FINITE ELEMENT ANALYSIS OF DISCONTINUOUS CONTACT PROBLEMS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

MEHMET ATA BODUR

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
ENGINEERING SCIENCES

JANUARY 2006

Approval of the Graduate School of Natural and Applied Sciences

_____
Prof.Dr. Canan ÖZGEN
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

_____
Prof. Dr. M. Ruşen GEÇİT
Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Sciences.

_____
Prof. Dr. M. Ruşen GEÇİT
Supervisor

Examining Committee Members

Prof. Dr. Turgut TOKDEMİR	(METU, ES)	_____
Prof. Dr. M. Ruşen GEÇİT	(METU, ES)	_____
Prof. Dr. Yusuf ORÇAN	(METU, ES)	_____
Assoc.Prof. Dr. Ahmet N. ERASLAN	(METU, ES)	_____
Assoc.Prof. Dr. Sibel TARI	(METU, CENG)	_____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last name : Mehmet Ata BODUR
Signature :

ABSTRACT

FINITE ELEMENT ANALYSIS OF DISCONTINUOUS CONTACT PROBLEM

BODUR, Mehmet Ata

M. Sc., Department of Engineering Sciences

Supervisor: Prof. Dr. M. Ruşen Geçit

January 2006, 122 pages

Contact is a phenomenon faced in every day life, which is actually a complex problem to tackle for engineers. Most of the times, may be impossible to get analytic or exact results for the interaction of bodies in contact.

In this thesis work, solution of the frictionless contact of an elastic body, touching to a rigid planar surface for two-dimensional elasticity; namely plane stress, plane strain and axi-symmetric formulations is aimed. The problem is solved numerically, with *Finite Element Method*, and an *Object Oriented* computer program in C++ for this purpose is written, and the results are verified with some basic analytic solutions and ABAQUS package program.

It is not aimed in this thesis work to give a new solution in the area of solution of contact problems, but instead, it is aimed to form a strong basis, and computational library, which is extendible for further development of the subject to include friction, plasticity, and different material modeling in this advanced field of mechanics.

Keywords: Finite Element, Contact, OOP, C++

ÖZ

SÜREKSİZ TEMAS PROBLEMLERİNİN SONLU ELEMANLAR YÖNTEMİ İLE ÇÖZÜMÜ

Bodur, Mehmet Ata

Yüksek Lisans, Mühendislik Bilimleri Bölümü

Tez Yöneticisi: Prof.Dr. M. Ruşen Geçit

Ocak 2006, 122 sayfa

Temas, günlük hayatta karşılaşılan bir olgu, fakat aslında mühendisler için uğraşması zor bir problemdir. Temasla etkileşen cisimler için çoğu zaman analitik ya da kesin çözüm elde etmek mümkün değildir.

Bu tez çalışmasında, bir elastik cisim ile bir katı düz yüzey arasındaki sürtünmesiz temasın, iki boyutlu elastik; düzlem gerilme, düzlem şekil değiştirme ya da eksenel simetrik olarak adlandırılan modellemeyle çözülmesi amaçlanmaktadır. Problem, sayısal olarak, *Sonlu Elemanlar Yöntemi* ile, ve bu amaçla C++ programlama dilinde *Nesne Tabanlı* bilgisayar programı yazılarak çözülmektedir, ve sonuçlar bazı bilinen analitik çözümlerle ve ABAQUS Paket Programı ile karşılaştırılmaktadır.

Bu tez çalışmasında, temas problemleri çözümü alanında çok yeni ve özel birşeyler eklemek yerine, hesaba dair konunun daha öteye, sürtünme, plastikleşme ve farklı malzeme modellemesine yönelik olarak genişletilebilmesine imkan tanıyan sağlam bir temel oluşturulması amaçlanmaktadır.

Anahtar Kelimeler: Sonlu Elemanlar, Temas, Nesne Tabanlı Programlama, C++.

To My Parents

For their endless trust and help

Prof. Dr. M. Ruşen Geçit

For his unceasing trust, belief and effort to me in completing this dissertation

Prof. Dr. Turgut Tokdemir

For his invaluble discussions and comments

ACKNOWLEDGMENTS

The author wishes to give his special thanks to Prof Dr. M. Ruşen Geçit for his unceasing help and support in completing this dissertation. The author is grateful for his proofreading, and correcting many things, and especially his belief in completion of this work, and giving inspiration; and Prof. Dr. Turgut Tokdemir for his comments and support for the program developed, and for teaching Finite Element, and other numerical methods.

Thanks also go to Prof. Dr. Yalçın Mengi for his sharing invaluable knowledge on Continuum Mechanics with the courses he teaches in the Engineering Sciences Department of METU, and Prof. Dr. Gerhard Wilhelm Weber in the Industrial Applied Mathematics department of METU for his optimization course and for his reading and checking the text, and being very supportive.

The author also wishes to mention his colleague PhD. candidate Celal Soyarslan, for his friendship, discussions, and sharing his experience and knowledge on Finite Element Method.

Finally, but most special thanks go to his parents Sabriye and İbrahim Bodur for their endless, unceasing support and love.

TABLE OF CONTENTS

LIST OF TABLES

Number ***Page***

LIST OF FIGURES

Number ***Page***

LIST OF ALGORITHMS

Number ***Page***

LIST OF SYMBOLS

b : Eulerian deformation tensor.
$\boldsymbol{B_0}$: Derivative matrix with respect to reference coordinates.
$\boldsymbol{B}$: Derivative matrix with respect to current coordinates.
C : Green Lagrange Deformation Tensor.
$\hat{\mathbf{C}}$: Constitutive tensor in reference coordinate system.
$\hat{c}$: Constitutive tensor in current coordinate system.
ds : Length of small vector in current coordinates.
dS : Length of small vector in reference coordinates.
$\mathrm{d}\mathbf{X}$: Infinitesimall vector in reference coordinates.
$\mathrm{d}\mathbf{x}$: Infinitesimall length in current coordinates.
$detF$: Determinant of deformation gradient.
E : Green's Strain Tensor.
e : Almansi's Strain Tensor.
$\boldsymbol{F}$: Deformation gradient.
F : Contact force for cross constraints method.
$\overline{\mathbf{F}}$: Modified force matrix for contact.
$\hat{F}$: Assumed contact force for cross constraints method.
J : Jacobian from master element to element in curent state.
J_0 : Jacobian from master element to element in reference state.
K : Contact stiffness for cross constraints method.
$\boldsymbol{K_L}$: Linear stiffness matrix.
$\boldsymbol{K_{NL}}$: Non-linear stiffness matrix.
$\boldsymbol{K_T}$: Tangential stiffness matrix for Newton solution.
$\overline{\mathbf{K}}$: Modified stiffness for contact.
$\hat{K}$: Assumed initial contact stiffness for cross constraints method.
$\hat{\mathbf{N}}$: Unit normal vector in reference coordinates
$\hat{\mathbf{n}}$: Unit normal vector in current coordinates.
$\mathbf{P}$: First Piola Kirchoff stress tensor.
$\mathbf{R}$: Rotation part of deformation gradient.
$\mathbf{R}$: Residual vector for Newton iteration.
$\mathbf{S}$: Second Piola Kirchoff stress tensor.
$\hat{\mathbf{S}}$: Second Piola Kirchoff stress tensor in matrix form for Newton iter.
$\mathbf{t}$: Traction vector in current coordinates.
$\mathbf{T}$: Traction vector in reference coordinates.
$\mathbf{U}$: Lagrangian stretch tensor.
$\mathbf{u}$: Displacement vector.
$\mathbf{V}$: Eulerian stretch tensor.
$\mathbf{X}$: Position vector in reference coordinates.
$\mathbf{x}$: Position vecctor in current coordinates.
ε : Engineering strain.
κ : Penalty stiffness.
λ : Lame's modulus.

ρ : Mass density in current coordinates.
ρ_0 : Mass density in reference coordinates.
μ : Shear modulus (p.21).
μ : Barrier parameter for barrier method (p.69)
ν : Poisson's ratio.
$\boldsymbol{\sigma}$: Cauchy stress tensor.
$\hat{\boldsymbol{\sigma}}$: Cauchy stress tensor in matrix form for Newton iteration.

CHAPTER 1

INTRODUCTION

Contact is a phenomenon faced in everyday life, but a problem hard to tackle for engineers. It is well known that it is a phenomenon dealt from the Egyptians time at least. In [1] some historical remarks have been given about history of approaches to the event. First modern approaches attributed to Da Vinci (15th century), Coulomb (1785), Euler (1748) etc.[1]. Though it is not a new event faced in human life, till the last few decades it was only possible to analyse some special types of contact problems analytically, with some crude assumptions. As the industry evolved, more and more elaborate techniques needed to deal with contact. In automobile industry, design of wheels interacting with road, design of clutches, brakes, gears, etc. needs elaborate techniques to analyze this natural event. In civil engineering applications, interaction of girder beams with supports, interaction of foundation with ground etc. are events simply coming into mind about contact. Also in the recent years, one can see some successful designs of plugs of the mobile phone chargers, and some other interesting industrial applications.

By the advent of the computers, new numerical solution techniques have been developed in the last few decades, one of which *Finite Element Method* (FEM) have found enormously wide applications in engineering. FEM is a numerical technique to solve mechanical problems in engineering with the aid of computers, where it is hard or impossible to get an analytic solution. For having a solution with FEM, one has to have a well posed mathematical model, which, representing the physical phenomena in a good way in the domain of the problem. The mathematical model, in general to be defined by differential equations, to be solved numerically by a set of governing algebraic equations [2]. [3] gives a summary about history of FEM. For an understanding of the subject, the reader should consult to references [2]-[5] or other uncountably many references in literature.

While posing the problem initially, one can assume a continuous body supported from some part of the boundary, with some boundary and internal forces defined, and another object to interact with is awaiting or moving in the process of deformation. As the deformation progresses, one expects to see an interaction of the bodies on their boundaries, which is unpredictable at the beginning, and makes the problem highly non-linear. Due to this nature of unpredictability, in the past, the contact interactions were approximated with special crude assumptions.

While attacking to the problem, it becomes very difficult to include all aspects of the problem at once. For that reason, in this study, the topic is bounded by obtaining a FEM implementation of discontinious, frictionless linearly elastic 2D contact, namely the *plane stress*, *plane strain* and *axi-symmetric* problems, which is to be robust, dependable and extendible for further abilities. For the robustness and extendibility issues, C++ programming language, which has been very popular in the last decade is selected for it provides robust and extendable object oriented environment. It also has the support of defining types different than the standard data types like integers, real numbers, arrays etc.. In the object oriented environment, data is organized and distributed in the classes, which provides seperate compilation, neater and cleaner programming environment. Also the data hiding and extraction mechanisms of C++ prevent many errors while programming. Even though the code is implemented for 2D case, most of the mathematical idea are valid and extendible to 3D cases.

The organization of the material is in the following order: In Chapter 2, general continuum equations used in the program developed are presented briefly. In Chapter 3, FEM formulation is introduced. For the completeness, triangular and rectangular elements are defined. In Chapter 4, constraint formulation techniques is dealt mathematically and application to contact formulation is discussed, application of these techniques to general FEM equations are explained briefly. In Chapter 5, OOP approach to FEM is discussed and some implementation details regarding this issue are introduced. In Chapter 6, some benchmark tests are considered and comparison to another FEM program ABAQUS and some exact analytical solutions are done. Finally, in Chapter 7, concluding remarks and further development issues are presented.

CHAPTER 2

CONTINUUM MECHANICS PRELIMINARIES

2.1 INTRODUCTION

In this section, necessary continuum mechanics equations are presented. Since there are too many items, which all cannot be mentioned here, the context in this chapter is restricted to the applied formulations to justify the applied ones. Most of the details are left to the reader with giving references [2] and [5] and the references therein.

2.2 STRAIN DISPLACEMENT RELATIONS

In the context of continuum mechanics, stresses are defined as the function of strains. Strain is in general a second order tensor representing the deformation state of the object at a point in the domain, which is a function of displacements. One can initially define an object moving and deforming in space and time (Figure 2.1). Initial configuration is defined as **X** and current configuration as **x**. It should be declared here that variables written in bold are representing vector values, where in 2D having two components, and in 3D having three components and parameters referring to initial state are defined in capital letters, whereby parameters referring to current state are represented by minuscule.

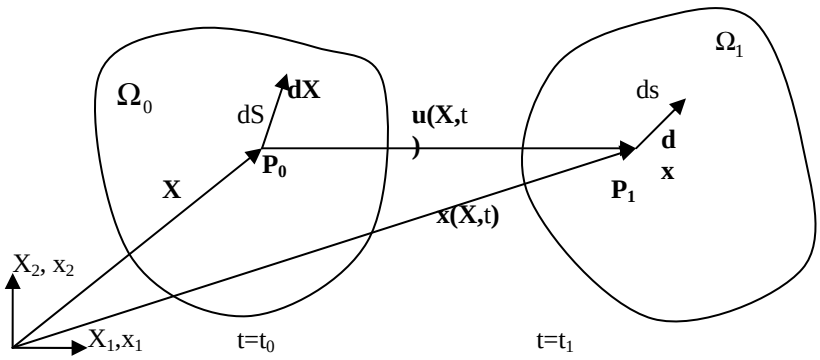


Figure 2.1: Initial state of object at time t_0 and deformed state of object at t_1. $\mathbf{P_0}$ at the initial state defined with the position vector **X**, deforming and moving to $\mathbf{P_1}$ defined with the position vector **x**. **x** is a function of **X** and t.

