME shape functions.
- Matrix Beta
  Thermalization or regularization parameter for the LME shape functions.

4.7.1 Detailed Description

This structure is devoted to store all the information of a list of particles

Definition at line 382 of file Variables.h.

4.7.2 Field Documentation

4.7.2.1 lp

```
Matrix GaussPoint::lp
```

Size of the voxel for each particle.

Variable for the uGIMP shape function

Definition at line 452 of file Variables.h.

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

- /home/migmolper2/NL-PartSol/nl-partsol/include/Variables.h

4.8 Load Struct Reference

```
#include <Variables.h>
```

Data Fields

- int NumNodes
  Number of nodes/GP with this load.
- int Dim
  Number of dimensions of the load.
- int * Dir
  Direction of the load {0,0} {1,0} {0,1} {1,1}.
- int * Nodes
  List of nodes with this load.
- Curve * Value
  Curve for each dimension with the evolution value with the time.
- char Info [100]
  Some information about this load.

4.8.1 Detailed Description

An important aclaration about this code, a force is a load, or even a boundary condition is defined as a load from the point of view of the code (as a structure). The idea is a force and a boundary condition both of them has a direction, a value, a list of nodes/GPs where it is applied,... So it's a good idea to use this structure for both porpouses.

Definition at line 243 of file Variables.h.

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

- /home/migmolper2/NL-PartSol/nl-partsol/include/Variables.h

4.9 Material Struct Reference

`#include <Variables.h>`

Data Fields

- int Id
  Index of the material.
- char Type [100]
  Name of the material.
- double Cel
  Material celerity.
- double rho
  Initial density.
- double E
  Elastic modulus.
- double mu
  Poisson ratio.
- double thickness
  Thickness of the Material.
- bool Eigenerosion
  Activate eigenerosion-fracture modulus.
- bool Eigensoftening
  Activate eigensoftening-fracture modulus.
- double Ceps
  Normalizing constant (Eigenerosion/Eigensoftening)
- double Gf
  Failure energy (Eigenerosion)
- double ft
  Tensile strengt of the material (Eigensoftening)
- double heps
  Bandwidth of the cohesive fracture (Eigensoftening)
- double Wc
  Critical opening displacement.

4.9.1 Detailed Description

Properties of a material model

Definition at line 302 of file Variables.h.

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

- /home/migmolper2/NL-PartSol/nl-partsol/include/Variables.h

4.10 Matrix Struct Reference

`#include <Variables.h>`

Data Fields

- int N_rows
  Number of rows.
- int N_cols
  Number of columns.
- double n
  Value Is an scalar.
- double * nV
  Pointer for a vector.
- double ** nM
  Table of pointers for a matrix.
- char Info [100]
  Aditional information.

4.10.1 Detailed Description

This structure is devoted to store in matricial the allocated memory

Definition at line 15 of file Variables.h.

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

- /home/migmolper2/NL-PartSol/nl-partsol/include/Variables.h

4.11 Mesh Struct Reference

`#include <Variables.h>`

Data Fields

- int NumNodesMesh

  Number of nodes in the mesh.
- int NumElemMesh

  Number of elements in the mesh.
- Matrix Coordinates

  Table with the coordinates of the nodes of the mesh.
- int * NumNodesElem

  Number of nodes in a element.
- ChainPtr * Connectivity

  List of nodes for each element (Connectivity)
- int * NumNeighbour

  Number of elements close to each node.
- ChainPtr * NodeNeighbour

  List of elements close to each node.
- int * SizeNodalLocality

  Number of nodes close to a node.
- ChainPtr * NodalLocality

  List of nodes close to a node.
- int * NumParticles

  List with the number of particles close to a node.
- ChainPtr * l_particles

  List of particles in a node.
- Boundaries Bounds

  List of boundaries of the domain.
- int Dimension

  Number of dimensions of the element (OLD)
- double DeltaX

  Minimum distance between nodes.
- char TypeElem [20]

  Name of the element (OLD)
- Matrix(* N_ref)(Matrix)

  Function with the interpolation technique.
- Matrix(* dNdX_ref)(Matrix)

  Function with the gradient of the interpolation technique.

4.11.1 Detailed Description

This structure is devoted to store all the information of a list of nodes

Definition at line 472 of file Variables.h.

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

- /home/migmolper2/NL-PartSol/nl-partsol/include/Variables.h

4.12 Table Struct Reference

```
#include <Variables.h>
```

Data Fields

- int N_rows
  Number of rows.
- int N_cols
  Number of columns.
- int n
  Scalar.
- int * nV
  1D list
- int ** nM
  2D list
- char Info [100]
  Aditional information.

4.12.1 Detailed Description

This structure is devoted to store a table with values of the type double.

Definition at line 80 of file Variables.h.

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

- /home/migmolper2/NL-PartSol/nl-partsol/include/Variables.h

4.13 Tensor Struct Reference

`#include <Variables.h>`

Data Fields

- int Order
  Order of the tensor.
- double * n
  First order tensor.
- double * N [3]
  Second order tensor.
- char Info [100]
  Aditional information.

