

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

Stephen Smith  
Queen's University Belfast

2018  
October

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	MFree . . . . .	1
1.2	Overview . . . . .	1
<b>2</b>	<b>Theory</b>	<b>2</b>
2.1	Meshfree methods . . . . .	2
2.2	Constitutive Modelling . . . . .	7
<b>3</b>	<b>Design</b>	<b>9</b>
3.1	Domain . . . . .	9
3.2	Shape functions . . . . .	11
3.3	Geometry . . . . .	11
<b>4</b>	<b>Furture Plans</b>	<b>12</b>
<b>5</b>	<b>Area of Improvements</b>	<b>13</b>
<b>6</b>	<b>Acknowledgements</b>	<b>14</b>

# Chapter 1

## Introduction

### 1.1 MFree

The Mfree library 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 languages 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 language has been chosen as it provides inbuilt concurrency along with a readability, and usability often not found in other high performance languages (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 continuous 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 differential 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.

$$\text{GRAD } P_{i,J} = \rho_0 \ddot{u}_i \quad (2.1)$$

where  $P$  is the first Piola-Kirchoff stress,  $\rho_0$  the material density in the reference frame (typically  $t = 0$ ) and  $\ddot{u}$  the acceleration. To complete the balance of linear momentum boundary conditions must be specified. A boundary is called a displacement boundary, denoted  $\Gamma_u$ , if a displacement  $u$  is prescribed on that boundary, likewise a similar definition is present for the traction

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

$$u_i = \bar{u}_i \text{ on } \Gamma_u \quad (2.2)$$

$$P_{ij}N_j = \bar{T}_i \text{ on } \Gamma_t \quad (2.3)$$

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

**Strong form of the momentum balance**

GRAD $P_{iJ} = \rho_0 \ddot{u}_i$	Momentum Balance
$u_i = \bar{u}_i \text{ on } \Gamma_u$	Displacement condition
$P_{ij}N_j = \bar{T}_i \text{ on } \Gamma_t$	Traction condition

### Strong form to weak form

To construct the weak form a test function  $\delta u$  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 displacement field  $u(X, t)$  such that the following equation is satisfied*

$$\int \delta u_i \cdot [\text{GRAD } P_{iJ} - \rho_0 \ddot{u}_i] d\Omega = 0 \quad (2.4)$$

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

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

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

$$\delta W = \delta W^{int} - \delta W^{ext} + \delta W^{kin} \quad (2.6)$$

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  $\delta u$  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 displacement 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  $\delta u$  the virtual work  $\delta W$  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

$$\implies \sum_I \phi_I = 1$$

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

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

This leads to the following form

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

and the derivative by:

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

The construction of the shape functions  $\phi_I$  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

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

which clearly satisfies the first condition (constant reproduction) as

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

However, in order to satisfy the linear reproducing conditions an additional enrichment term is required. Consider the general form of the meshfree approximation given by

$$u^h(x) = \sum_i^n C_i(x) \Gamma_i(x) \quad (2.13)$$

.....

### 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 probabilistic approach to mesh-free shape functions was developed by Ortiz [], known as the MAXENT scheme. The principle behind the MAXENT scheme is that if the informational entropy, denoted  $H$ , is maximising probability distribution will be the least-biased one. In this approach the problem is: *Given a set of mutually independent discrete events  $e_1, e_2, \dots, e_n$ , that occur with probabilities  $p_1, p_2, \dots, p_n$*



**MAXENT Construction:** 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  $\lambda^k$  at the  $k$ th iteration is given, then the minimisation problem can be expanded using a Taylor series

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

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

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

The increment  $\Delta\lambda^K$  at the  $k$ th 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:

$$\phi_i = \frac{e^{\lambda_1 x_i - \lambda_2 y_i}}{\sum_{j=1}^n e^{\lambda_1 x_j - \lambda_2 y_j}} \quad (2.14)$$

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

## 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:

$$S_{IJ} = \lambda E_{KK} \delta_{KK} \delta_{IJ} + 2\mu E_{IJ} \quad (2.15)$$

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  $\Psi$ . For example, the strain energy density

function for the St. Venant Kirchhoff material is given by:

$$\psi(E_{IJ}) = \frac{\lambda}{2} [E_{IJ} : \delta_{MN}] \quad (2.16)$$

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

$$I_1 =$$

$$I_2 =$$

$$I_3 =$$

The routine for defining, or using a hyperelastic 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:

$$\sigma = \frac{2}{J} \left( \frac{\partial \Psi}{\partial I_1} \right) \quad (2.17)$$

# 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 stabilised 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**

## **3.3 Geometry**

## Chapter 4

### Furture Plans

# Chapter 5

## Area of Improvements

As the matrix library (gonum) is based off the blas libraries 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 redesign the shape function library.

## Chapter 6

# Acknowledgements

The following packages have been used in this project