In the above, all the coordinates defined with respect to a reference frame, which, fixed in space and time, is the *Lagrangian Description*, where in the opposite case, in which the reference frame is moving in space and time, is called the *Eulerian Description*. In the Lagrangian analysis, the particles are followed individually, whereby in the Eulerian approach, particles passing through a fixed point are watched. In structural analysis, in general, the *Lagrangian Description*, whereby in fluid dynamics *Eulerian Description* is preferred.

The motion of the body can be defined as:

$$\mathbf{x} = \boldsymbol{\varphi}(\mathbf{X}, t)$$

Or in indicial notation

$$x_i = \varphi_i(X_j, t) \tag{2.1}$$

The function $\boldsymbol{\varphi}(\mathbf{X}, t)$ maps the reference configuration at time *t=0* into current configuration at time *t=t* and, it is called the mapping from the initial to current configuration. At time *t=0* , **x** is coincident to **X** .

From the above figure, it can easily be seen that:

$$\mathbf{u} = \mathbf{x} - \mathbf{X}\text{, or in indicial notation } u_i = x_i - X_I \cdot \delta_{Ii} \tag{2.2}$$

The above can be also written as:

$$u_i = \varphi_i(X_J, t) - X_I \cdot \delta_{Ii} \tag{2.3}$$

At this point, one may like to define the *deformation gradient* **F**, which transforms the infinitely small vector in the reference configuration to the current configuration.

In tensorial notation:

$$\mathbf{F} = \frac{\partial \varphi}{\partial \mathbf{X}} \equiv \frac{\partial \mathbf{x}}{\partial \mathbf{X}} \equiv (\nabla_0 \varphi)^T \text{ , or in indicial notation:}$$

$$F_{iJ} = \frac{\partial \varphi_i}{\partial X_J} = \frac{\partial x_i}{\partial X_J} \tag{2.4}$$

From the above formula, it is obvious that:

$$\mathbf{dx} = \mathbf{F} \cdot \mathbf{dX} \text{ , or in indicial notation: } dx_i = F_{iJ} \cdot dX_J \tag{2.5}$$

The formula in Eq. (2.2) can be applied to the previous Eq. (2.4) with an arrangement and the equation below is obtained.

$$F_{iJ} = \delta_{iJ} + \frac{\partial u_i}{\partial X_J} \tag{2.6}$$

The deformation gradient can be written as a matrix expression in 3D as follows:

$$\mathbf{F} = \begin{bmatrix} \frac{\partial x_1}{\partial X_1} & \frac{\partial x_1}{\partial X_2} & \frac{\partial x_1}{\partial X_3} \\ \frac{\partial x_2}{\partial X_1} & \frac{\partial x_2}{\partial X_2} & \frac{\partial x_2}{\partial X_3} \\ \frac{\partial x_3}{\partial X_1} & \frac{\partial x_3}{\partial X_2} & \frac{\partial x_3}{\partial X_3} \end{bmatrix} \tag{2.7}$$

The determinant of **F** is important in the formulation of general equations of continuum mechanics in transforming integrations from current to initial state forms. In literature it is defined as *J*, but since in finite element context *J* refers to mapping from master element to actual element, which will be clear in the foregoing chapters, the determinant of **F** is denoted by *detF* instead of *J*.

For mapping from the reference to current configuration to be possible, φ should be one to one, continuously differentiable, and *detF>0*. The condition one to one means, there exists only one point in the current configuration for a point in reference configuration or vice versa. For the backward compatibility, F should be invertible, which requires that $\det F \neq 0$. In the above, the more strict condition requiring that $\det F > 0$ is written, which comes from mass conservation, and is dealt in conservation equations. Continuous differentiability is obviously necessary for calculation of **F**.

The above conditions can be violated in special situations, such as crack propagation, but in the context of this dissertation, formulation is based on the above assumptions (Belytschko et al. [5]).

Here it should also be stated that *detF* relates the volume in reference configuration to present configuration as:

$$\det F \cdot dV_0 = dV \tag{2.8}$$

In the above equation, dV_0 is the volume in the reference configuration, and dV is the volume in the current configuration.

The deformation gradient can be decomposed into rotation and stretch parts as:

$$\mathbf{F} = \mathbf{R} \cdot \mathbf{U} \tag{2.9}$$

and,

$$\mathbf{F} = \mathbf{V} \cdot \mathbf{R} \tag{2.10}$$

In both of the decompositions, **R** is the rotation part; **U** is the stretch with respect to the initial state and is called the *Lagrangian stretch tensor*. Conversely, **V** is the stretch with respect to current state and is the *Eulerian stretch tensor*.

Now that deformation gradient has been defined, it is expected to have a relation for strain. The general requirements for strain can be stated as:

It must vanish for any rigid body motion, in particular for rigid body rotation; should increase as the deformation increases (Belytschko et al. [5]). Those requirements are crucial, especially, in the non-linear theory.

A small length in the current state ds, can be related to the initial state dS, by use of Eq. (2.5) and the formulations below:

$ds^2 = (dx_i \cdot dx_i)$, or in matrix form:

$$ds^2 = \mathbf{dx}^T \cdot \mathbf{dx} \tag{2.11}$$

$dS^2 = (dX_I \cdot dX_I)$, or in matrix form:

$$dS^2 = \mathbf{dX}^T \cdot \mathbf{dX} \tag{2.12}$$

By use of Eq. (2.5):

$ds^2 = (F_{iJ} \cdot dX_J) \cdot (F_{iM} \cdot dX_M)$ or in matrix form:

$$ds^2 = \mathbf{dX}^T \cdot \mathbf{F}^\mathbf{T} \cdot \mathbf{F} \cdot \mathbf{dX} \tag{2.13}$$

In the above formulation, $\mathbf{C} = \mathbf{F}^\mathbf{T} \cdot \mathbf{F}$ is called the *right Cauchy-Green deformation tensor*.

Then,

$$ds^2 = \mathbf{dX}^\mathbf{T} \cdot \mathbf{C} \cdot \mathbf{dX} \tag{2.14}$$

From the above formula, stretch of the vector can be defined and given in indicial form as below:

$$\frac{ds}{dS} = \left(F_{iJ} \cdot \frac{dX_J}{dS} \cdot F_{iM} \cdot \frac{dX_M}{dS} \right)^{1/2} \tag{2.15}$$

which is equivalent to:

$$\frac{ds}{dS} = (\hat{\mathbf{N}}_J \cdot \mathbf{C}_{JM} \cdot \hat{\mathbf{N}}_M)^{1/2} \tag{2.16}$$

where, $\hat{\mathbf{N}}$ is the unit vector defined in reference configuration in the direction of $\mathbf{dX}$.

Now the change in square length can be defined as:

$$ds^2 - dS^2 = \mathbf{dX}^T \cdot \mathbf{C} \cdot \mathbf{dX} - \mathbf{dX}^T \cdot \mathbf{dX} \tag{2.17}$$

Dividing both sides of Eq. (2.17) by $dS^2 = \mathbf{dX}^T \cdot \mathbf{dX}$, the equation following can be obtained:

$$\frac{ds^2 - dS^2}{dS^2} = \hat{\mathbf{N}}^T \cdot (\mathbf{C} - \mathbf{I}) \cdot \hat{\mathbf{N}} \tag{2.18}$$

In literature, the above $(\mathbf{C} - \mathbf{I})$ is defined as:

$$2\mathbf{E} = (\mathbf{C} - \mathbf{I}) \tag{2.19}$$

in which $\mathbf{E}$ is called the *Green's strain tensor*.

From the formula in Eq. ((2.18) one can also write the stretch defined as follows:

$$\lambda = \frac{ds}{dS} = \sqrt{\hat{\mathbf{N}}^T \cdot 2\mathbf{E} \cdot \hat{\mathbf{N}} + 1} \tag{2.20}$$

Here one can define the well-known *engineering strain* in one dimension:

$$\boldsymbol{\varepsilon} = \frac{ds - dS}{dS} = \sqrt{\hat{\mathbf{N}}^{T} \cdot 2\mathbf{E} \cdot \hat{\mathbf{N}} + 1} - 1$$

(2.21)

When $\hat{\mathbf{N}}^{T} \cdot 2\mathbf{E} \cdot \hat{\mathbf{N}}$ is small, by Taylor's expansion of the square root, ignoring higher order terms, one can get:

$$\boldsymbol{\varepsilon} = \frac{ds - dS}{dS} = \hat{\mathbf{N}}^{T} \cdot \mathbf{E} \cdot \hat{\mathbf{N}}$$

(2.22)

This is the engineering definition of strain for one-dimensional state. For strain being small, the difference between *Green's strain* and the *engineering strain* becomes ignorable. Nevertheless, for large deformation case one obviously needs to employ a strain definition different from the engineering strain i.e., the *Green's strain*. In the sequel, this distinction will be further well understood while deriving Non-Linear FEM equations.

Up to this point, a measure for deformation of a vector has been obtained. Now, a relation is needed for the deformation of the edges of a rectangular infinitesimally small element in the reference configuration.

Now that infinitesimal length in current state has been related to infinitesimal length in initial state, one can derive relations for deformation of the edges of a rectangle, which is the angle change, in reference configuration to current state (Figure 2.2).

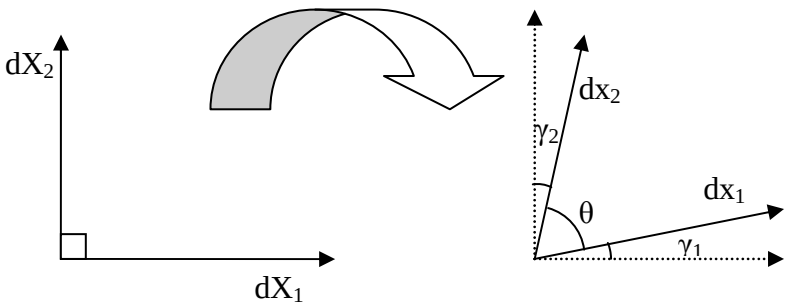


Figure 2.2: Deformation of the edges of a rectangle in initial state to current state.

The dot product of the edges of the parallelogram in the current state referring to the rectangle edges in the initial state can be written as:

$$\left|\mathbf{dx}_1\right| \cdot \left|\mathbf{dx}_2\right| \cdot \cos(\theta) = \mathbf{dX}_1^T \cdot \mathbf{F}^{\mathbf{T}} \cdot \mathbf{F} \cdot \mathbf{dX}_2 = \mathbf{dX}_1^T \cdot \mathbf{C} \cdot \mathbf{dX}_2 \tag{2.23}$$

or rearranging terms:

$$\cos(\theta) = \frac{\mathbf{dX}_1}{\left|\mathbf{dx}_1\right|} \cdot \mathbf{C} \cdot \frac{\mathbf{dX}_2}{\left|\mathbf{dx}_2\right|} \tag{2.24}$$

By utilization of Eq. (2.20), the above is equivalent to:

$$\cos(\theta) \equiv \sin(\gamma 1 + \gamma 2) = \frac{\hat{\mathbf{N}}_1}{\sqrt{\hat{\mathbf{N}}_1^T \cdot \mathbf{2E} \cdot \hat{\mathbf{N}}_1 + 1}} \cdot \mathbf{C} \cdot \frac{\hat{\mathbf{N}}_2}{\sqrt{\hat{\mathbf{N}}_2^T \cdot \mathbf{2E} \cdot \hat{\mathbf{N}}_2 + 1}} \tag{2.25}$$

Considering the small strain situation in the above equations, and observing that N1 and N2 are parallel to axes, i.e. parallel to i and j axes:

$$\sqrt{\hat{\mathbf{N}}_1^T \cdot \mathbf{2E} \cdot \hat{\mathbf{N}}_1 + 1} \approx 1, \text{ and}$$

$$\sqrt{\hat{\mathbf{N}}_2^T \cdot \mathbf{2E} \cdot \hat{\mathbf{N}}_2 + 1} \approx 1 \tag{2.26}$$

$$C_{ij} = 2E_{ij}, \qquad \text{when } i \neq j,$$

$$\sin(\gamma 1 + \gamma 2) \approx (\gamma 1 + \gamma 2) = \gamma, \text{ then:}$$

$$\gamma \approx 2E_{ij} \tag{2.27}$$

In the above, it is not aimed to create confusion to the reader, but it is aimed to present the distinction and understanding of strain in large deformation state, and recovery to general engineering definitions of strain in small deformation case.

In the sequel of this section, it is aimed to relate current state to the initial state. As previously stated, the deformation gradient **F** is one to one, and there exists an inverse relation from current state to initial state. Hence, one can define:

$$\mathbf{X} = \varphi^{-1}(\mathbf{x}, t) \equiv \mathbf{X}(\mathbf{x}, t) \tag{2.28}$$

An infinitesimally small length in the current state can be transformed back to initial state by the next equation:

$$\frac{\partial \mathbf{X}}{\partial \mathbf{x}} = \frac{\partial \varphi^{-1}}{\partial \mathbf{x}} \equiv \mathbf{F}^{-1} \text{ or } d\mathbf{X} = \mathbf{F}^{-1} \cdot d\mathbf{x} \tag{2.29}$$

A small length in initial state *dS*, can be related to current state *ds*, by use of Eq. (2.29) as follows:

$$dS^2 = (F^{-1}{}_{iM} \cdot dx_i) \cdot (F^{-1}{}_{jM} \cdot dx_j), \qquad \text{or in matrix form:}$$

$$dS^2 = \mathbf{dx}^T \cdot \mathbf{F}^{-\mathbf{T}} \cdot \mathbf{F}^{-1} \cdot \mathbf{dx} \tag{2.30}$$

This is equivalent to:

$$dS^2 = \mathbf{dx}^T \cdot \mathbf{b}^{-1} \cdot \mathbf{dx} \tag{2.31}$$

In the above, **b** is called the *left Cauchy-Green deformation tensor*.