4.13.1 Detailed Description

This structure is devoted to help the code to deal with high level functions for the B-free approach

Definition at line 137 of file Variables.h.

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

- /home/migmolper2/NL-PartSol/nl-partsol/include/Variables.h

4.14 Time_Int_Params Struct Reference

`#include <Variables.h>`

Data Fields

- double GA_alpha

  Generalized alpha parameter alpha.
- double GA_beta

  Generalized alpha parameter alpha.
- double GA_gamma

  Generalized alpha parameter gamma.

4.14.1 Detailed Description

Parameters of the time integration scheme

Definition at line 591 of file Variables.h.

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

- /home/migmolper2/NL-PartSol/nl-partsol/include/Variables.h

Chapter 5

File Documentation

5.1 /home/migmolper2/NL-PartSol/nl-partsol/include/Constitutive.h File Reference

File with the prototype of the contitutive models.

Functions

- Tensor SolidRigid (Tensor Strain)

  This function is devoted to make a material point behaves as a solid rigid.
- Tensor LinearElastic (Tensor Strain, Tensor Stress, Material Mat)

  This function is devoted to make a material point behaves as a linear elastic material.
- void EigenerosionAlgorithm (int p, Matrix ji, Matrix W, Matrix Mass, Matrix Rho, Matrix Stress, Material MatPro, ChainPtr *Beps, double DeltaX)

  Function to compute is a material point is or not eroded. Here the notation is the same as in the paper : A.Pandolfi & M.Ortiz. An eigenerosion approach to brittle fracture. International Journal for Numerical Methods in Enginnering. 92:694-714, 2012.
- void EigensofteningAlgorithm (int, Matrix, Matrix, Matrix, Matrix, Matrix, Material, ChainPtr *)

  Pedro Navas, Rena C.

5.1.1 Detailed Description

File with the prototype of the contitutive models.

5.1.2 Function Documentation

5.1.2.1 EigenerosionAlgorithm()

```
void EigenerosionAlgorithm (
            int p,
            Matrix ji,
            Matrix W,
            Matrix Mass,
            Matrix Rho,
            Matrix Stress,
            Material MatPro,
            ChainPtr * Beps,
            double DeltaX )
```

Function to compute is a material point is or not eroded. Here the notation is the same as in the paper : A.Pandolfi & M.Ortiz. An eigenerosion approach to brittle fracture. International Journal for Numerical Methods in Enginnering. 92:694-714, 2012.

Inputs :

Parameters

p	: Index of the particle
ji	: Damage status
W	: Internal work
Mass	: Mass field
Rho	: Density field
Stress	: Stress field of each particle
Properties	: Define the material properties of the particle
B_eps	: Define the particles close to each particle
DeltaX	: Mesh size

Definition at line 6 of file Fracture.c.

5.1.2.2 EigensofteningAlgorithm()

```
void EigensofteningAlgorithm (
            int ,
            Matrix ,
            Matrix ,
            Matrix ,
            Matrix ,
            Matrix ,
            Material ,
            ChainPtr *  )
```

Pedro Navas, Rena C.

Yu, Bo Li & Gonzalo Ruiz. Modeling the dynamic fracture in concrete: an eigensoftening meshfree approach. International Journal of Impact Engineering. 113 (2018) 9-20 NOTE : Here notation is the same as in the paper. Inputs :

Parameters

Ji_k0	: Matrix with the value of the damage parameter.
Mass	: Matrix with the mass of the GP.
StrainF	: Value of the strain field at the failure init.
Beps	: Table with the list of neighbours per GP.
Neps	: Number of neighbours per GP
Num_GP	: Number of GP of the mesh.

Define auxiliar variable

Material properties of the eigensoftening algorithm

Get the tensile strengt of the material

Get the bandwidth of the cohesive fracture (Bazant)

Get the critical opening displacement

Only for intact particles

Get the number of neighbours

Definition at line 85 of file Fracture.c.

5.1.2.3 LinearElastic()

```
Tensor LinearElastic (
            Tensor Strain,
            Tensor Stress,
            Material Mat )
```

This function is devoted to make a material point behaves as a linear elastic material.

Parameters

Strain	
Stress	
Mat	

Definition at line 4 of file LE.c.

5.1.2.4 SolidRigid()

```
Tensor SolidRigid (
            Tensor Strain )
```

This function is devoted to make a material point behaves as a solid rigid.

Parameters

Strain	

Definition at line 3 of file SolidRigid.c.

5.2 /home/migmolper2/NL-PartSol/nl-partsol/include/Fields.h File Reference

File with the prototype with the function to free memory.

Functions

- Fields **allocate_Fields** (int)
- void **free_Fields** (Fields)

5.2.1 Detailed Description

File with the prototype with the function to free memory.

5.3 /home/migmolper2/NL-PartSol/nl-partsol/include/Formulations.h File Reference

File with the prototype of time integration scheme.

