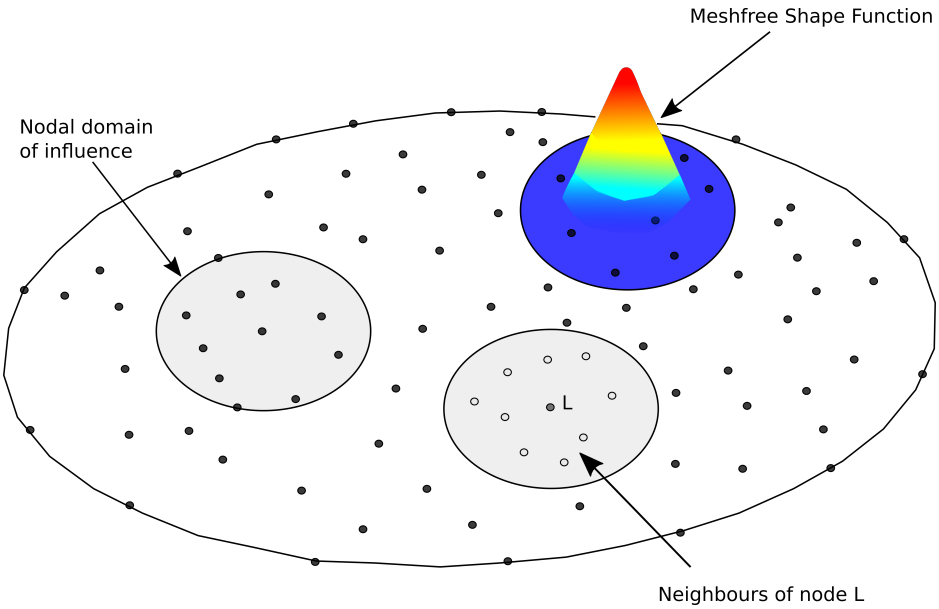

MFree: An Explicit Meshfree Code for Non-Linear Solid Mechanics release 0.1

Stephen Smith
Queen's University Belfast

2018
October

Contents

Chapter 1

Introduction

1.1 MFree

The Mfree libary is a framework developed in go-lang for mesh-free modelling in a research environment. It was developing during my PhD for application to stretch blow moulding a manufacturing technique used to produce polymer bottles. The main features of the code are the ability to simulate non-linear solid mechanics problems using an explicit solver. The use of the MFree library is intended to simplify the use of mesh-free methods within a research context, and further provide a learning resource for meshfree methods.

To facilitate this goal, this manuscript has been created in order to describe the main features of this code. The code has been intended to be designed around the idea of the interaction between objects, similar to the object oriented style of programming, without the complexities of inheritance and polymorphism present in other langauges such as C++ and Java. Hence, it is the aim of this code is to provide a balance between a user friendly black-box (similar to Abaqus) and a tool for researchers within computational mechanics.

The go-lang langauge has been chosen as it provides inbuilt conncurncy along with a readability, and usability often not found in other high performance langauges (C, Fortran, C++).

1.2 Overview

Chapter 2

Theory

In order to understand how to use the library it is necessary to have an appreciation of the theory behind the application of meshfree methods to non-linear solid mechanics. This chapter explains the fundamental ideas of meshfree methods, and discretization of a continous problem in a discrete, and numerically solvable one.

2.1 Meshfree methods

Meshfree methods were developed in the middle of the 1990's to overcome difficulties associated with the finite element method. The fundamental problem in meshfree or finite elements is to provide a set of approximation function (subject to a set of constraints) that can be used as an approximation space for the trial and test functions in the weak form of a partial different equation. To illustrate this problem, consider the balance of linear momentum, cast in a Lagrangian(reference) form Eq. (2.1). The intention in this manuscript is to use capital symbols to describe material coordinates, however often irregularities will exist between this intention and the result.