One can also express the length change with respect to current state:

$$\frac{ds^2 - dS^2}{ds^2} = \hat{\mathbf{n}}^T \cdot (\mathbf{I} - \mathbf{b}^{-1}) \cdot \hat{\mathbf{n}} \tag{2.32}$$

where $\hat{\mathbf{n}}$ is the unit vector in the direction of **dx** in the current state. From the above formulation, *Almansi's strain tensor* is defined as:

$$2\mathbf{e} = (\mathbf{I} - \mathbf{b}^{-1}) \tag{2.33}$$

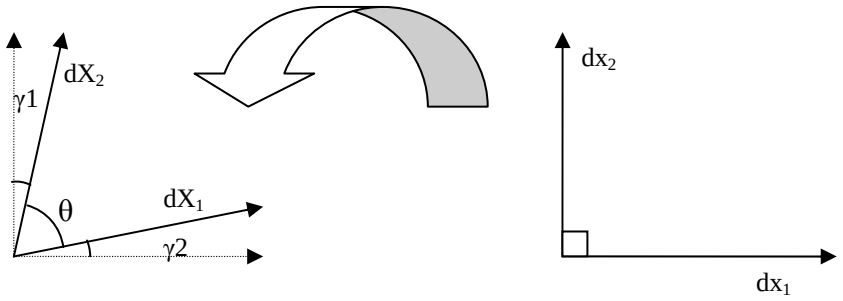


Figure 2.3: Angle change of the rectangle in current state from the initial un-deformed parallel piped.

Angle change for the edges of a rectangle in the current state can also be defined as in the *Lagrange strain tensor* case by the following:

$$\left|\mathbf{dX_1}\right| \cdot \left|\mathbf{dX_2}\right| \cdot \cos(\theta) = \mathbf{dx_1^T} \cdot \mathbf{F^{-T}} \cdot \mathbf{F^{-1}} \cdot \mathbf{dx_2} = \mathbf{dx_1^T} \cdot \mathbf{b^{-1}} \cdot \mathbf{dx_2} \tag{2.34}$$

or rearranging terms:

$$\cos(\theta) = \frac{\mathbf{dx_1}}{\left|\mathbf{dX_1}\right|} \cdot \mathbf{b^{-1}} \cdot \frac{\mathbf{dx_2}}{\left|\mathbf{dX_2}\right|} \tag{2.35}$$

which is equivalent to:

$$\cos(\theta) \equiv \sin(\gamma 1 + \gamma 2) = \frac{\hat{\mathbf{n}}_1^{\mathbf{T}}}{\sqrt{\mathbf{1} - \hat{\mathbf{n}}_1^{\mathbf{T}} \cdot \mathbf{2e} \cdot \hat{\mathbf{n}}_1}} \cdot \mathbf{b^{-1}} \cdot \frac{\hat{\mathbf{n}}_2}{\sqrt{\mathbf{1} - \hat{\mathbf{n}}_2^{\mathbf{T}} \cdot \mathbf{2e} \cdot \hat{\mathbf{n}}_2}} \tag{2.36}$$

Considering the small strain case from the above equations and observing that $\hat{\mathbf{n}}_1$ and $\hat{\mathbf{n}}_2$ are parallel to axes in current configuration, the angle change from initial state to current state can be defined as γ, which can be obtained as follows:

$\sqrt{1-\hat{\mathbf{n}}_{\mathbf{1}}^{\mathbf{T}}\cdot\mathbf{2e}\cdot\hat{\mathbf{n}}_{\mathbf{1}}}\approx 1$, and

$\sqrt{1-\hat{\mathbf{n}}_{2}^{T}\cdot\mathbf{2e}\cdot\hat{\mathbf{n}}_{2}}\approx 1$

$-b^{-1}{}_{ij}=2e_{ij}$, when $i\neq j$,

$\sin(\gamma 1+\gamma 2)\approx(\gamma 1+\gamma 2)=\gamma$,

$\gamma=2e_{ij}$

(2.37)

At this point, one should look at what is obtained when *Almansi's Strain Tensor* is multiplied with the deformation gradient on both sides:

$$\mathbf{F^T\cdot 2e\cdot F=F^T(I-(F\cdot F^T)^{-1})\cdot F=(F^TF-I)=2E}$$

(2.38)

The above is defined as the pull back operation of *Almansi's Strain Tensor*.

One more thing to be considered left is writing the strain tensors in displacement form, which constitutes the main framework for working in FEM displacement formulation. Using Eq. (2.6), *Green's strain tensor* can be written as:

$$E_{IJ}=\frac{1}{2}\left(\frac{\partial u_I}{\partial X_J}+\frac{\partial u_J}{\partial X_I}+\frac{\partial u_M}{\partial X_I}\cdot\frac{\partial u_M}{\partial X_J}\right)$$

(2.39)

In the same way, the *Almansi's strain tensor* can also be represented by:

$$e_{ij}=\frac{1}{2}\left(\frac{\partial u_i}{\partial x_j}+\frac{\partial u_j}{\partial x_i}-\frac{\partial u_m}{\partial x_i}\cdot\frac{\partial u_m}{\partial x_j}\right)$$

(2.40)

For small strain conditions, in both of these equations; the multiplication terms become small and the difference between the current and the initial states becomes negligible, then the equations reduce to:

$$\varepsilon_{ij}=\frac{1}{2}\left(\frac{\partial u_i}{\partial x_j}+\frac{\partial u_j}{\partial x_i}\right)$$

(2.41)

It is shown up to here that, in small strain conditions, the difference of terms being defined either in the initial state or in the final state is not being obvious. In contrast, in large strain conditions, the difference becomes considerable by taking into account of the multiplication terms. For the non-linear case, the term strain is being rather a mathematical definition, the physical meaning not directly being obvious to a user in contrast to the engineering definition of strain. In large deformation analysis, length square change is employed, while in the small strain analysis, simply the length change is used. That creates consistency problems to deal with in relating stress to strain. This will be made clear in the sequel of this chapter. It must be stated that no further terms exist in the expression for for *Green's strain* and *Almansi's strain* as given by Eqs. (2.39) and (2.40) i.e. no Taylor's expansion and truncation of higher order terms have been performed. That is, they are complete. In the following, the definition of stress will be given.

2.3 STRESS

In this section, the same approach to explanation of stress for large deformation analysis and limiting case for small deformation solutions will be followed as in the previous. Again, one needs to distinguish the initial and the current states for the definition of stress. Although in literature many different stress definitions exist, in this dissertation, only the two of them will be considered since they are applied in the written program for including geometric non-linearity. They are the *Cauchy stress* and the *Second Piola Kirchoff stress*. Except the *Cauchy stress*, the stress definitions have a rather mathematical meaning; they are in general not attributed to a direct physical meaning. Although *Cauchy stress* has a meaning in engineering point of view, the *Cauchy stress* varies under rotations, which creates difficulties in some FEM formulations. That is why different stress definitions exist in literature. The conversion is virtually always possible from one definition to the other by use of the deformation gradient, or components of it. The main reason for selection of one or the other is the computational efficiency. In the context of this dissertation, consideration will not be given to all the stress definitions, but two of them, which are applied in the formulations and solutions in this dissertation.

In the engineering point of view, *Cauchy stress*, which is defined in the current state, has major importance and meaning, and it will be obtained. Other definitions are, in general, means to reach *Cauchy stresses*, and they are rather mathematical expressions.

Now consider a cut on a body in current state on which some forces and tractions are acting (Figure 2.4).

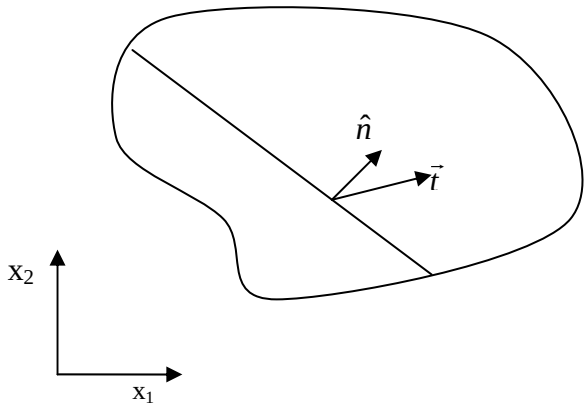


Figure 2.4: A cut on a body in current state and traction defined on the surface per unit area.

The traction on the surface is defined by the formulae below:

$$\mathbf{t} = \hat{\mathbf{n}}^T \cdot \boldsymbol{\sigma} ,$$

or in indicial notation:

$$t_i = \hat{n}_j \cdot \sigma_{ji} \tag{2.42}$$

where, $\mathbf{t}$ is the traction vector, $\hat{\mathbf{n}}$ is the unit normal vector on the cut and $\boldsymbol{\sigma}$, a second order tensor, is defined to be the *Cauchy's stress tensor*. In engineering analysis, it has a major importance and has a physical meaning. The first index represents the cut normal direction, and the second index represents the direction of the traction with respect to the reference frame in the current state. It is written in matrix form as:

$$\sigma = \begin{bmatrix} \sigma_{11} & \sigma_{12} & \sigma_{12} \\ \sigma_{21} & \sigma_{22} & \sigma_{23} \\ \sigma_{31} & \sigma_{32} & \sigma_{33} \end{bmatrix}$$

2.43

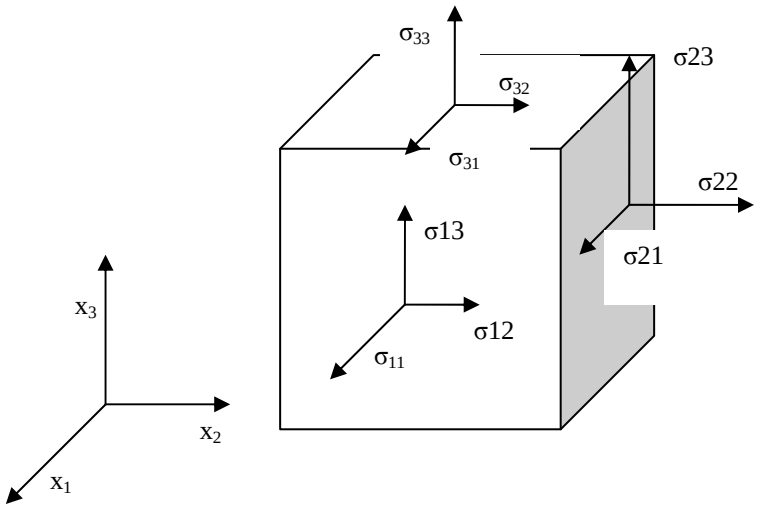


Figure 2.5: Cauchy stress tensor components in current state.

Now one may consider the traction on a body, and want to retrieve the fictitious same traction defined in the initial state such that:

$$\mathbf{t} \cdot dA = \mathbf{T} \cdot dA_0$$

(2.44)

where **t** is the traction in the current state, dA is the infinitesimal area in the current state, **T** is the assumed fictitious same traction corresponding to reference state, and dA_0 is the same area in the reference state.

Next one needs the formulation of the area change from the current state to the initial state or vice versa for formulating the transformation:

$$\hat{n}_i dA = \varepsilon_{ijk} dx_j dx_k \tag{2.45}$$

$$\hat{N}_R dA_0 = \varepsilon_{RJK} dX_J dX_K \tag{2.46}$$

Eq. (2.45) can be converted to current state by:

$$\varepsilon_{rjk} F_{jJ} dX_J F_{kK} dX_K \delta_{ri} \equiv \varepsilon_{rjk} F_{rR} F_{jJ} F_{kK} dX_J dX_K F_{iR}^{-1},$$

or,

$$\varepsilon_{RJK} \det F \cdot dX_J dX_K F_{iR}^{-1} \equiv \det F \cdot \hat{N}_R dA_0 \cdot F_{iR}^{-1} \tag{2.47}$$

The above can be written in matrix form as:

$$\hat{\mathbf{n}} \cdot dA = \det F \cdot dA_0 \cdot \mathbf{F}^{-T} \cdot \hat{\mathbf{N}} \tag{2.48}$$

For derivation of **T** in Eq. (2.44)**,** another stress definition in the initial state may be written such that:

$$\hat{\mathbf{N}}^T \cdot \mathbf{P} = \mathbf{T} \tag{2.49}$$

In the above equation, P is defined to be the nominal stress tensor. Combining Eqs. (2.44), (2.48), and (2.49):

$$\hat{\mathbf{N}}^T \cdot \mathbf{P} \cdot dA_0 = \det F \cdot dA_0 \cdot \hat{\mathbf{N}}^T \cdot \mathbf{F}^{-1} \cdot \boldsymbol{\sigma} \tag{2.50}$$

Then from above it can easily be deduced that:

$$\mathbf{P} = \det F \cdot \mathbf{F}^{-1} \cdot \boldsymbol{\sigma} \tag{2.51}$$

Here **P** is not symmetric in general, and changes under rotations. Due to this reason, in general it is not used in this form. It is transformed by multiplying both sides by $\mathbf{F}^{-T}$, and another stress definition; *Second Piola Kirchoff stress* is obtained as:

$$\mathbf{S} = \det F \cdot \mathbf{F}^{-1} \cdot \boldsymbol{\sigma} \cdot \mathbf{F}^{-\mathrm{T}} \tag{2.52}$$

Second Piola Kirchoff stress is a symmetric second order tensor. For small deformation case, **F** is approximate to identity, *detF* is approximately 1, and thus no considerable difference between the *Cauchy stress*, *Nominal stress*, and the *Second Piola Kirchoff stress* is observed. Nevertheless, in *non-linear elasticity* or *non-linear plastic* analysis, where there exist large straining, and large deformations, the analyst must perform the operations either in the current state or in the initial state. In case of performing the operations in the initial state, **S** must be used for the stress definition. However, in case the analysis is performed in the current state, *Cauchy stress* or some other variants should be used. **S** may be considered as the *pull back* of **σ** from current state to initial state. S is frame indifferent as will be shown in the sequel. Thus, **S** is preferred in some analysis when frame indifference is to be considered, but the *pull back* and *push forward* operations constitute a large amount of work. For this reason, sometimes, invariant variances of *Cauchy stress* are preferably used in some analysis, where stress incrementation is necessary. This fact will not be dealt here.