Functions

- double **DeltaT_CFL** (GaussPoint, double)
- void U_FE (Mesh, GaussPoint)

  Displacement formulation with a Forward-Euler integration scheme.
- void U_GA (Mesh, GaussPoint)

  Displacement formulation with a Generalized-alpha integration scheme.
- void U_PCE (Mesh, GaussPoint)

  Explicit predictor corrector.

5.3.1 Detailed Description

File with the prototype of time integration scheme.

5.3.2 Function Documentation

5.3.2.1 U_GA()

```
void U_GA (
            Mesh FEM_Mesh,
            GaussPoint MPM_Mesh )
```

Displacement formulation with a Generalized-alpha integration scheme.

Displacement formulation with a Generalized-alpha integration scheme.

DOI : 10.1002/nme.6138. Tran and Solowski

Definition at line 3 of file U_GA.c.

5.3.2.2 U_PCE()

```
void U_PCE (
            Mesh FEM_Mesh,
            GaussPoint MPM_Mesh )
```

Explicit predictor corrector.

Explicit predictor corrector.

Definition at line 3 of file U_PCE.c.

5.4 /home/migmolper2/NL-PartSol/nl-partsol/include/InOutFun.h File Reference

File with the prototype in/out functions.

Functions

- int **parse** (char $**$, char $*$, char $*$)
- void **generate_route** (char $*$, char $*$)
- Matrix **Read_CSV** (char $*$, int)
- Curve **ReadCurve** (char $*$)
- void **free_Curve** (Curve)
- Mesh **ReadGidMesh** (char $*$)
- ChainPtr **File2Chain** (char $*$)
- Mesh GramsBox (char $*$)
- Boundaries **GramsBoundary** (char $*$, int)
- GaussPoint **GramsSolid2D** (char $*$, Mesh)
- void **GramsTime** (char $*$)
- Material $*$ **GramsMaterials** (char $*$, GaussPoint, int)
- void **GramsInitials** (char $*$, GaussPoint, int)
- void **GramsShapeFun** (char $*$)
- void **GramsOutputs** (char $*$)
- Load $*$ **GramsNeumannBC** (char $*$, int, int)
- Load $*$ **GramsBodyForces** (char $*$, int, int)
- void **WriteVtk_MPM** (char $*$, GaussPoint, Matrix, int)
- void **WriteVtk_FEM** (char $*$, Mesh, Matrix, int)
- void **WriteVtk_Float_Scalar** (char $*$, Matrix)
- void **WriteVtk_Float_Vector** (char $*$, Matrix)
- void **WriteVtk_Float_Tensor** (char $*$, Matrix)
- void **WriteGnuplot** (Matrix, Matrix, double, double, int, int, char[20])

5.4.1 Detailed Description

File with the prototype in/out functions.

5.4.2 Function Documentation

5.4.2.1 GramsBox()

```
Mesh GramsBox (
            char *  )
```

Initialize particle connectivity of each node

Definition at line 8 of file Read_GramsBox.c.

5.5 /home/migmolper2/NL-PartSol/nl-partsol/include/Matlib.h File Reference

File with the prototype of math library.

Macros

- #define **SQR**(a) ((sqr_arg=(a)) == 0.0 ? 0.0 : sqr_arg∗sqr_arg)
- #define **DSQR**(a) ((dsqr_arg=(a)) == 0.0 ? 0.0 : dsqr_arg∗dsqr_arg)
- #define **DMAX**(a, b)
- #define **DMIN**(a, b)
- #define **FMAX**(a, b)
- #define **FMIN**(a, b)
- #define **LMAX**(a, b)
- #define **LMIN**(a, b)
- #define **IMAX**(a, b)
- #define **IMIN**(a, b)
- #define **SIGN**(a, b) ((b) >= 0.0 ? fabs(a) : -fabs(a))

Functions

- Matrix Newton_Rapson (Matrix(∗Function)(Matrix, Matrix), Matrix, Matrix(∗Jacobian)(Matrix, Matrix), Matrix, Matrix, Matrix)

  Newton-Rapson method to solve non-linear sistems of equations : Y = Y(X) -> We solve -> F(X) = Y - Y(X) = 0 F(X + DeltaX) = F(X) + J(X)∗DeltaX = 0 —> DeltaX = - J(X)^{-1}∗F(X) Inputs :
- Matrix **Solve_Linear_Sistem** (Matrix, Matrix)
- Matrix Conjugate_Gradient_Method (Matrix, Matrix, Matrix)

  Practical aspects of the finite element method.
- Matrix Jacobi_Conjugate_Gradient_Method (Matrix, Matrix, Matrix)