$$\text{GRAD } P_{iJ} = \rho_0 \ddot{u}_i \tag{2.1}$$

where P is the first Piola-Kirchoff stress, ρ_0 the material density in the reference frame (typically $t = 0$) and $\ddot{u}$ the acceleration. To complete the balance of linear moemntum boundary conditions must be specified. A boundary is called a displacemnt boundary, denoted Γ_u, if a displacement u is prescribed on that boundary, likewise a similar defintion is present for the traction

boundary, denoted Γ_t The boundary conditions for Eq. (2.1) are typically given as:

$$u_i = \bar{u_i} \quad on \quad \Gamma_u \tag{2.2}$$

$$P_{ij}N_j = \bar{T_i} \quad on \quad \Gamma_t \tag{2.3}$$

Equation 2, subject to the boundary conditions (2.2,2.3) is often termed the strong form

Strong form of the momentum balance

$$\begin{aligned} \text{GRAD } P_{iJ} &= \rho_0 \ddot{u}_i && \text{Momentum Balance} \\ u_i &= \bar{u_i} \quad on \quad \Gamma_u && \text{Displacement condition} \\ P_{ij}N_j &= \bar{T_i} \quad on \quad \Gamma_t && \text{Traction condition} \end{aligned}$$

Strong form to weak form

To construct the weak form a test function δu is introduced, which is assumed to be smooth enough up to the required order of the problem, and vanishes on the displacement boundary. The strong form, denoted $\mathcal{S}(u)$ of Box 1 is now cast into an alternate, but equivalent problem:

Given a strong form $\mathcal{S}(u)$, find the trial dispalcemnt field $u(X,t)$ such that the following equation is satisfied

$$\int \delta u_i \cdot [GRAD \ P_{iJ} - \rho_0 \ddot{u}] d\Omega = 0 \tag{2.4}$$

which is obtained from integrating the product of the strong form $S(u)$ and the test functions δu. Expanding (2.4) using the product rule of integration leads to the following form:

$$\int \big(\text{GRAD } \delta u_i P_{iJ} + \delta u_i \rho_0 \ddot{u}_i\big) d\Omega - \int \bar{T_i} \delta u_i d\Gamma_t = 0 \tag{2.5}$$

This form of the equation can conveniently (for the sake of physical interpretation) be cast into the the principle of virtual work:

$$\delta W = \delta W^{int} - \delta W^{ext} + \delta W^{kin} \tag{2.6}$$

Before summarising the derivation it is necessary to make a note on the definition of the trial and test functions. Firstly as mentioned above the test functions δ_u should vanish on the displacement boundary, and be continuous up to order required, which in the above form requires the existence of the first derivatives. With reference to this we define the space of functions $\mathcal{U}_0$ that satisfy these conditions:

$$\delta u(X) \in \mathcal{U}_0 \text{ where } \mathcal{U}_0 = \{\delta u(X) | \delta u(X) \in C^0(X), \delta u = 0 \text{ on } \Gamma_u\} \quad (2.7)$$

In a similar manner the functions space for the trial functions $u(X,t)$ can be defined, with the condition that the displacemnt boundary conditions are satisfied.

$$u(X,t) \in \mathcal{U} \text{ where } \mathcal{U} = \{u(X,t) | u(X,t) \in C^0(X), u(X,t) = \bar{u} \text{ on } \Gamma_u\} \quad (2.8)$$

Combining these definitions with the principle of virtual work above leads to the complete problem, given by:

Principle of virtual work (Weak form) Find the trial functions $u(X,t)$ such that for any admissible (member of $\mathcal{U}_0$) virtual displacement δu the virtual work δW is zero.

$$\delta W = \delta W^{int} - \delta W^{ext} + \delta W^{kin}$$

where

$$\delta W^{int} = \int \text{GRAD } \delta u_i P_{iJ} d\Omega$$

$$\delta W^{ext} = \int \bar{T}_i \delta u_i d\Gamma_t$$

$$\delta W^{kin} = \int \delta u_i \rho_0 \ddot{u}_i d\Omega$$

Discretization

The weak form in Box 2 still requires the determination of the continuous function $u(X,t)$ which in most cases will be impossible. A set of discrete equations can be developed by considering an approximation of the displacements, which in its most general form: *Given a set of data points* $X \in \mathcal{R}^d$,

with nodal values u_d. Any approximation scheme is subject to a minimum of two constraints in order to be applied to the virtual work statement above:

Shape function requirements

1. The shape functions should be able to reproduce a constant field

$$\Longrightarrow \sum_I \phi_I = 1$$

2. In order to ensure first order convergence the shape functions should be able to reproduce a polynomial

$$\Longrightarrow \sum_I \phi_I x_{kI} = x_k \quad for\ k = 1, 2, 3$$

This leads to the following form

$$u(X,t) = \sum_I^N \phi_I(X) u_{Ii}(t) \tag{2.9}$$

and the derivative by:

$$u(X,t)_{,J} = \sum_I^N \phi_I(X)_{,J}\, u_{Ii}(t) \quad where\ j = 1, .., n_d \tag{2.10}$$

The construction of the shape functions ϕ_I is dependent on the method used. If the data points X are arranged into convex polygons, then the shape functions coincide with those used in the finite element method. However, this predefined computational mesh can create issues in large deformation, which lead to the develop of meshfree approximation methods. In this case an arbitrary, but local support is assigned to each node Fig. 1. The simplest shape function that can be constructed for this domain is the shepard function, defined by a ratio of the weight functions

$$\phi_i = \frac{\omega_a(x; x - x_i)}{\sum_j^n \omega_a(x; x - x_j)} \tag{2.11}$$

which clearly satisfies the first condition (constant reproduction) as

$$\sum_i \phi_i = \frac{\sum_i^n \omega_a(x; x - x_i)}{\sum_j^n \omega_a(x; x - x_j)} = 1 \tag{2.12}$$

However, in order to satisfy the linear reproducing conditions an additional enrichment term is required. This is typically ahcieved by introducing a monomial basis terms into the approximation, yieding a general form of meshfree shape functions

$$u^h(x) = \sum_i^n C_i(x)\Gamma_i(x) \tag{2.13}$$

where ω is a weight function, and C_I corrective terms, which meet the reproducing conditions. However, in order to satisfy the linear reproducing conditions an additional enrichment term is required. Consider the general form of the meshfree aporixmation given by However, in order to satisfy the linear reproducing conditions an additional enrichment term is required. Consider the general form of the meshfree aporixmation given by However, in order to satisfy the linear reproducing conditions an additional enrichment term is required. Consider the general form of the meshfree aporixmation given by

Probabilistic Approach

The approach above for developing meshfree shape functions leads to one drawback, the shape functions no longer interpolate data, which presents problems in implementing boundary conditions. In order to fix this issue a probablistic approach to mesh-free shape functions was developed by Ortiz [], known as the MAXENT scheme. In this approach we consider the shape function approximation form $u^h(x) = \phi_I u_I$ and associate the shape function with a probability, we interpret the shape functions ϕ_I as *the probability p_I to which point I influences x*, then the shape function approximation gives the expected value $\sum \phi_I u_I = \mathcal{E}(u)$. In order to find the probablity distrubitoon (or shape functions) the principle of maximum entropy is used, which states for a given probability distribution ϕ_i with $\sum_i \phi_i = 1$ then the least biased distribution is the one that maximises information entropy is given by

Principle of maximum entropy (MAXENT):

$$\max_{\phi_I} \quad H(\phi_i) = -\sum_i \phi_i log(\phi_i)$$

subject to the reproducing constraints (in 2D):

$$\sum_I \phi_I = 1, \qquad \sum_I \phi_I x_{Ik} = x_k,$$

The constaints are added to the MAXENT principle uisng Lagrange multipliers, which produces the following Lagrangian:

$$\delta \left[\sum_I -\phi_I log \phi_I + \lambda_0 (1 - \sum_I \phi_i) + \lambda_k (x_k - \sum \phi_I x_{Ik}) \right] = \mathbf{0} \tag{2.14}$$

simplifying 2.14 and using the relation $\lambda_0 = log(Z) - 1$ yields the following:

$$\phi_I = \frac{e^{-(\lambda_k x_{Ik})}}{Z}, \quad Z(\lambda_k) = \sum_I e^{-(\lambda_k x_{Ik})} \tag{2.15}$$

subject to the k constraints:

$$f_k(\lambda_k) = \frac{\sum e^{-(\lambda_k x_{Ik})}}{Z} - x_k \ \text{ for } k = (1, ..., nd) \tag{2.16}$$