2.4 FRAME INDIFFERENCE

Since in the above definitions of stress or strain, frame indifference is declared, for the continuity of the subject, this concept will be discussed a little. Assume that the body dealt with makes a rigid body motion in which there is only translation and rotation, where there is no deformation except the previous stresses and strains remain intact. Now, regard the stress and the strain definitions on the body, and compare those stress or strain definitions for both states.

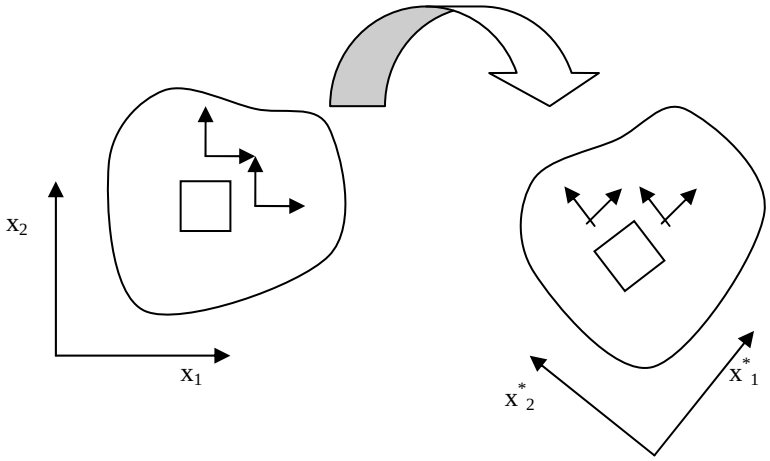


Figure 2.6: A deformable object making rigid body motion; translating and rotating in space. With respect to the frame of the body, stress and strains remain unchanged. However for the fixed frame, at the initial state some definitions of stress or strains change.

The motion of the body can be defined as:

$$\mathbf{x}^* = \mathbf{Q}(t) \cdot \mathbf{x} + \mathbf{c}(t),$$
$$\mathbf{Q}^{-1} = \mathbf{Q}^T \tag{2.53}$$

where **x** is the coordinate in the current state, **c**(t) represents the translation of the object and **Q**(t) represents only the rotational motion. The deformation gradient in this case would constitute of the rotation part only. Now see what happens to the stress or the strain tensors as the body rotates.

Considering the deformation gradient, the equation below must hold:

$$\mathbf{dx}^* = \mathbf{Q} \cdot \mathbf{dx} = \mathbf{Q} \cdot \mathbf{F} \cdot \mathbf{dX} = \mathbf{F}^* \cdot \mathbf{dX} \tag{2.54}$$

Thus, from the above equation, it can be concluded that the deformation gradient transforms like a vector under rotations of the object with respect to the initial frame, which can be stated mathematically as:

$$\mathbf{F}^{*} = \mathbf{Q} \cdot \mathbf{F} \tag{2.55}$$

When one looks at the rotated *Cauchy stress* from the initial x-frame, then:

$$\boldsymbol{\sigma}^{*} = \mathbf{Q} \cdot \boldsymbol{\sigma} \cdot \mathbf{Q}^{T} \tag{2.56}$$

and it is obvious that it does not remain the same under rotations. Therefore, it rotates with the rotating frame.

Looking at the *Second Piola Kirchoff stress*:

$$S^{*} = \det F \cdot \mathbf{F}^{*-1} \cdot \boldsymbol{\sigma}^{*} \cdot \mathbf{F}^{*-T} \equiv \det F \cdot \mathbf{F}^{-1} \cdot \underbrace{\mathbf{Q} \cdot \mathbf{Q}^{T}}_{\mathbf{I}} \cdot \boldsymbol{\sigma} \cdot \underbrace{\mathbf{Q} \cdot \mathbf{Q}^{T}}_{\mathbf{I}} \cdot \mathbf{F}^{-T} \tag{2.57}$$

Thus, from above it can easily be concluded that:

$$\mathbf{S}^{*} = \mathbf{S} \tag{2.58}$$

The above equation simply implies that **S** is frame indifferent, which means it does not change under rotations and/or translations of frame.

Continuing the procedure for the *Green-Lagrange deformation tensor*:

$$\mathbf{C}^{*} = \mathbf{F}^{*\mathbf{T}} \cdot \mathbf{F}^{*} = \mathbf{F}^{\mathbf{T}} \cdot \underbrace{\mathbf{Q}^{\mathbf{T}} \cdot \mathbf{Q}}_{\mathbf{I}} \cdot \mathbf{F} = \mathbf{C} \tag{2.59}$$

The above equation means that *Green Lagrange Deformation Tensor* is unaffected by rotations of the object. It implies also frame indifference of the *Green's Strain Tensor* **E.**

Considering the *Eulerian deformation tensor*:

$$\mathbf{b}^{*} = \mathbf{F}^{*} \cdot \mathbf{F}^{*\mathbf{T}} = \mathbf{Q} \cdot \mathbf{F} \cdot \mathbf{F}^{\mathbf{T}} \cdot \mathbf{Q}^{\mathbf{T}} \neq \mathbf{b} \tag{2.60}$$

From the above equation, it is seen that *Eulerian deformation tensor* rotates with the object. It directly implies that *Eulerian strain tensor* also rotates with the object rotation.

2.5 CONSTITUTIVE RELATIONS

Constitutive relations are the equations relating the strains to the stresses. For this kind of a relation to exist, a consistent material model is needed. In literature, many different material constitutive relations exist. For the linear *small deformation*, *small strain* analysis, the relation of stress to strain in 2D for *plane stress*, *plane strain* and *axisymmetric* cases are defined simply as below:

Plane Stress:

$$\begin{bmatrix} S_{11} \\ S_{22} \\ S_{12} \end{bmatrix} \cong \begin{bmatrix} \sigma_{11} \\ \sigma_{22} \\ \sigma_{12} \end{bmatrix} = \begin{bmatrix} \frac{\lambda(1-2\nu)}{(1-\nu)} + 2\mu & \frac{\lambda(1-2\nu)}{(1-\nu)} & 0 \\ \frac{\lambda(1-2\nu)}{(1-\nu)} & \frac{\lambda(1-2\nu)}{(1-\nu)} + 2\mu & 0 \\ 0 & 0 & \mu \end{bmatrix} \begin{bmatrix} \varepsilon_{11} \\ \varepsilon_{22} \\ 2\varepsilon_{12} \end{bmatrix} \tag{2.61}$$

Plane Strain:

$$\begin{bmatrix} S_{11} \\ S_{22} \\ S_{12} \end{bmatrix} \cong \begin{bmatrix} \sigma_{11} \\ \sigma_{22} \\ \sigma_{12} \end{bmatrix} = \begin{bmatrix} \lambda+2\mu & \lambda & 0 \\ \lambda & \lambda+2\mu & 0 \\ 0 & 0 & \mu \end{bmatrix} \begin{bmatrix} \varepsilon_{11} \\ \varepsilon_{22} \\ 2\varepsilon_{12} \end{bmatrix} \tag{2.62}$$

Axi-symmetric:

$$\begin{bmatrix} S_{11} \\ S_{22} \\ S_{12} \\ S_{33} \end{bmatrix} \cong \begin{bmatrix} \sigma_{11} \\ \sigma_{22} \\ \sigma_{12} \\ \sigma_{33} \end{bmatrix} = \begin{bmatrix} \lambda+2\mu & \lambda & 0 & \lambda \\ \lambda & \lambda+2\mu & 0 & \lambda \\ 0 & 0 & \mu & 0 \\ \lambda & \lambda & 0 & \lambda+2\mu \end{bmatrix} \begin{bmatrix} \varepsilon_{rr} \\ \varepsilon_{zz} \\ 2\varepsilon_{rz} \\ \varepsilon_{\theta\theta} \end{bmatrix} \tag{2.63}$$

In the above formulation, it should be noted that **S**, **σ**, and **ε** are written in vector form, which is called as *Voigt notation* in [5]. Unless otherwise stated, in the FEM context, they will be assumed in this vector form, but in the general continuum mechanics equations, they should be considered in the matrix form. Here, λ and μ are the *Lame's constants* defined as:

$$\lambda = \frac{E\nu}{(1+\nu)(1-2\nu)} \tag{2.64}$$

$$\mu = \frac{E}{2(1+\nu)} \tag{2.65}$$

The *non-linear case* is more complicated. Consistency becomes an important issue to deal with. In that case, the models are classified in general for path independence, reversibility, and non-dissipative behaviors (Belytschko et al. [5]). In this work, only two of the constitutive models are dealt with; one of which is the direct extension of the *Hooke's Law*, called the *Saint Venant-Kirchoff material model*, and the other is the *Hyperelastic Neo-Hookean material model*.

Kirchoff Material model is a model used in general for *large deformation – small strain* type of problems. It does not have much practical importance in general. It only includes the rotations of the body. In the small strain range, both of the strain matrices approximate to engineering definition of strains. That is why it is used in the small strain range. However, in case of the large straining, it results in stiffer results. The reason may be explained by the strain definition is changing considerably in case of large strain. Referring to initial state, the relation is defined as:

$$S_{IJ} = \widehat{C}_{IJKL} E_{KL}, \text{ or in matrix form: } \mathbf{S} = \widehat{\mathbf{C}} : \mathbf{E} \tag{2.66}$$

where, **S** is the *Second Piola Kirchoff stress tensor*, and **E** is the *Green's strain tensor*, which are defined before, and $\widehat{\mathbf{C}}$ is the constant constitutive matrix defined in the assumed unstressed initial state. When referring to current state, stress or strain definitions should change accordingly as:

$\sigma_{ij} = \hat{c}_{ijkl} e_{kl}$, or in matrix form: $\boldsymbol{\sigma} = \hat{\mathbf{c}} : \mathbf{e}$

(2.67)

where **σ** is the *Cauchy stress*, and **e** is the *Almansi's strain tensor* as are previously defined, and $\hat{\mathbf{c}}$ is the constitutive matrix, changing as the deformation state changes. The change of $\hat{\mathbf{c}}$ is defined by the transformation rule from the unstressed initial state constitutive matrix according to the formulation:

$$\hat{c}_{ijkl} = \frac{1}{\det F} F_{iI} F_{jJ} F_{kK} F_{lL} \hat{C}_{IJKL}$$

(2.68)

Hyperelastic model is used for large deformation and large strain analysis. In this model, stored strain energy potential is defined as a function of *Green's deformation tensor* or *Green Lagrange strain tensor*. Stresses and constitutive relations are obtained accordingly from the potential function. This formulation has variations [5], [6], but the one in [6] is adapted for the program developed. Hyperelastic model guarantees path independent work and is more consistent with the non-linear stress definitions. In terms, this means:

$$\mathbf{S} = 2\frac{\partial \Psi(\mathbf{C})}{\partial \mathbf{C}} = \frac{\partial w(\mathbf{E})}{\partial \mathbf{E}}$$

(2.69)

In the above formulation, Ψ is the potential defined for the *Green's deformation tensor*, while *w* is the potential defined for *Green's strain tensor*. The transformation to the current state is performed by use of Eq. (2.52) as:

$$\boldsymbol{\sigma} = \frac{2}{\det F} \cdot \mathbf{F} \cdot \frac{\partial \Psi}{\partial \mathbf{C}} \cdot \mathbf{F}^T = \frac{1}{\det F} \cdot \mathbf{F} \cdot \frac{\partial w}{\partial \mathbf{E}} \cdot \mathbf{F}^T$$

(2.70)

The constitutive matrices are derived from Eq. (2.69) by taking one more derivative, which yields:

$$\hat{\mathbf{C}} = 4 \cdot \frac{\partial^2 \Psi}{\partial \mathbf{C} \partial \mathbf{C}} = \frac{\partial^2 w}{\partial \mathbf{E} \partial \mathbf{E}}$$

(2.71)

The above equation is also transformed to the current state as written in Eq. (2.68). These formulations are general to *hyperelastic constitutive model*.