  To increase the rate of convergence of Conjugate_Gradient_Method(), preconditioning is used.
- Matrix **One_Iteration_Lumped** (Matrix, Matrix, Matrix)
- void ∗ **Allocate_Array** (int, int)
- void ∗ **Allocate_ArrayZ** (int, int)
- void ∗∗ **Allocate_Matrix** (int, int, int)
- void ∗∗ **Allocate_MatrixZ** (int, int, int)
- Matrix **MatAlloc** (int, int)
- Matrix **MatAllocZ** (int, int)
- Matrix **MatAssign** (int, int, double, double ∗, double ∗∗)
- void **FreeMat** (Matrix)
- void **PrintMatrix** (Matrix, int, int)
- double **StatsDouMatrix** (double ∗, int, char ∗)
- double **StatsIntMatrix** (int ∗, int, char ∗)
- Matrix **CopyMat** (Matrix)
- double **Norm_Mat** (Matrix, int)
- double **Cond_Mat** (Matrix, double)
- double **Get_Determinant** (Matrix)
- Matrix **Get_Inverse** (Matrix)
- Matrix **Transpose_Mat** (Matrix)
- Matrix **Scalar_prod** (Matrix, Matrix)
- Matrix **get_A_dot_B_Mat** (Matrix, Matrix)
- Matrix **get_a_dot_b_Mat** (Matrix, Matrix)
- Matrix **get_A_dot_b_Mat** (Matrix, Matrix)
- Matrix **get_a_dot_B_Mat** (Matrix, Matrix)
- Matrix Matrix_x_Scalar (Matrix, double)

  Brief description of Matrix_x_Scalar.
- Matrix **Vectorial_prod** (Matrix, Matrix)
- Matrix **Tensorial_prod** (Matrix, Matrix)
- Matrix **Incr_Mat** (Matrix, Matrix)
- Matrix **Add_Mat** (Matrix, Matrix)
- Matrix **Sub_Mat** (Matrix, Matrix)
- Matrix **Get_Lumped_Matrix** (Matrix)
- double Area_Poligon (Matrix)

  This function returns the the area of a Poligon using the Shoelace formula : https://en.wikipedia.org/wiki/Shoelace_formula.
- Matrix Centroid_Poligon (Matrix)

  This function returns the centroid of a Poligon and the area.
- int InOut_Poligon (Matrix, Matrix)

  Check if a point is or not (1/0) inside of a Poligon.
- double **SignumFunct** (double x)
- Matrix **SolvePolynomial** (Matrix)
- Matrix **get_nurbs_distance** (Matrix)
- double **Distance** (Matrix, Matrix)
- void **get_SVD_Of** (Matrix A, Matrix W, Matrix V)

- ChainPtr **Pointer_to_Set** (int ∗, int)
- int ∗ **Set_to_Pointer** (ChainPtr, int)
- ChainPtr **RangeChain** (int, int)
- void **free_Set** (ChainPtr)
- void **free_SetTable** (ChainPtr ∗, int)
- bool **is_in_Set** (ChainPtr, int)
- void **push_to_Set** (ChainPtr ∗, int)
- void **pop_from_Set** (ChainPtr ∗, int)
- ChainPtr **CopyChain** (ChainPtr)
- int **get_Lenght_Set** (ChainPtr)
- ChainPtr **get_Union_Of** (ChainPtr ∗, int)
- ChainPtr **get_Intersection_Of** (ChainPtr, ChainPtr)
- void **print_Set** (ChainPtr)
- void **order_Set** (ChainPtr ∗, ChainPtr ∗, Matrix)
- Tensor **alloc_Tensor** (int Order)
- Tensor **memory_to_Tensor** (double ∗A_mem, int Order)
- void **free_Tensor** (Tensor A)
- double **get_I1_Of** (Tensor A)
- double **get_I2_Of** (Tensor A)
- double **get_I3_Of** (Tensor A)
- double **get_J1_Of** (Tensor A)
- double **get_J2_Of** (Tensor A)
- double **get_J3_Of** (Tensor A)
- Tensor **get_Eigenvalues_Of** (Tensor)
- double **get_EuclideanNorm_Of** (Tensor A)
- Tensor **get_I** ()
- Tensor **get_Inverse_Of** (Tensor A)
- Tensor **get_Transpose_Of** (Tensor A)
- double **get_innerProduct_Of** (Tensor A, Tensor B)
- Tensor **get_vectorProduct_Of** (Tensor a, Tensor b)
- Tensor **get_dyadicProduct_Of** (Tensor a, Tensor b)
- Tensor **get_firstOrderContraction_Of** (Tensor A, Tensor b)

5.5.1 Detailed Description

File with the prototype of math library.

5.5.2 Macro Definition Documentation

5.5.2.1 DMAX

```
#define DMAX(
              a,
              b )
```

Value:

```
(dmax_arg1=(a),dmax_arg2=(b),(dmax_arg1) > (dmax_arg2) ? \
(dmax_arg1) : (dmax_arg2))
```

Definition at line 15 of file Matlib.h.

5.5.2.2 DMIN

```
#define DMIN(
               a,
               b )
```

Value:

```
(dmin_arg1=(a),dmin_arg2=(b),(dmin_arg1) < (dmin_arg2) ? \
(dmin_arg1) : (dmin_arg2))
```

Definition at line 18 of file Matlib.h.