For numerical reasons (give some?) introducing the shifted coordinates $\tilde{x_{Ik}} = x_{Ik} - x_k$ is advantageus as it casts the maximisation into the dual problem of minimization. In the shifted coordinates the k constraints are:

$$\sum_I \phi_I \tilde{x}_{kI} = 0 \tag{2.17}$$

and ϕ and $\tilde{Z}$

$$\phi_I = \frac{e^{-(\lambda_k \tilde{x}_{Ik})}}{\tilde{Z}}, \qquad \tilde{Z} = \sum_I e^{-(\lambda_k \tilde{x}_{Ik})} \tag{2.18}$$

Substitution of Eqs 2.18 into the MAXENT principle yields:

$$H_{max} = log(\tilde{Z}) \tag{2.19}$$

subject to the constraints

$$\frac{\partial log(\tilde{Z})}{\partial \lambda_k} = 0 \tag{2.20}$$

Hence, the solution to this problem is as follows: *Find the set of Lagrange multipliers (λ_k) that minimise the convex potential function $F(\lambda_k) = H_{max}$.*

This problem is solved iteratively using schemes such as *steepest-descent method* or *Newton-Raphson* methods.

MAXENT Construction in 2D: Goal: Minimise the convex potential function $F(\lambda_1, \lambda_2) = log(\sum_I e^{-\lambda_1 \tilde{x}_i - \lambda_2 \tilde{y}_i})$. Method: Assume that the solution λ^k at the kth iteration is given, then the minimisation problem can be expanded using a Taylor series

$$F(\lambda^k + \Delta\lambda^k) = \mathcal{R}(\lambda^k) + \nabla\nabla F(x)\Delta\lambda^k = 0$$

where $\mathcal{R}$ is the residual at the kth iteration:

$$\mathcal{R} = \nabla F(\lambda_1^k, \lambda_2^k) > 0$$

The increment $\Delta\lambda^K$ at the kth iteration is given by:

$$\Delta\lambda^k = -\mathcal{H}^{-1}\nabla F$$

Where $\mathcal{H}$ is the Hessian matrix ($\mathcal{H} := \nabla^2 F$). After convergence is reached, subject to a tolerance, the shape functions are given by:

$$\phi_i = \frac{e^{\lambda_1 x_i - \lambda_2 y_i}}{\sum_{j=1}^{n} e^{\lambda_1 x_j - \lambda_2 y_j}} \tag{2.21}$$

Eqs (2-3) describe a Newton-Raphson scheme.

Discrete Equations

To form the discrete equations used in the meshfree simulation we now substitute the shape function approximation of the displacement eq. (5) into the virtual work expression, Box 2. As this is a Galerkin method the same basis functions are used for the trial and test functions, such that:

$$u_i^h = \sum_I \phi_I u_{Ii}, \quad \delta u_i = \sum_I \phi_I \delta u_{Ii} \tag{2.22}$$

Considering the virtual work expression evaluated at a point X_L, Fig 3. The n nodes that influence of the point x is given by the set:

$$\mathcal{I}_L = \{x_I : w(x - X_I) > 0\} \tag{2.23}$$

• set of nodes $\mathcal{I}_L$

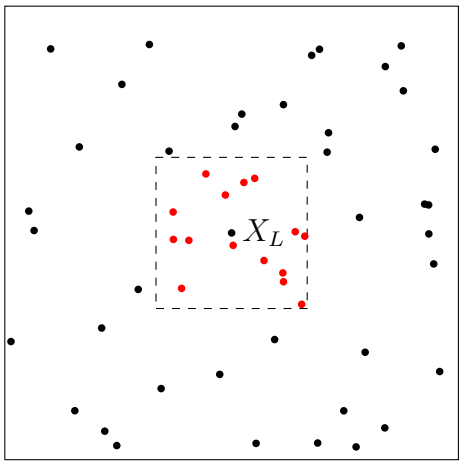

Figure 2.1: Virtual work at a point X_L

And therefore the displacement given by:

$$u_i(x_L) = \sum_I \phi_I(X_L) u_{Ii} \quad \text{for } I \in \mathcal{I}_L \tag{2.24}$$

or given by the k vector equations

$$u_K(X_L) = \mathbf{\Phi}^T \mathbf{u}_k =< \phi_I, ..., \phi_N > \cdot < u_{1k}, ..., u_{Nk} >^T \tag{2.25}$$

where $\text{len}(\mathbf{u}) = card(\mathcal{I}_L)$. The derivatives of u can be developed in a similar manner

$$\frac{\partial u_k(X_L)}{\partial X_J} = \Phi^T_{,J} u_k = \mathbf{g}^T_J \mathbf{u}_k \tag{2.26}$$

The first term of the internal virtual work ($GRAD\ \delta u$) can hence be evaluated as

$$(GRAD\ \delta u)_{kJ} = \frac{\partial \delta u_k}{\partial X_J} = \underbrace{\begin{bmatrix} \mathbf{g}_1^T \delta \mathbf{u}_1 & \mathbf{g}_2^T \delta \mathbf{u}_1 & \mathbf{g}_3^T \delta \mathbf{u}_1 \\ \mathbf{g}_1^T \delta \mathbf{u}_2 & \mathbf{g}_2^T \delta \mathbf{u}_2 & \mathbf{g}_3^T \delta \mathbf{u}_2 \\ \mathbf{g}_1^T \delta \mathbf{u}_3 & \mathbf{g}_2^T \delta \mathbf{u}_3 & \mathbf{g}_3^T \delta \mathbf{u}_3 \end{bmatrix}}_{\mathbf{G}} \tag{2.27}$$

For computational reasons (give some) it is necessary to transform 2.27 into a matrix-vector equations, by means of Voigt notation, which reduces a second

order tensor (such as the deformation gradient) into a vector, and a fourth order tensor (such as the material stiffness tensor) into a matrix.

Voigt Notation:

$$\{F\} = Voigt\left(\begin{bmatrix} F_{11} & F_{12} & F_{13} \\ F_{21} & F_{22} & F_{23} \\ F_{31} & F_{32} & F_{33} \end{bmatrix} \right)$$
$$=< F_{11}, F_{22}, F_{33}, F_{23}, F_{13}, F_{12}, F_{21}, F_{31}, F_{32} >^T$$

or for symmetric tensors

$$\{S\} = Voigt\left(\begin{bmatrix} S_{11} & S_{12} & S_{13} \\ S_{21} & S_{22} & S_{23} \\ S_{31} & S_{32} & S_{33} \end{bmatrix} \right)$$
$$=< S_{11}, S_{22}, S_{33}, S_{23}, S_{13}, S_{12} >^T$$

Using Voigt notation, and after some manipulation with row padding, the following form of the virtual deformation gradient is given by:

$$\{GRAD\ \delta u\}_K = \underbrace{\begin{bmatrix} g_{11} & 0 & 0 & \dots & g_{N1} & 0 & 0 \\ 0 & g_{21} & 0 & \dots & 0 & g_{N2} & 0 \\ 0 & g_{21} & 0 & \dots & 0 & g_{N2} & 0 \\ 0 & g_{21} & 0 & \dots & 0 & g_{N2} & 0 \\ 0 & g_{21} & 0 & \dots & 0 & g_{N2} & 0 \\ 0 & g_{21} & 0 & \dots & 0 & g_{N2} & 0 \\ 0 & g_{21} & 0 & \dots & 0 & g_{N2} & 0 \\ 0 & g_{21} & 0 & \dots & 0 & g_{N2} & 0 \\ 0 & g_{21} & 0 & \dots & 0 & g_{N2} & 0 \end{bmatrix}}_{\mathbf{B_I}(X_L)} \begin{bmatrix} \delta u_{11} \\ \delta u_{12} \\ \delta u_{13} \\ \vdots \\ \vdots \\ \delta u_{N1} \\ \delta u_{N2} \\ \delta u_{N3} \end{bmatrix} \tag{2.28}$$