Now there comes the definition of the energy function. For the *compressible Neo-Hookean hyperelastic model*, the energy function is defined as ([6], Chapter 7):

$$\Psi(\mathbf{C}) = \frac{\lambda}{4} \cdot ((\det F)^2 - 1) - (\frac{\lambda}{2} + \mu) \cdot \ln(\det F) + \frac{1}{2} \cdot \mu \cdot (tr(\mathbf{C}) - 3) \tag{2.72}$$

Then substituting equations (2.69) (2.71) in (2.72) gives:

$$\mathbf{S} = \frac{\lambda}{2} \cdot ((\det F)^2 - 1) \cdot \mathbf{C}^{-1} + \mu \cdot (\mathbf{I} - \mathbf{C}^{-1}),$$

or in indicial form:

$$S_{IJ} = \lambda \cdot ((\det F)^2 - 1) \cdot C_{IJ}^{-1} + \mu \cdot (\delta_{IJ} - C_{IJ}^{-1}) \tag{2.73}$$

$$\widehat{\mathbf{C}}_{IJKL} = \lambda \cdot (\det F)^2 \cdot \mathbf{C}_{IJ}{}^{-1}\mathbf{C}_{KL}^{-1} + \frac{1}{2}(2\mu - \lambda((\det F)^2 - 1) \cdot (\mathbf{C}_{IK}^{-1}\mathbf{C}_{JL}^{-1} + \mathbf{C}_{IL}^{-1}\mathbf{C}_{JK}^{-1}) \tag{2.74}$$

are obtained. Eqs. (2.73) and (2.74) are defined for the *Total Lagrange Approach*, which means calculations are performed in the initial state. The same transformations as in Eqs. (2.52), and (2.68) apply for the current state calculations; stresses and strains are also defined for *Updated Lagrange Approach* as follows:

$$\boldsymbol{\sigma} = \frac{\lambda}{2\det F} \cdot ((\det F)^2 - 1) \cdot \mathbf{I} + \frac{\mu}{\det F} \cdot (\mathbf{b} - \mathbf{I}),$$

or in indicial form:

$$\sigma_{ij} = \frac{\lambda}{2\det F}((\det F)^2 - 1)\delta_{ij} + \frac{\mu}{\det F}(b_{ij} - \delta_{ij}) \tag{2.75}$$

$$c_{ijkl} = \lambda(\det F)^2 \delta_{ij}\delta_{kl} + \frac{1}{2}(2\mu - \lambda((\det F)^2 - 1) \cdot (\delta_{ik}\delta_{jl} + \delta_{il}\delta_{kj})$$

(2.76)

In "*Voigt notation*" (Belytschko et al. [5]), the *constitutive matrix* can be written in 2D for *updated Lagrange method* as:

Plane Stress:

$$\hat{\mathbf{c}} = \begin{bmatrix} \frac{\lambda(1-2\nu)}{(1-\nu)} + 2\mu & \frac{\lambda(1-2\nu)}{(1-\nu)}(\det F)^2 & 0 \\ \frac{\lambda(1-2\nu)}{(1-\nu)}(\det F)^2 & \frac{\lambda(1-2\nu)}{(1-\nu)} + 2\mu & 0 \\ 0 & 0 & \mu - \frac{\lambda(1-2\nu)}{2(1-\nu)}((\det F)^2 - 1) \end{bmatrix}$$

(2.77)

Plane Strain:

$$\hat{\mathbf{c}} = \begin{bmatrix} \lambda + 2\mu & \lambda(\det F)^2 & 0 \\ \lambda(\det F)^2 & \lambda + 2\mu & 0 \\ 0 & 0 & \mu - \frac{\lambda}{2}((\det F)^2 - 1) \end{bmatrix}$$

(2.78)

Axi-symmetric:

$$\hat{\mathbf{c}} = \begin{bmatrix} \lambda + 2\mu & \lambda(\det F)^2 & 0 & \lambda(\det F)^2 \\ \lambda(\det F)^2 & \lambda + 2\mu & 0 & \lambda(\det F)^2 \\ 0 & 0 & \mu - \frac{\lambda}{2}((\det F)^2 - 1) & 0 \\ \lambda(\det F)^2 & \lambda(\det F)^2 & 0 & \lambda + 2\mu \end{bmatrix}$$

(2.79)

For the *Kirchoff model*, at the initial undeformed state, the constitutive matrix is the same as in the linear case. Nevertheless, for the deformed state, the transformations must be performed according to Eq. (2.68). Those transformations are done for the fourth order tensor, than reduced to second order tensor in *Voigt notation* form.

It should be stressed here that, in the h*yperelastic model*, the stresses are calculated from Eqs. (2.73) or (2.75) and the *constitutive matrix* from Eqs. (2.74) or (2.76).

In the *plane stress* analysis, for the stress calculations, λ should be changed accordingly as:

$$\lambda_{plstrs} = \lambda \cdot \frac{1-2\cdot\nu}{1-\nu} = \frac{E\cdot\nu}{1-\nu^2} \tag{2.80}$$

In addition, *constitutive matrix* should be changed accordingly for the *plane stress* analysis. For the *updated Lagrange formulation* it is written as in Eq. (2.77) but for the *total Lagrange formulation* it must be written as below:

$$\widehat{C}_{IJKL} = \lambda \cdot \frac{(1-2\cdot\nu)}{(1-\nu)} \cdot (\det F)^2 \cdot C_{IJ}^{-1} C_{KL}^{-1} + (\mu - \frac{\lambda \cdot (1-2\cdot\nu)}{2} \frac{}{(1-\nu)} \cdot ((\det F)^2 - 1)) \cdot (C_{IK}^{-1} C_{JL}^{-1} + C_{IL}^{-1} C_{JK}^{-1}) \tag{2.81}$$

2.6 CONSERVATION EQUATIONS

Having defined the stress, strain, and the constitutive relations, conservation equations may be stated briefly as promised in the previous sections. In the framework of continuum mechanics, four conservation equations related to the context may be defined, namely the *mass conservation, linear momentum conservation, angular momentum conservation* and the *energy conservation*.

2.6.1 Mass Conservation

In Newtonian mechanics mass is conserved. That is, no mass is lost, and no mass is produced during deformation of a body. In FEM context it is not used, or included in to the equations directly, but its result is used indirectly. In the mathematical form, mass conservation may be stated as:

$$\int \rho_0 \cdot dV_0 = \int \rho \cdot dV \tag{2.82}$$

In the above equation, ρ_0 is the mass density in the reference configuration, while ρ is the mass density in the current configuration. Considering Eq. (2.8) and the integral Eq. (2.82), the following formula can be written:

$$\frac{\rho}{\rho_0} = \frac{1}{\det F} \tag{2.83}$$

If *detF* approaches zero and if $\rho_0 \neq 0$, ρ approaches infinity, which is not admissible; then *detF* must be greater than zero.

Here it should be emphasized that, the above equation is written for the Lagrangian mesh, since we are dealing with this kind of a system as stated previously. For Eulerian meshes, it should be stated in a different form [5].

2.6.2 Linear Momentum Conservation

Newton's second law states that the rate of linear momentum is equal to the applied external forces. In the quasi-static case, where the forces are applied slowly, i.e. acceleration terms are omitted and the motion is independent of time, the linear momentum equation reduces to equilibrium equations. Here, it is sufficient to give the direct result of linear momentum equation, be the equilibrium equation as:

$$\nabla \cdot \boldsymbol{\sigma} + \rho \cdot \mathbf{f}_B = 0 \tag{2.84}$$

where $\boldsymbol{\sigma}$ is the *Cauchy stress*, ρ is the mass density and $\mathbf{f_B}$ is the internal body force per unit mass. It should be obvious to the reader that the above equation is defined over the current state of the body.

2.6.3 Angular Momentum Conservation

The angular momentum is obtained by the cross product of the terms in linear momentum equation by the position vector. The direct result is the symmetry of the *Cauchy stress tensor*:

$$\sigma = \sigma^T \tag{2.85}$$

This result is important and is used to reduce the number of equations to be solved. No other result is obtained by angular momentum conservation equation.

2.6.4 Energy Conservation

Energy relations constitute the mainframe of FEM method. In the context of mechanical problems, the sum of work done by internal stresses and external forces must be minimized. In the actual case of the energy conservation, one has the equilibrium of internal and external energy rates, but time variations are not in the context of this dissertation. Here one is only interested in the form of internal work done by the internal elastic stresses and external work done by the external forces instantaneously, which is called the *quasi-static* case.

As may be seen from the previous section, *Second Piola Kirchoff stress* is related to the *Green's Strain tensor*, while *Cauchy stress tensor* is related to the *Almansi's strain tensor*. In literature, they are defined to be work conjugate. Internal Energy forms for both the *total Lagrangian* and *updated Lagrangian* formulations may be written for the *Kirchoff material model* as:

$$\Pi_{\text{int}} = \frac{1}{2}\int_{V_0} \mathbf{S} : \mathbf{E} dV_0 = \frac{1}{2}\int_{V} \boldsymbol{\sigma} : \mathbf{e} dV \tag{2.86}$$

It should be noted for the above equation that, in the first integral, integration is performed in the initial volume, whereas in the second, integration is performed in the current volume. Conversion from one to other is easy, which means they are dependent and equivalent. Both of them should give the same result as long as the conversions from one state to another are performed consistently. One of them is selected for internal energy calculations. Depending on the selection of the integration form, formulation is called either *Total Lagrangian*, or *Updated Lagrangian*, respectively.

In case of the *Hyperelastic material model*, the above can be written as the integration of Eq. (2.72) as:

$$\Pi_{\text{int}} = \int_{V_0} \Psi(\mathbf{C}) dV_0 \tag{2.87}$$

The external work done by external forces may be formulated as follows:

$$\Pi_{ext} = -\int_{V_0} \mathbf{u}^T \cdot \rho_0 \mathbf{f}_B dV_0 - \int_{\Gamma_{0_t}} \mathbf{u}_t^T \cdot \mathbf{t}_t d\Gamma_{0_t} - \int_{\Gamma_{0_u}} \mathbf{u}_u^T \cdot \mathbf{t}_u d\Gamma_{0_u} \tag{2.88}$$

In this equation, the first integral is the work done by the internal body forces (i.e., magnetic, gravity, etc.). Here ρ_0 is the mass density defined in the initial state, $\mathbf{f_B}$ is the body force per mass. The second integral is the work done by the tractions on traction-defined surface. The third integral is the work done by the displacements of the restrained nodes on restraint surface. The above formulation in Eq. (2.88) in the initial state can also be written in the current state without loss of generality as:

$$\Pi_{ext} = -\int_{V} \mathbf{u}^T \cdot \rho \mathbf{f}_B dV - \int_{\Gamma_t} \mathbf{u}_t^T \cdot \mathbf{t}_t d\Gamma_t - \int_{\Gamma_u} \mathbf{u}_u^T \cdot \mathbf{t}_u d\Gamma_u \tag{2.89}$$

The difference between equations (2.88) and (2.89) is that, in Eq. (2.87) the integrations are performed in the initial volume or surface, but in Eq. (2.88), the integrations are performed in the current volume or surface. However, they are in general equivalent, and since the follower forces are not dealt with, for either *total Lagrange formulation* or *updated Lagrange formulation*, Eq. (2.88) may be utilized for the prescribed forces and displacements. That is, non-linear forces are not used in the implementation program. Non-linearity is only associated with the internal strain energy.

CHAPTER 3

FEM FORMULATION

3.1 INTRODUCTION

In this chapter, element formulation for plane stress, plane strain and axisymmetric, 2D elastic solutions and implementation of FEM is going to be investigated. This part will be presented here only for the completeness of the subject matter. It will not be elaborately dealt with since there are numerous books and publications about this issue. The interested reader should refer to references [2]-[5] for a deeper understanding.

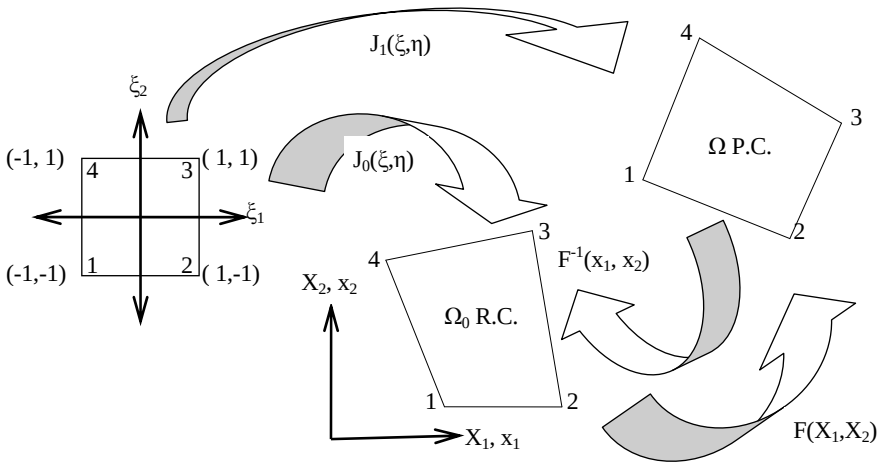


Figure 3.1: A FEM element patch defined in initial state mapped to current state. Also mapping from master element to both initial and current states of the element is represented.