5.5.2.3 FMAX

```
#define FMAX(
               a,
               b )
```

Value:

```
(max_arg1=(a),max_arg2=(b),(max_arg1) > (max_arg2) ? \
(max_arg1) : (max_arg2))
```

Definition at line 21 of file Matlib.h.

5.5.2.4 FMIN

```
#define FMIN(
               a,
               b )
```

Value:

```
(min_arg1=(a),min_arg2=(b),(min_arg1) < (min_arg2) ? \
(min_arg1) : (min_arg2))
```

Definition at line 24 of file Matlib.h.

5.5.2.5 IMAX

```
#define IMAX(
               a,
               b )
```

Value:

```
(imax_arg1=(a),imax_arg2=(b),(imax_arg1) > (imax_arg2) ? \
(imax_arg1) : (imax_arg2))
```

Definition at line 33 of file Matlib.h.

5.5.2.6 IMIN

```
#define IMIN(
               a,
               b )
```

Value:

```
          (imin_arg1=(a),imin_arg2=(b),(imin_arg1) < (imin_arg2) ? \
          (imin_arg1) : (imin_arg2))
```

Definition at line 36 of file Matlib.h.

5.5.2.7 LMAX

```
#define LMAX(
               a,
               b )
```

Value:

```
          (lmax_arg1=(a),lmax_arg2=(b),(lmax_arg1) > (lmax_arg2) ? \
          (lmax_arg1) : (lmax_arg2))
```

Definition at line 27 of file Matlib.h.

5.5.2.8 LMIN

```
#define LMIN(
               a,
               b )
```

Value:

```
          (lmin_arg1=(a),lmin_arg2=(b),(lmin_arg1) < (lmin_arg2) ? \
          (lmin_arg1) : (lmin_arg2))
```

Definition at line 30 of file Matlib.h.

5.5.3 Function Documentation

5.5.3.1 Conjugate_Gradient_Method()

```
Matrix Conjugate_Gradient_Method (
            Matrix K,
            Matrix F,
            Matrix U0 )
```

Practical aspects of the finite element method.

M.Pastor, P.Mira, J.A.Fernández Merodo

DOI : 10.1080/12795119.2002.9692737

Section 6.1 : Conjugate Gradient Method with Preconditioning One of the most effective and simple iterative methods (when used with preconditioning) for solving $Ax = b$ is the conjugate gradient algorithm. The algorithm is based on the idea that the solution $Ax = b$ minimizes the total potential $\Pi = \frac{1}{2} x^T A x - x^T b $. Hence, the task in the iteration is, given an approximate x^k to x for wich the potential is Π^k, to find an improved aproximation x^{k+1} for wich $\Pi^{k+1}<\Pi^k$. Howerver, not only do we want the total potential to decrease each iteration but we also want the total potential to decrease in each iteration but we also want x^{k+1} to be calculate efficiently and the decrease in the total potential to occur rapidly. Then the iteration will converge fast.

In the conjugate gradient method, we use in the kth iteration the linearly independent vectors $p^1,p^2,p^3,\ldots,p^k$ and calculate the minimum of the potential in the space of the potential in the space spanned by these vectors. This gives x^{k+1}. Also, we stablish the additional basis vector p^{k+1} used in the subsequent iteration.

The algorithm can be summarized as follows: \begin{enumerate}

\item Choose the iteration vector x^1 (frequently x^1 is the null vector).

\item Calculate the residual $r^1 = b - Ax^1$. If $r^1 = 0$, quit.

\item Else : \begin{enumerate} \item $p^1 = r^1$ \item Calculate for $k = 1,2, \ldots$ \begin{equation} \alpha^k = \frac{r^{k^T} r^k}{p^{k^T} A p^k} x^{k+1} = x^k + \alpha^k p^k r^{k+1} = r^k - \alpha^k A p^k \beta^k = \frac{r^{k+1^T} r^{k+1}}{r^{k^T} A r^k} p^{k+1} = p^{k+1} + \beta^k p^k \end{equation} \end{enumerate}

\end{enumerate}

We continue iterating until $||r^k|| \leseq \epsilon$, where ϵ in the convergence tolerance. A convergence criterion on $||x^k||$ could also be used.

The conjugate gradient algorithm satisfies two important orthogonality properties regarding the direction vectors p_i and the residual r_i

See also : https://en.wikipedia.org/wiki/Conjugate_gradient_method#The_preconditioned_conjugate_gradient_method

Definition at line 129 of file Solvers.c.

5.5.3.2 InOut_Poligon()

```
int InOut_Poligon (
             Matrix X_Point,
             Matrix Poligon )
```

Check if a point is or not (1/0) inside of a Poligon.

Inputs :

- *X_Point* : Coordinates of the point
- *Poligon* : Coordinates of the vertex 0,1,....,n,0

Definition at line 111 of file MathOp.c.

5.5.3.3 Jacobi_Conjugate_Gradient_Method()

```
Matrix Jacobi_Conjugate_Gradient_Method (
             Matrix K,
             Matrix F,
             Matrix U0 )
```

To increase the rate of convergence of Conjugate_Gradient_Method(), preconditioning is used.