where $g_{IJ} = \frac{\partial \phi_I}{\partial X_J}$. The matrix B is known as the strain-displacement matrix, and after applying Voigt notation to the 1PKF stress, P yields the following form of the internal force

$$\delta W^{int} = \int (B\delta u)^T P d\Omega \tag{2.29}$$

As $\delta\mathbf{u}$ is not a function of the material coordinates it can be taken out of the

integral,

$$a = \tag{2.30}$$
$$\implies b = 0 \tag{2.31}$$

2.2 Constitutive Modelling

An essential stage in the solution of the virtual work expression, Box 2, is to relate the trial displacement field $u(X,t)$ to the stresses. This is made possible through the use of a constitutive equation. The simplest form of a constitutive law for large strain is the St Venant-Kirchoff law, which is linear in the strain measure (but not the displacements due to the non-linear relationship between the strains and the displacements), represented by the equation:

$$S_{IJ} = \lambda E_{KK}\delta_{KK}\delta_{IJ} + 2\mu E_{IJ} \tag{2.32}$$

Materials such as this one are part of a general class of materials known as hyper-elastic materials, where the constitutive law is given in terms of a strain energy density function Ψ. For example, the strain energy density function for the St. Venant Kirchoff material is given by:

$$\psi(E_{IJ}) = \frac{\lambda}{2}\big[E_{IJ} : \delta_{MN}\big] \tag{2.33}$$

In general the strain energy density function for isotropic hyper-elastic models is defined in terms of the invariants of the right Cauchy-Green tensor C, given by

$$I_1 = tr(C)$$
$$I_2 = (tr(C))^2 - tr(C^2)$$
$$I_3 = det(F)$$

The routine for defining, or using a hyper-elastic is as follows

Hyperelastic Material: Given a functional relationship between the strain energy density function, and the invariants I_1, I_2, I_3 of the right-Cauchy deformation tensor, the Cauchy stress can be obtained from the following:

$$\sigma = \frac{2}{J}\left(\frac{\partial \Psi}{\partial I_1}\right) \tag{2.34}$$

2.3 Integration

In order to solve the virtual work expression integration is required, in order to achieve this in the finite element method Gauss integration is typically used. However, the application of the Gauss integration to meshfree methods leads to poor performance, due the fact that meshfree shape functions are not polynomials, and also have an arbitrary supports, which does not necessarily align with the integration structure. In response to this different integration schemes have been proposed, such as support integration, stress point nodal integration and corrected gauss integration. In this library stabilised nodal integration is used, which ensures that the first order reproducing conditions of the shape functions are met, Box 3. This condition imposes a restriction on the integration scheme, which can be shown by considering a linear patch test, Fig 2, where a linear displacement field is prescribed on the displacement boundary Γ_u, given by:

$$u_{iI} = a_{i0} + a_{ij}x_{jI} \quad \text{for I on } \Gamma_u \tag{2.35}$$

In order to satisfy the patch test, the meshfree solution should be given by a linear polynomial $u_i = a_{i0} + a_{ij}x_j$, and hence the strains by:

$$\begin{aligned} \epsilon_x &= u_{x,x} = a_{x1} \\ \epsilon_y &= u_{x,x} = a_{y2} \\ 2\epsilon_{xy} &= u_{x,y} + u_{y,x} = a_{x2} + a_{y1} \end{aligned}$$

which are constant throughout the domain, hence the stress σ^c is also constant. Considering the small-strain virtual work expression, with no inertial term (equivalent to the except the use of the small strain tensor $\epsilon :=$