In the context of FEM, the approach is to discretize a complicated body Ω into simpler rectangular or triangular patches Ω_i, and let connectivity between them. Those patches are further transformed to base square parametric elements for being able to do the Gauss integrations, and a switch between the base element and the actual patch is realized with the Jacobian transformation (Figure 3.1).

3.2 FORMULATION OF STRAINS

In the FEM context, information is lumped in the nodes, and values are interpolated on rectangular or triangular elements from the nodes by utilization of the *Lagrange interpolation functions*. Displacement formulation is followed, that is displacement form of strains, Eq. (2.39) or Eq. (2.40) is used. To be able to make calculations, the derivatives with respect to spatial coordinates are needed; which requires following some sub steps and equations.

For the elements, iso-parametric element formulation is being used. That is, the same interpolation functions are utilized for the interpolation of coordinates and the displacements at a point in an element. At an interior point of the element domain, a variable may be found by multiplying each nodal value by the corresponding nodal base function value at the point. Actually, the base functions define the weight of the corresponding node at a point in the domain of the element. In mathematical terms:

$$
\begin{aligned}
X_i &= \sum_{I=1}^{N} \phi_I(\xi) \cdot X_i^I, \\
x_i &= \sum_{I=1}^{N} \phi_I(\xi) \cdot x_i^I \\
u_i &= \sum_{I=1}^{N} \phi_I(\xi) \cdot u_i^I
\end{aligned}
\tag{3.1}
$$

Here, *i* refers to the spatial index, where for the 2D case it would assume the values 1,2, and in the 3D the values 1,2,3. The index *I* refers to the local index of the nodes on the element and ranges from 1 to *N*, where *N* is the number of nodes of the element. The above may also be written in matrix form:

$$\begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \underbrace{\begin{bmatrix} \phi_1(\xi) & 0 & \phi_2(\xi) & 0 & \phi_3(\xi) & \cdots \\ 0 & \phi_1(\xi) & 0 & \phi_2(\xi) & 0 & \cdots \end{bmatrix}_{2x2N}}_{\mathbf{H}} \begin{bmatrix} u_1^1 \\ u_2^1 \\ u_1^2 \\ u_2^2 \\ u_1^3 \\ \vdots \end{bmatrix}_{2Nx1}$$

(3.2)

As may be seen in previous chapter, one will need the derivatives of the displacements with respect to spatial coordinates. This is achieved by the Jacobian transformation written by the formula:

$$\frac{\partial(\bullet)}{\partial \xi_i} = \frac{\partial x_j}{\partial \xi_i} \cdot \frac{\partial(\bullet)}{\partial x_j}$$

(3.3)

in which, (•) refers to any variable defined in the domain to take the derivative, and is straightforward. In the two dimensional case, it may be written in open form as:

$$\begin{bmatrix} \frac{\partial(\bullet)}{\partial \xi_1} \\ \frac{\partial(\bullet)}{\partial \xi_2} \end{bmatrix} = \underbrace{\begin{bmatrix} \frac{\partial x_1}{\partial \xi_1} & \frac{\partial x_2}{\partial \xi_1} \\ \frac{\partial x_1}{\partial \xi_2} & \frac{\partial x_2}{\partial \xi_2} \end{bmatrix}}_{\mathbf{J}} \begin{bmatrix} \frac{\partial(\bullet)}{\partial x_1} \\ \frac{\partial(\bullet)}{\partial x_2} \end{bmatrix}$$

(3.4)

The Jacobian **J** in the above equation may be expressed more clearly for transformations to the reference state and transformations to the current state respectively as:

$$J_0(\xi) = \begin{bmatrix} \sum_{I=1}^{N} \frac{\partial \phi_I(\xi_1,\xi_2)}{\partial \xi_1} \cdot X_1^I & \sum_{I=1}^{N} \frac{\partial \phi_I(\xi_1,\xi_2)}{\partial \xi_1} \cdot X_2^I \\ \sum_{I=1}^{N} \frac{\partial \phi_I(\xi_1,\xi_2)}{\partial \xi_2} \cdot X_1^I & \sum_{I=1}^{N} \frac{\partial \phi_I(\xi_1,\xi_2)}{\partial \xi_2} \cdot X_2^I \end{bmatrix}$$

$$J(\xi) = \begin{bmatrix} \sum_{I=1}^{N} \frac{\partial \phi_I(\xi_1,\xi_2)}{\partial \xi_1} \cdot x_1^I & \sum_{I=1}^{N} \frac{\partial \phi_I(\xi_1,\xi_2)}{\partial \xi_1} \cdot x_2^I \\ \sum_{I=1}^{N} \frac{\partial \phi_I(\xi_1,\xi_2)}{\partial \xi_2} \cdot x_1^I & \sum_{I=1}^{N} \frac{\partial \phi_I(\xi_1,\xi_2)}{\partial \xi_2} \cdot x_2^I \end{bmatrix}$$

(3.5 a, b)

As stated in the above paragraphs, in the actual case, the derivatives with respect to true global coordinates are necessary, which may be obtained by rearrangement of Eq. (3.3) or (3.4).

$$\frac{\partial(\bullet)}{\partial \mathbf{x}} = \mathbf{J}^{-1}(\xi) \cdot \frac{\partial(\bullet)}{\partial \zeta}, or$$
$$\frac{\partial(\bullet)}{\partial \mathbf{X}} = \mathbf{J}_0^{-1}(\xi) \cdot \frac{\partial(\bullet)}{\partial \zeta} \tag{3.6}$$

Now that those transformations from base element to real elements in reference and current states have been defined, the transformation from reference to current, or the inverse relation may be expressed. It is actually the deformation gradient **F,** or the inverse of it, defined in the previous chapter. It should be stated here that multiplicative decomposition is valid for the deformation gradient:

$$\mathbf{J}(\xi)^T = \mathbf{F} \cdot \mathbf{J_0}(\xi)^T \tag{3.7}$$

In Eq. (3.7), the only unknown is the deformation gradient **F.** By rearranging the terms:

$$\mathbf{F}(\xi) = \mathbf{J}(\xi)^T \cdot \mathbf{J}_0^{-1}(\xi)^T \tag{3.8}$$

This equation requires the Jacobian to be invertible. One may note here that too much distorted elements may jeopardize the inversion of **J** for its determinant may be too small in that case.

For the linear analysis, which is the simple case and constitute the beginning point for the non-linear analysis, the engineering strain **ε** may be written as:

$$\varepsilon = \begin{bmatrix} \varepsilon_{11} \\ \varepsilon_{22} \\ \gamma_{12} = 2\varepsilon_{12} \end{bmatrix} = \begin{bmatrix} \dfrac{\partial u_1}{\partial x_1} \\ \dfrac{\partial u_2}{\partial x_2} \\ \dfrac{\partial u_1}{\partial x_2} + \dfrac{\partial u_2}{\partial x_1} \end{bmatrix} \tag{3.9}$$

which can be written in open form for 2D as:

$$\varepsilon = B \cdot u,$$
$$where:$$
$$B = \begin{bmatrix} \dfrac{\partial}{\partial x_1} & 0 & \dfrac{\partial}{\partial x_1} & 0 & \cdots \\ 0 & \dfrac{\partial}{\partial x_2} & 0 & \dfrac{\partial}{\partial x_2} & \cdots \\ \dfrac{\partial}{\partial x_2} & \dfrac{\partial}{\partial x_1} & \dfrac{\partial}{\partial x_2} & \dfrac{\partial}{\partial x_1} & \cdots \end{bmatrix}_{3x2N}, u = \begin{bmatrix} u_1^1 \\ u_2^1 \\ u_1^2 \\ u_2^2 \\ \vdots \end{bmatrix}_{2N} \tag{3.10}$$

The above formula is standard in FEM formulations and the same notational convention is used in virtually all of the FEM books. Note that, the derivatives are with respect to the current state coordinates, but the reader should know once more that for the linear analysis, the distinction between the initial or the current state becomes invisible. Thus taking the derivatives with respect to initial state is also valid, and implemented so for the linear analysis. However, for the non-linear analysis, this distinction will be important, and **B** will be called $\mathbf{B_0}$ when referring to initial state from here and after.

The strains seen in Eq. (2.39) or (2.40) are split into linear part and non-linear part. The nonlinear part has been treated in linearization of equations. For the formulation, Bathe [2], Chapter 6 is followed. The linear part is dealt within the Newton iterations as described in the sequel.

3.3 RECTANGULAR ELEMENT FORMULATION

A rectangular element has at least four nodes, thus has at least four interpolation functions for each node. For the i^{th} node, i^{th} interpolating function is assuming the value 1.0. If the element is of second order, it may have up to nine nodes. Nine-node rectangular element would be complete second order. In the implementation program, only four node rectangular elements have been implemented in the graphical interface, however beneath the graphical interface, up to eight nodes can be implemented. The four interpolation functions may be written as follows:

$$\phi_i(\xi_1,\xi_2)=\frac{1}{4}(1+\xi_1^i\cdot\xi_1)(1+\xi_2^i\cdot\xi_2),\quad i=1,2,3,4 \tag{3.11}$$

For details and higher order element formulation, reference [4] may be followed.

The strains are defined as a 2^{nd} order tensor, but *Voigt notation*, which reduces the second order tensor to a first order tensor is preferred in general for the FEM formulation (Belytschko et al. [5]).

3.4 TRIANGULAR ELEMENT FORMULATION

For the triangular element, area coordinates are used. A triangular element has at least three nodes, thus it has at least three interpolation functions for each node. For the i^{th} node, i^{th} interpolating function is assuming the value 1.0 in the same way as the rectangular element. If the element is of second order, it may have up to six nodes. Six-node triangular element would be complete second order. In the implementation program, only three node triangular elements have been implemented in the graphical interface, however beneath the graphical interface, up to six nodes can be implemented. The three interpolation functions may be written as follows:

$$\phi_1(\xi_1,\xi_2)=\xi_1=\frac{A_1}{A}$$
$$\phi_2(\xi_1,\xi_2)=\xi_2=\frac{A_2}{A}$$
$$\phi_3(\xi_1,\xi_2)=1-\xi_1-\xi_2=\frac{A_3}{A} \tag{3.12}$$

For details and higher order element formulation, reference [4] may be followed.

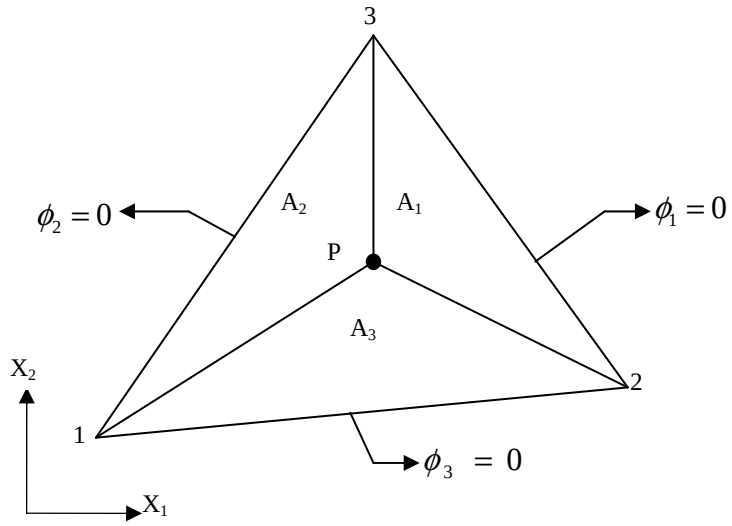


Figure 3.2: Triangular element area coordinates with total area A.

3.5 VARIATIONAL FORM

In the context of FEM, Eq. (2.84) is called the *strong form* of *momentum balance equation*. To obtain the weak form, that equation is multiplied with a variational displacement δ**u** called the *test function* and integrated over the body. The property of the test function is such that, it is zero over the displacement-defined boundary. It is actually a small perturbation to the system at the equilibrium:

$$\delta\Pi = \int_V \delta u_i \left(\frac{\partial \sigma_{ji}}{\partial x_j} + \rho \cdot f_{B_i}\right) \cdot dV = 0 \tag{3.13}$$

Integrating by parts and using the *Gauss Theorem* for the above equation:

$$\delta\Pi = \int_V \frac{\partial(\delta u_i)}{\partial x_j} \sigma_{ji} dV - \int_V \rho \cdot \delta u_i f_{B_i} \cdot dV - \int_{\Gamma_t} \delta u_i t_i \cdot d\Gamma = 0 \tag{3.14}$$

This the weak form of the *momentum equation*. Note that in the above equation, the boundary conditions are included. The weak form above implies the strong form. That is the strong form can be obtained from the weak form by one more integration by parts.