The basic idea is that instead of solving $K U = F$, we solve : \begin{equation} \tilde{K}^{-1}K U =\tilde{K}^{-1}F \end{equation} where $\tilde{K}$ is called the preconditioner. The objective with this transformation is to obtain a matrix $\tilde{K}^{-1}K$ with a much improved conditioned number choosing an easy inverting matrix $tilde{A}$. Various preconditioners have been proposed, the the choose of the diagonal part of K results in the Jacoby Conjugate method (JCG). The new algorithm introduces an additional set of vectors z^k defined by: \begin{equation} z^k = \tilde{K}^{-1} r^k \end{equation} who modifies the definition of α^k, β^k, p^k : \begin{equation} \alpha^k = \frac{z^k^T r^k}{p^k^T A p^k} \beta^k = \frac{z^{k+1^T} r^{k+1}}{z^{k^T} r^k} p^{k+1} = z^{k+1} + \beta^k p^k \end{equation}

Definition at line 333 of file Solvers.c.

5.5.3.4 Matrix_x_Scalar()

```
Matrix Matrix_x_Scalar (
             Matrix A,
             double a )
```

Brief description of Matrix_x_Scalar.

Function to multiply a Matrix with a scalar.

The parameters for this functions are :

Parameters

A	: Input Matrix
a	: Input scalar

Definition at line 1127 of file MatrixOp.c.

5.5.3.5 Newton_Rapson()

```
Matrix Newton_Rapson (
             Matrix(*)(Matrix, Matrix) Function,
             Matrix Parameter_F,
             Matrix(*)(Matrix, Matrix) Jacobian,
             Matrix Parameter_J,
             Matrix Y,
             Matrix X )
```

Newton-Rapson method to solve non-linear sistems of equations : Y = Y(X) -> We solve -> F(X) = Y - Y(X) = 0 F(X + DeltaX) = F(X) + J(X)∗DeltaX = 0 —> DeltaX = - J(X)^{-1}∗F(X) Inputs :

- Y : Value of the function
- Function(X,Parameter_F) : Pointer to function to solve
- Parameter_F : F function optional parameters
- Jacobian(X,Parameter_J) : Pointer to the jacobian of the function
- Parameter_J : Jacobian optional parameters
- X : Initial value of the objetive

Definition at line 14 of file Solvers.c.

5.6 /home/migmolper2/NL-PartSol/nl-partsol/include/MPM.h File Reference

File with the prototype of the material point functions/utilities.

Functions

- Curve **BcDirichlet** (char *)
- void **imposse_NodalMomentum** (Mesh, Matrix, int)
- void **imposse_NodalVelocity** (Mesh, Matrix, int)
- Matrix **Eval_Body_Forces** (Load *, int, int, int)
- Matrix **Eval_Contact_Forces** (Load *, int, int, int)
- Matrix **compute_Reactions** (Mesh, Matrix)
- void **free_Load** (Load)
- void **free_LoadTable** (Load *)
- void **free_Boundaries** (Boundaries)
- void **ComputeDamage** (int, GaussPoint, Mesh)
- Tensor **compute_RateOfStrain** (Matrix, Matrix)
- Tensor **update_Strain** (Tensor, Tensor, double)
- double **update_Density** (double, double, Tensor)
- Tensor **compute_Stress** (Tensor, Tensor, Material)
- void **update_LocalState** (Matrix, GaussPoint, Mesh, double)
- double **compute_InternalEnergy** (Tensor, Tensor)
- Matrix **compute_InternalForces** (Matrix, GaussPoint, Mesh)
- Matrix **compute_BodyForces** (Matrix, GaussPoint, Mesh, int)
- Matrix **compute_ContacForces** (Matrix, GaussPoint, Mesh, int)
- Matrix **compute_equilibrium_U** (Matrix, GaussPoint, Mesh, double)
- Matrix **compute_NodalMomentumMass** (GaussPoint, Mesh)
- Matrix **compute_NodalVelocity** (Mesh, Matrix)
- void update_NodalMomentum (Mesh, Matrix, Matrix)

  Brief description of UpdateGridNodalMomentum.
- void **update_Particles_FE** (GaussPoint, Mesh, Matrix, Matrix, double)
- Matrix **compute_NodalMass** (GaussPoint, Mesh)
- Matrix **compute_VelocityPredictor** (GaussPoint, Mesh, Matrix, Matrix, Time_Int_Params, double)
- Matrix **compute_VelocityCorrector** (Mesh, Matrix, Matrix, Matrix, Time_Int_Params, double)
- void **update_Particles_PCE** (GaussPoint, Mesh, Matrix, Matrix, Matrix, double)
- void GA_UpdateNodalKinetics (Mesh, Matrix, Matrix, Time_Int_Params)

  Brief description of Update_Nodal_Acceleration_Velocity.
- Matrix GetNodalKinetics (GaussPoint, Mesh)

  Nodal_Kinetics = {mass, a0, a1, v}.
- Matrix GetNodalVelocityDisplacement (GaussPoint, Mesh)