- prescribed displacement node
- free node

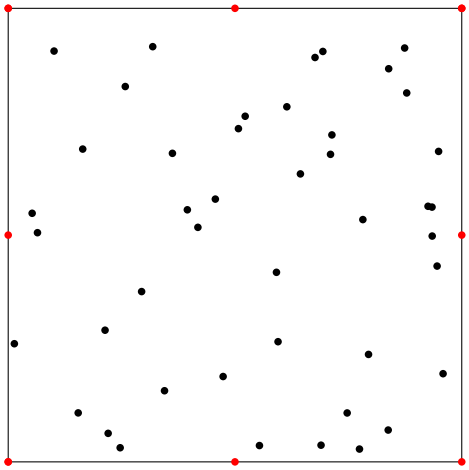

Figure 2.2: Patch Test

$sym(\nabla u)$, and Cauchy stress σ), the virtual work is given by:

$$\int B_I^T \sigma d\Omega = \int \phi_I t_i d\Gamma_t \tag{2.36}$$

As the boundary is unloaded, $\int \phi_I t_i d\Gamma_t = 0$, leading to the condition:

Divergence-free Condition:

$$\int B_I d\Omega = 0 \quad \text{for}$$

If the integration in 2.36 is undertaken using Gauss integration the condition, which leads to poor convergence. In order to create an integration scheme that will satisfy this condition it is necessary redefine the strain displacement matrix B. This is achieved by considering smoothing of the strain over a representative volume, which for the case of the small strain tensor ϵ is given by:

$$\bar{\delta\epsilon} = \frac{1}{A_L} \int sym(\nabla \delta u) d\Omega_L \tag{2.37}$$

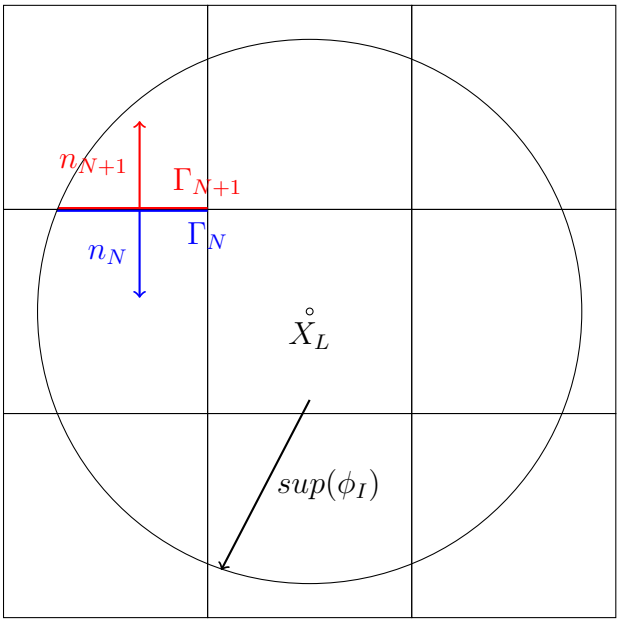

Figure 2.3: A figure

which through divergence theorem, can be related to the flux across the surface

$$\bar{\delta\epsilon} = \frac{1}{A_L} \int \frac{1}{2} (\delta u_i \cdot n_j + \delta u_j \cdot n_i) d\Gamma_L \tag{2.38}$$

where n_j are the components of the surface normals.