In the above integration formula, terms may be ascribed physical names. The first integral is called the virtual internal work, the second and the third terms are the virtual external work. In the first integral, the derivative of the variation of **u** may be split into symmetric and anti-symmetric parts. Since **σ** is defined to be symmetric in the previous chapter, multiplication with the anti-symmetric part vanishes. Then, the first integral above may be written as:

$$\int_V \frac{\partial(\delta u_i)}{\partial x_j} \sigma_{ij} dV = \int_V \frac{1}{2} \left(\frac{\partial \delta u_i}{\partial x_j} + \frac{\partial \delta u_j}{\partial x_i}\right) \sigma_{ij} dV \tag{3.15}$$

Now take the variation of the *Almansi's strain tensor* at equilibrium, expecting to get some interesting results.

$$\delta e = \frac{1}{2} \left(\frac{\partial \delta u_i}{\partial x_j} + \frac{\partial \delta u_j}{\partial x_i} - \underbrace{\frac{\partial \delta u_m}{\partial x_i} \cdot \frac{\partial u_m}{\partial x_j}}_{0} - \underbrace{\frac{\partial u_m}{\partial x_i} \cdot \frac{\partial \delta u_m}{\partial x_j}}_{0}\right) \tag{3.16}$$

Since at the solution point, u_m is assumes the displacement corresponding to the stationary point satisfying equilibrium, derivative with respect to x vanishes. Then the above integral in Eq. (3.15) becomes equivalent to:

$$\int_V \frac{\partial \delta u_i}{\partial x_j} \sigma_{ij} \cdot dV = \int_V \boldsymbol{\sigma}_{ij} \delta \mathbf{e}_{ij} \cdot dV$$

(3.17)

This equation may be compared to Eq. (2.86) and can be concluded to be the variational form of energy. Equivalently the variation of *Green Lagrange strain* can be taken and one more equivalence may be obtained. Thus, variational equality may be written:

$$\delta \Pi_{\text{int}} = \int_V \boldsymbol{\sigma} : \delta \mathbf{e} \cdot dV = \int_{V_0} \mathbf{S} : \delta \mathbf{E} \cdot dV_0$$

(3.18)

For small stress and small strain analysis, the above formula may be written as:

$$\delta \Pi_{\text{int}} = \delta \mathbf{u}^T \int_{V_0} \mathbf{B}_0{}^{\mathbf{T}} \cdot \widehat{\mathbf{C}} \cdot \mathbf{B}_0 \cdot dV_0 \cdot \mathbf{u}$$

(3.19)

in which, $\mathbf{B_0}$ is the derivative matrix defined in Eq. (3.10) with the only difference that the derivative is taken with respect to initial coordinates.

The variational form of the external work is the sum of the second and third integrals in Eq. (3.14). Since one is not interested in follower forces in the context of this dissertation, they are not changed much for the current and deformed states. One can write the variation of external forces as:

$$\delta \Pi_{ext} = -\int_V \rho \cdot \delta u_i f_{B_i} \cdot dV - \int_{\Gamma_t} \delta u_i t_i \cdot d\Gamma$$

(3.20)

Thus, a variational form is obtained, which will be helpful in the linearization of the general non-linear FEM. Since the summation of the internal and the external energies is supposed to be minimized, it is expected that the summation of the variations of the internal and the external energies vanish. Actually, Eq. (3.14) implies this result. In the next section, this variational form will be linearized for application of Newton algorithm to the solution.

3.6 FEM LINEARIZATION

For the details of this section, Bathe [2], Chapter 6, should be followed. Also in [4] and [7], the topic of FEM linearization has been considered. For solution of the non-linear equations, Newton solution technique has been applied, which requires linearization of the variation of the total energy. In general, it is required that the total energy is minimized, which requires that the gradient of total energy function vanishes (first order necessary condition). Another requirement would be the Hessian of the potential is positive definite (second order necessary condition). In this section, those concepts will be made comprehensible to the reader.

In the context of Newton Algorithm, the stationary point of a function is aimed, beginning from some initially assumed unstressed position by tangents to the function (Figure 3.3).

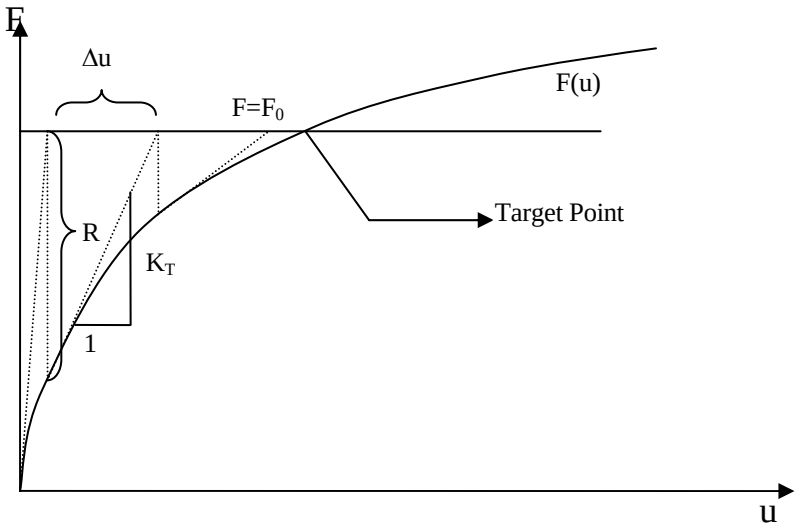


Figure 3.3: Representative Newton approximation scheme.

One is to solve the equations following in general:

$$K_T^t \cdot \Delta u^t = R^t,$$
$$u^{t+\Delta t} = u^t + \Delta u^t$$

(3.21)

Here, K_T^t is called the tangential stiffness at virtual time t, which in mathematical terms is the Hessian of the internal energy term and is always positive definite. Δu^t is the incremental displacement for current virtual time t, and R^t is the residual force at virtual time t.

The tangents of the function are needed in general. This is achieved by the linearization process. Consider beginning linearization on the initial configuration. It is already mentioned that, when everything is transformed to initial configurations and calculations are performed there, the method is called the *total Lagrange approach*. Now considering the rightmost integral in Eq. (3.18), the linearization can be written as:

$$\begin{aligned} D\delta\Pi_{\text{int}} &= \left.\frac{d}{d\varepsilon}\right|_{\varepsilon=0} \delta\Pi_{\text{int}}(\phi + \varepsilon\Delta u) = \int_{V_0} D(\delta\mathbf{E} : \mathbf{S}) dV_0 \\ &= \int_{V_0} (D(\delta\mathbf{E}) : \mathbf{S}) dV_0 + \int_{V_0} (\delta\mathbf{E} : D\mathbf{S}) dV_0 \end{aligned}$$

(3.22)

To achieve the linearization of the above expression, it is needed that some intermediate components be linearized.

The deformation gradient can be linearized as [7]:

$$\begin{aligned} D\mathbf{F} &= \left.\frac{d}{d\varepsilon}\right|_{\varepsilon=0} \mathbf{F}(\phi + \varepsilon\Delta\mathbf{u}) = \left.\frac{d}{d\varepsilon}\right|_{\varepsilon=0} \frac{\partial(\phi + \varepsilon\Delta\mathbf{u})}{\partial\mathbf{X}} \\ &= \frac{\partial\Delta\mathbf{u}}{\partial\mathbf{X}} = \nabla(\Delta\mathbf{u}) \cdot \mathbf{F} = \nabla_0(\Delta\mathbf{u}) \end{aligned}$$

(3.23)

in which, $\Delta\mathbf{u}$ is the small change in displacement as stated before, and $\nabla\mathbf{u}$ represents the gradient with respect to current coordinates, while $\nabla_0\mathbf{u}$ represents the gradient with respect to the initial coordinates.

The *Green Lagrange strain* can be linearized as follows [7]:

$$D\mathbf{E} = \frac{1}{2}\mathbf{F}^T(\nabla(\Delta\mathbf{u}) + \nabla(\Delta\mathbf{u})^T)\mathbf{F} = \frac{1}{2}(\mathbf{F}^T\nabla_0(\Delta\mathbf{u}) + (\nabla_0(\Delta\mathbf{u}))^T\mathbf{F}),$$

(3.24)

This is the pull back of the small strain defined at the current state to the initial state. It can also be written in terms of **u** and Δ**u** in indicial notation by use of Eq. (2.6) for deformation gradient **F** as:

$$\mathrm{D}(\mathrm{E}_{\mathrm{IJ}}) = \frac{1}{2}\left[\frac{\partial(\Delta u_I)}{\partial X_J} + \frac{\partial(\Delta u_J)}{\partial X_I} + \frac{\partial(\Delta u_M)}{\partial X_I}\cdot\frac{\partial u_M}{\partial X_J} + \frac{\partial u_M}{\partial X_I}\cdot\frac{\partial(\Delta u_M)}{\partial X_J}\right]$$

(3.25)

When looking at Eq. ((3.25) carefully, it should be recognized that, it is linear in Δ**u** when **u** is known. It can be written as:

$$D\mathbf{E} = (\mathbf{B_0} + \mathbf{B_1})^T\Delta\mathbf{u}$$

(3.26)

Here, $\mathbf{B_0}$ is defined in Eq.(3.10) and the paragraph following, with the only difference that, derivatives are taken with respect to current coordinates. Furthermore, $\mathbf{B_1}$ is defined as (Bathe [2], Chapter 6):

$$B_1 = \begin{bmatrix} u_{1,1}\phi^1_{,1} & u_{2,1}\phi^1_{,1} & u_{1,1}\phi^2_{,1} & u_{2,1}\phi^2_{,1} \\ u_{1,2}\phi^1_{,2} & u_{2,2}\phi^1_{,2} & u_{1,2}\phi^2_{,2} & u_{2,2}\phi^2_{,2} \\ u_{1,1}\phi^1_{,2} + u_{1,1}\phi^1_{,1} & u_{2,1}\phi^1_{,2} + u_{2,2}\phi^1_{,1} & u_{1,1}\phi^2_{,2} + u_{1,2}\phi^2_{,1} & u_{2,1}\phi^2_{,2} + u_{2,2}\phi^2_{,1} \\ \cdots & u_{1,1}\phi^N_{,1} & u_{2,1}\phi^N_{,1} & \\ \cdots & u_{1,2}\phi^N_{,2} & u_{2,2}\phi^N_{,2} & \\ \cdots & u_{1,1}\phi^N_{,2} + u_{1,2}\phi^N_{,1} & u_{2,1}\phi^N_{,2} + u_{2,2}\phi^N_{,1} & \end{bmatrix},$$

(3.27)

in which, $u_{I,M}$ is the derivative of displacements with respect to initial coordinates (I={1,2} and M={1,2} for 2D case). Furthermore, $\phi^k_{,J}$ is the derivative of kth base function with respect to initial coordinates (J={1,2} for 2D case) and is given in Eq. (3.6). This derivative is taken by use of the inverse Jacobian. The matrix in Eq. (3.27) is only used for the *Total Lagrange approach* as will be clearer in the sequel. It is not used for the *Updated Lagrange* case. Writing those statements in equation form to make things more clear:

$$u_{I,M} = \frac{\partial u_I}{\partial X_M} = \sum_{k=1}^{N} \phi^k_{,J} u^k, \text{ and}$$
$$\phi^k_{,J} = J^{-1}_{0(J,1)}(\xi_1,\xi_2)\frac{\partial \phi^k}{\partial \xi_1} + J^{-1}_{0(J,2)}(\xi_1,\xi_2)\frac{\partial \phi^k(\xi_1,\xi_2)}{\partial \xi_2},$$

J=1,2 for 2D case.

(3.28)

In Eq. (3.22), the derivation of D(δ**E)** and D**S** are also needed:

$$D\delta\mathbf{E} = D\left[\frac{1}{2}(\delta\mathbf{F}^T\mathbf{F} + \mathbf{F}^T\delta\mathbf{F})\right] = D\left[\frac{1}{2}[(\nabla_0(\delta\mathbf{u}^T))\cdot\mathbf{F} + \mathbf{F}^T(\nabla_0(\delta\mathbf{u}))]\right]$$
$$= \frac{1}{2}\left[(\nabla_0\delta\mathbf{u})^T(\nabla_0\Delta\mathbf{u}) + (\nabla_0\Delta\mathbf{u})^T(\nabla_0\delta\mathbf{u})\right]$$

(3.29)

Eq. ((3.29) can also be written in indicial form by use of Eq. (2.6) for deformation gradient **F** as:

$$D\delta E_{IJ} = \frac{1}{2}\left[\frac{\partial \delta u_M}{\partial X_I}\frac{\partial \Delta u_M}{\partial X_J} + \frac{\partial \Delta u_M}{\partial X_I}\frac{\partial \delta u_M}{\partial X_J}\right]$$

(3.30)

This is used in the first integral on the right hand side of Eq. (3.22). By use of the symmetry, the integrand may be written as:

$$\int_{V_0} D(\delta\mathbf{E}) : \mathbf{S} dV_0 = \delta\mathbf{u}^T \int_{V_0} \mathbf{B}^{\mathbf{T}}_{\mathbf{0}\,NL} \cdot \widehat{\mathbf{S}} \cdot \mathbf{B}_{\mathbf{0}\,NL} \cdot \Delta\mathbf{u} \cdot dV_0$$

(3.31)

where,

$$B_{0\,NL} = \begin{bmatrix} \phi^1_{,1} & 0 & \phi^2_{,1} & 0 & \phi^3_{,1} & 0 & \cdots & \phi^N_{,1} & 0 \\ \phi^1_{,2} & 0 & \phi^2_{,2} & 0 & \phi^3_{,2} & 0 & \cdots & \phi^N_{,2} & 0 \\ 0 & \phi^1_{,1} & 0 & \phi^2_{,1} & 0 & \phi^3_{,1} & \cdots & 0 & \phi^N_{,1} \\ 0 & \phi^1_{,2} & 0 & \phi^2_{,2} & 0 & \phi^3_{,2} & \cdots & 0 & \phi^N_{,2} \end{bmatrix}$$