  Nodal_Kinetics = {m, d, v}.
- void **update_Particles_GA** (GaussPoint, Mesh, Matrix, Time_Int_Params)
- void **GetInitialGaussPointPosition** (Matrix, Mesh, int)
- double **GetMinElementSize** (Mesh)
- void **GetNodalConnectivity** (Mesh)
- ChainPtr **DiscardElements** (ChainPtr, Matrix, Matrix, Mesh)
- void **LocalSearchGaussPoints** (GaussPoint, Mesh)
- void **ComputeBeps** (int, GaussPoint, Mesh)
- void **GPinCell** (ChainPtr *, ChainPtr *, Matrix, int, double)
- Element **get_Element** (int, ChainPtr, int)
- Matrix **get_set_Coordinates** (ChainPtr, Matrix, Matrix)
- Matrix **get_Element_Field** (Matrix, Element)
- ChainPtr **get_locality_of_node** (int, Mesh)
- int **get_closest_node_to** (Matrix, ChainPtr, Matrix)
- bool **InOut_Element** (Matrix, ChainPtr, Matrix)
- int **search_particle_in** (int, Matrix, ChainPtr, Mesh)
- Matrix **ElemCoordinates** (ChainPtr, Matrix)
- void **asign_particle_to_nodes** (int, ChainPtr, Mesh)

5.6.1 Detailed Description

File with the prototype of the material point functions/utilities.

5.6.2 Function Documentation

5.6.2.1 GA_UpdateNodalKinetics()

```
void GA_UpdateNodalKinetics (
              Mesh FEM_Mesh,
              Matrix Nodal_Kinetics,
              Matrix Nodal_Forces,
              Time_Int_Params Params )
```

Brief description of Update_Nodal_Acceleration_Velocity.

The parameters for this functions are :

Parameters

FEM_Mesh	
Nodal_Kinetics	= {m, a0, a1, v}
Nodal_Forces	
Params	

Definition at line 306 of file Update_Eulerian.c.

5.6.2.2 update_NodalMomentum()

```
void update_NodalMomentum (
              Mesh FEM_Mesh,
              Matrix Phi_I,
              Matrix F_I )
```

Brief description of UpdateGridNodalMomentum.

Compute the nodal contribution of each GP to the total forces.

The parameters for this functions are :

Parameters

MPM_Mesh	: Mesh with the material points.
Phi_I	{P_I | M_I}
F_I	: Nodal value of the total forces.

Definition at line 115 of file Update_Eulerian.c.

5.7 /home/migmolper2/NL-PartSol/nl-partsol/include/ShapeFun.h File Reference

File with the prototype of the interpolation techniques here adopted.

Functions

- Matrix **L2** (Matrix)
- Matrix **dL2** (Matrix)
- Matrix **Get_F_Ref_L2** (Matrix, Matrix)
- Matrix **Get_X_GC_L2** (Matrix, Matrix)
- Matrix **T3** (Matrix)
- Matrix **dT3** (Matrix)
- Matrix **Get_F_Ref_T3** (Matrix, Matrix)
- Matrix **Get_dNdX_T3** (Matrix, Matrix)
- Matrix **Get_X_GC_T3** (Matrix, Matrix)
- void **Get_X_EC_T3** (Matrix, Matrix, Matrix)
- void **Q4_Initialize** (GaussPoint, Mesh)
- Matrix **Q4_N** (Matrix)
- Matrix **Q4_dN_Ref** (Matrix)
- Matrix **Q4_F_Ref** (Matrix, Matrix)
- Matrix **Q4_dN** (Matrix, Matrix)
- Matrix **Q4_Xi_to_X** (Matrix, Matrix)
- void **Q4_X_to_Xi** (Matrix, Matrix, Matrix)
- void **uGIMP_Initialize** (GaussPoint, Mesh)
- double **uGIMP_Sip** (double, double, double)
- double **uGIMP_dSip** (double, double, double)
- Matrix **uGIMP_N** (Matrix, Matrix, double)
- Matrix **uGIMP_dN** (Matrix, Matrix, double)
- ChainPtr **uGIMP_Tributary_Nodes** (Matrix, int, Matrix, Mesh)
- void LME_Initialize_Beta (Matrix, double, int)

  Shape functions based in : "" Local maximum-entropy approximation schemes : a seamless bridge between finite elements and meshfree methods "" by M.Arroyo and M.Ortiz, 2006.
- void **LME_Initialize** (GaussPoint, Mesh)
- Matrix LME_Beta (Matrix, Matrix, double)

  Function to update the value of beta.
- Matrix LME_lambda_NR (Matrix, Matrix, Matrix)

  Get the lagrange multipliers "lambda" (1 x dim) for the LME shape function.
- double LME_fa (Matrix, Matrix, Matrix)

  Output : -> fa : the function fa that appear in [1] (scalar).
- Matrix LME_p (Matrix, Matrix, Matrix)