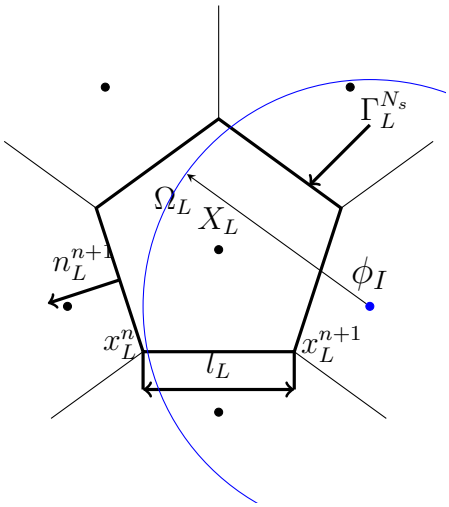

Figure 2.4: Voronoi Diagram

Chapter 3

Design

3.1 Domain

Overview

The domain package is intended to hold together the geometric details of the problem, i.e the nodes, the intergration cells and the degrees of freedom attached to the domain. From this other packages will reference this domain, such as the meshfree packaage, which builds the meshfree domain, and the integration package. The domain is described by the structure

```
type Domain struct {
   Name string // name the domain
   Nodes []node.Node // nodes within the domain
   num_nodes int // number of nodes
   voronoi *voronoi.Voronoi // Voronoi diagram
   dim int // dimension of domain
   boundaryNodes []int // index of nodes on boundary
   global_basis *[]*Dir //basis vectors e.g <1,0> <0,1>
       for 2D
}
```

At present there are two ways to build a domain, manually by adding nodes to a domain object, or more simply by providing a planar-straight line graph (PLSG) to the domain constructor, given by:

```
domain := domain.DomainNew(fileName string, options string,
    dim int, global_coordinate coordinateSystem)
```

Where *fileName* is the name of the PLSG, *options* provides a set of rules the mesh, and voronoi generator, see `https://www.cs.cmu.edu/~quake/triangle.html` for a description. The variable *dim* ensures that correct number of degrees of freedom(DOFs) will be set. The coordinate system can be generated using the following function (for Cartesian coordinates),

```
globalCS := coordinatesystem.CreateCartesian()
```

axisymmetric (cylindrical coordinates) are also supported. This function will construct the nodes, the degrees of freedom (assuming they are all free to start with) and the Voronoi diagram, which is used in the stabalised nodal integration scheme (SCNI).

Public functions

In this section the public functions available to a domain object are described

```
func (domain *Domain) GetDim() int
```

Returns the dimensions of the Domain object *domain*

```
func (domain *Domain) SetCoordinateSystem()
```

Sets the coordinate system of the Domain object *domain*

```
func (domain *Domain) AddNodes(nodes ...*node.Node)
```

Appends the nodes to the domain and increments the number of nodes counter

```
GetNumNodes
```

```
GetNodesIn
```

```
TriGen
```

```
UpdateDomain
```

`CopyDomain`

`CreatDofs`

`GenerateClippedVoronoi`

`GetVoronoi`

`PrintNodesToImg`

Private functions

None so far

3.2 Shape functions

The shape function routines are contained within the *shapefunctions* package. The fundamental data structure in this routine is the meshfree structure, which is a 'layer' of meshfree information implemented on-top of the physical domain

```
type Meshfree struct {
   domain *domain.Domain // reference to the underlying
      physical domain
   nodalSpacing []float64 // distance to the closest node
      for each node
   gamma []float64 // support size multiplier for each node
   isConstantSpacing bool// whether base domain sizes are
      the same
   isVariousPoints bool // whether finding basis functions
      at multiple points
   basisFunctionRadii []float64 // radius(support) of each
      basis function
   dim int // does not need to be here
   tol float64// tolerance for maxent convergence
}
```

To construct a meshfree structure the following constructing function can be used:

```
func NewMeshfree(domain *domain.Domain, isConstantSpacing bool,
    isVariousPoints bool, dim int, gamma []float64, tol float64)
```

if the support size parameter ($gamma[]float64$) is left empty it can be set using the following functions:

```
func (meshfree *Meshfree) SetConstantGamma(gamma float64)
```

if a constant gamma (same support size parameter for each node) or

```
func (meshfree * Meshfree) setGamma(gamma []float64)
```

Note: the length of gamma should be equal to the number of nodes in the domain.

private functions:

3.3 Geometry

Chapter 4

Furture Plans

Chapter 5

Area of Improvements

As the matrix library (gonum) is based off the blas libaries the performance for smaller domains is expected to be poor due to the overhead of calling the blas routines. Therefore it may be best to rediesng the shape function library.

Chapter 6

Acknowledgements

The following packages have been used in this project