  Get the value of the shape function "pa" (1 x neighborhood) in the neighborhood nodes.
- Matrix LME_r (Matrix, Matrix)

  Get the gradient "r" (dim x 1) of the function log(Z) = 0.
- Matrix LME_J (Matrix, Matrix, Matrix)

  Get the Hessian "J" (dim x dim) of the function log(Z) = 0.
- Matrix LME_dp (Matrix, Matrix)

  Value of the shape function gradient "dp" (dim x neighborhood) in the neighborhood nodes.
- ChainPtr **LME_Tributary_Nodes** (Matrix, Matrix, int, Mesh)
- Matrix **compute_ShapeFunction** (Element, GaussPoint, Mesh)
- Matrix **compute_ShapeFunction_Gradient** (Element, GaussPoint, Mesh)

5.7.1 Detailed Description

File with the prototype of the interpolation techniques here adopted.

5.7.2 Function Documentation

5.7.2.1 LME_dp()

```
Matrix LME_dp (
             Matrix l,
             Matrix p )
```

Value of the shape function gradient "dp" (dim x neighborhood) in the neighborhood nodes.

Input parameters : -> l : Matrix with the distances to the neighborhood nodes (neighborhood x dim). -> p : Shape function value in the neighborhood nodes (neighborhood x 1).

Definition at line 375 of file LME.c.

5.7.2.2 LME_fa()

```
double LME_fa (
             Matrix la,
             Matrix lambda,
             Matrix Beta )
```

Output : -> fa : the function fa that appear in [1] (scalar).

Input parameters : -> la : Matrix with the distance to the neighborhood node "a" (1 x dim). -> lambda : Initial value of the lagrange multipliers (dim x 1). -> Gamma : Tunning parameter (scalar).

Definition at line 242 of file LME.c.

5.7.2.3 LME_Initialize_Beta()

```
void LME_Initialize_Beta (
            Matrix Beta,
            double DeltaX,
            int Np )
```

Shape functions based in : "" Local maximum-entropy approximation schemes : a seamless bridge between finite elements and meshfree methods "" by M.Arroyo and M.Ortiz, 2006.

Here we employ the same nomenclature as in the paper. With the single different of the "l" variable wich represents the distances between the evaluation point and the neighborhood nodes.

List of functions :

- LME_Init_lambda
- LME_lambda_NR
- LME_fa
- LME_p
- LME_r
- LME_J
- LME_dp
- LME_Tributary_Nodes

Function to get a initial value of Beta

Definition at line 31 of file LME.c.

5.7.2.4 LME_J()

```
Matrix LME_J (
            Matrix l,
            Matrix p,
            Matrix r )
```

Get the Hessian "J" (dim x dim) of the function log(Z) = 0.

Input parameters : -> l : Matrix with the distances to the neighborhood nodes (neighborhood x dim). -> p : Shape function value in the neighborhood nodes (neighborhood x 1). -> r : Gradient of log(Z) (dim x 1).

Definition at line 337 of file LME.c.

5.7.2.5 LME_lambda_NR()

```
Matrix LME_lambda_NR (
             Matrix l,
             Matrix lambda,
             Matrix Beta )
```

Get the lagrange multipliers "lambda" (1 x dim) for the LME shape function.

The numerical method for that is the Newton-Rapson.

Input parameters : -> l : Matrix with the distances to the neighborhood nodes (neighborhood x dim). -> lambda : Initial value of the lagrange multipliers (1 x dim). -> Beta : Tunning parameter (scalar). -> h : Grid spacing (scalar). -> TOL_zero : Tolerance for Newton-Rapson.

Definition at line 158 of file LME.c.

5.7.2.6 LME_p()

```
Matrix LME_p (
             Matrix l,
             Matrix lambda,
             Matrix Beta )
```

Get the value of the shape function "pa" (1 x neighborhood) in the neighborhood nodes.

Input parameters : -> l : Matrix with the distances to the neighborhood nodes (neiborghood x dim). -> lambda : Initial value of the lagrange multipliers (1 x dim). -> Beta : Tunning parameter (scalar).

Definition at line 265 of file LME.c.

5.7.2.7 LME_r()

```
Matrix LME_r (
             Matrix l,
             Matrix p )
```

Get the gradient "r" (dim x 1) of the function log(Z) = 0.

Input parameters : -> l : Matrix with the distances to the neighborhood nodes (neighborhood x dim). -> p : Shape function value in the neighborhood nodes (1 x neighborhood).

Definition at line 308 of file LME.c.

5.8 /home/migmolper2/NL-PartSol/nl-partsol/include/Variables.h File Reference

File with the definitions of the user-defined variables.

Data Structures

- struct Matrix
- struct Chain
- struct Table
- struct Curve
- struct Tensor
- struct Fields
- struct Load
- struct Boundaries
- struct Material
- struct GaussPoint
- struct Mesh
- struct Element
- struct Time_Int_Params

Typedefs

- typedef struct Chain **Chain**
- typedef Chain * **ChainPtr**

5.8.1 Detailed Description

File with the definitions of the user-defined variables.

Index