(3.32)

The elements of the matrix in Eq. (3.32) are defined in Eq. (3.28). Note that **S** is defined a little differently to be able to write things in this form:

$$\widehat{S} = \begin{bmatrix} S_{11} & S_{12} & 0 & 0 \\ S_{21} & S_{22} & 0 & 0 \\ 0 & 0 & S_{11} & S_{12} \\ 0 & 0 & S_{21} & S_{22} \end{bmatrix}$$

(3.33)

Now, the only term, linearization of **S** with respect to **u,** is left. It can simply be written as:

$$D(\mathbf{S}) = \frac{\partial \mathbf{S}}{\partial \mathbf{E}} : D(\mathbf{E}) = \widehat{\mathbf{C}} : D(\mathbf{E}),$$
$$\text{or in indicial form :}$$
$$\mathrm{D(S_{IJ})} = \frac{\partial S_{IJ}}{\partial E_{KL}} D(E_{KL}) = \widehat{C}_{IJKL} : D(E_{KL})$$

(3.34)

Now one has tools for the linearization of the variational internal energy. Then Eq. (3.22) can be written more explicitly in the form:

$$\begin{aligned} D\delta\Pi_{\text{int}} &= \delta\mathbf{u}^{\mathbf{T}} \int_{V_0} (\mathbf{B_0} + \mathbf{B_1})^T : \widehat{\mathbf{C}} : (\mathbf{B_0} + \mathbf{B_1}) \Delta\mathbf{u} dV_0 \\ &\quad + \delta\mathbf{u}^{\mathbf{T}} \int_{V_0} \mathbf{B}_{0\,NL}^T \cdot \widehat{\mathbf{S}} \cdot \mathbf{B}_{\mathbf{0}\,NL} \Delta\mathbf{u} dV_0 \\ &= \delta\mathbf{u}^{\mathbf{T}} \mathbf{K}_{\mathbf{T}} \Delta\mathbf{u} \end{aligned}$$

(3.35)

where $\mathbf{K_T}$ is the tangential stiffness defined as the summation of the integrals. This is the end of the derivations for the linearization of the internal energy variation for the non-linear FEM, TL formulation.

Now considering the first integral in Eq. (3.18), when linearization is performed for this integral, the solution strategy is called the *updated Lagrange approach*. Nevertheless, this is not possible since the current state of the object is not known. However, there is the last state of the object at hand on which one can make all the linearization. In this case, one is still dealing with the *Lagrangian mesh*. So, transforming the previous linearization to the latest incremental state at hand is enough for the *updated Lagrange* formulation. In general, one may expect to get the same results for both of the methods. The choice on one to the other is the

computational efficiency on the specific problem type. For elasticity problems, no considerable gaining or loss have been observed on using one method to the other.

At this point note that when writing **x**, it is meant the last obtained incremental step of the body in consideration. Now formulate the same internal energy term at that state:

$$D\delta\Pi_{\text{int}} = \left.\frac{d}{d\varepsilon}\right|_{\varepsilon=0} \delta\Pi_{\text{int}}(\phi + \varepsilon\Delta u) = \int_{V_t} D(\delta E^t : S^t)dV_t$$
$$= \int_{V_t} (D(\delta E^t) : S^t)dV_t + \int_{V_t} (\delta E^t : DS^t)dV_t$$

(3.36)

Considering the system this way, one may obtain the same system of equations as in the above formulation for the TL formulation. There are only two differences to be considered. One is the use of $\mathbf{x}^{\mathbf{t}}$, which is the position at the latest state, and the other is getting rid of $\mathbf{B_1}$ term in the integration. The reason is that in Eq. ((3.25), in the last two terms, $\mathbf{u_M}$ is being actually equivalent to $\Delta\mathbf{u_M}$ since solution is being performed from the last incremental state. Then the integral formula in Eq. ((3.35) is converted to:

$$D\delta\Pi_{\text{int}} = \delta\mathbf{u}^{\mathbf{T}} \int_{V_t} \mathbf{B}^T : \hat{\mathbf{c}} : \mathbf{B} \cdot \Delta\mathbf{u} dV_t + \delta\mathbf{u}^{\mathbf{T}} \int_{V_t} \mathbf{B}_{NL} \cdot \hat{\boldsymbol{\sigma}} \cdot \mathbf{B}_{NL} \cdot \Delta\mathbf{u} dV_t$$
$$= \delta\mathbf{u}^{\mathbf{T}} \mathbf{K}_{\mathbf{T}} \Delta\mathbf{u}$$

(3.37)

Here, $\mathbf{K_T}$ is the tangential stiffness matrix defined as the summation of the integrals. It is equivalent with Eq. ((3.35), but with the difference that calculations are performed in the current state, and $\mathbf{B_1}$ does not exist. It should also be noted that, $\boldsymbol{\sigma}$ is defined a bit differently in the second integral on the right side, which is written as:

$$\hat{\sigma} = \begin{bmatrix} \sigma_{11} & \sigma_{12} & 0 & 0 \\ \sigma_{21} & \sigma_{22} & 0 & 0 \\ 0 & 0 & \sigma_{11} & \sigma_{12} \\ 0 & 0 & \sigma_{21} & \sigma_{22} \end{bmatrix}$$

(3.38)

Examining Eq. (3.20), it is seen that, forces are defined in the current state. Since in the context of this dissertation, follower forces (i.e. fluid pressure), are not

considered, the external forces do not change. In addition, the body force lumped on the nodes do not change. Thus, the linearization of loads vanishes.

For Eq. ((3.21), now left is the residual term R^t. It is already mentioned, at equilibrium, that the total of internal variational energy and the external variational energy vanish. However, at an approximate state, one may expect to have some residual, which is the unbalancing of internal energy and the external energy. It can be written in mathematical terms as follows:

$$\begin{aligned} R &= \delta\Pi_{ext} - \delta u^T \cdot \nabla\sigma \\ &= \int_V \rho \cdot \delta u_i \, f_{B_i} \cdot dV + \int_{\Gamma_t} \delta u_i t_i \cdot d\Gamma - \int_V \delta u_i \cdot \partial_j \sigma_{ji} \end{aligned} \tag{3.39}$$

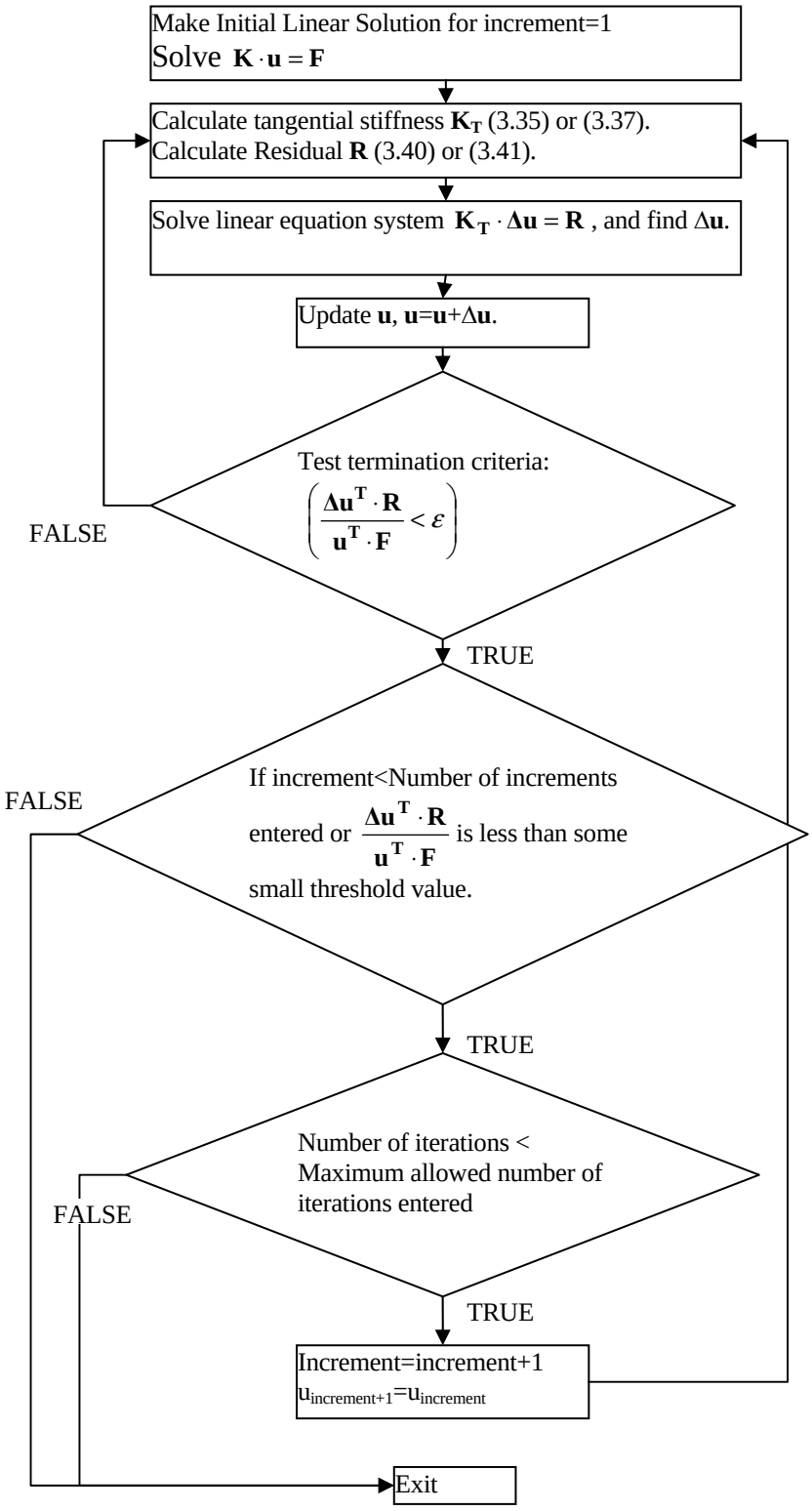
For the TL and UL cases, the above is written respectively as:

$$R = \delta\mathbf{u}^T \cdot \int_{V_0} \rho \cdot \mathbf{f}_{0\mathbf{B}} \cdot dV_0 + \delta\mathbf{u}^T \cdot \int_{\Gamma_{0t}} \mathbf{t} \cdot d\Gamma - \delta\mathbf{u}^T \cdot \int_{V_0} (\mathbf{B}_0 + \mathbf{B}_1)^T \cdot \mathbf{S} \cdot dV_0 \tag{3.40}$$

$$R = \delta\mathbf{u} \cdot \int_V \rho \cdot \mathbf{f}_{\mathbf{B}} \cdot dV + \delta\mathbf{u} \cdot \int_{\Gamma_t} \mathbf{t} \cdot d\Gamma - \delta\mathbf{u} \cdot \int_V \mathbf{B} \cdot \boldsymbol{\sigma} \cdot dV \tag{3.41}$$

In Eqs. (3.39) (3.40), **S** and **σ** are to be written in the vector form as in Eqs. (2.61), (2.62) or (2.63).

It should also be stated here that *Newton's method* is highly instable away from the solution. It may cause oscillations or diverging of residual. To prevent this, the loads and displacements must be applied slowly to force to convergence, that is, incremental solution may be needed. In the program interface, number of increments is entered as a solution parameter. In addition, some measure must be taken for in case oscillations to occur and the program fall into infinite loop, which is not terminating. For that reason, maximum number of iterations is entered for solution parameters of non-linear elastic solution. The formulations may be summarized by the following algorithm:



Algorithm 1: Newton's method for FEM equation system.

The statements and formulas above may seem hard to comprehend. However, they are straightforward for implementation. Having written the general FEM equations briefly, the main topic, contact formulation and the solution techniques, and the binding to the above equations may be derived. In the next chapter, those will be dealt in detail.

CHAPTER 4

CONTACT FORMULATION

4.1 INTRODUCTION

In this chapter, formulation of contact constraints, and the solution methods will be presented. In order to simulate contact with FEM, one needs a mathematical contact model. In the context of FEM, *penalty method*, and the *Lagrange multiplier method* are the two main strategies for dealing with contact. Other methods, in general, are extension of those methods, at least currently. They will be presented in this chapter, as applied to the program written for this dissertation. Before directly relating the subject matter to FEM formulation, some optimization concepts will be included in the chapter. Then, general active set solution strategies, application details to the linear elastic case will be presented. Afterwards, the difficulties arising in the application of the techniques to non-linear elastic case will be discussed, and solution strategies offered in [11], [13], [14], [15] will be briefly explained and the method proposed in [15], which constituting the backbone of this dissertation will be presented in a bit more detail.

4.2 PROBLEM STATEMENT

In general, the minimization problem of the sum of internal energy due to the straining of the body, and the external energy due to the applied external loads on that body is to be solved. Nevertheless, by defining a contact surface, a constraint is defined on that energy equation of the body, such that the energy imposed being very large or infinity when penetration to occur. In effect, that deteriorates the smoothness properties of the energy equation system. In the optimization world, inequality constrained minimization is defined as:

$$\begin{aligned} &\text{minimize} \quad f(\mathbf{x}), && \left\{ f : \Re^n \to \Re \right\} \\ &\text{subject to} \quad h(\mathbf{x}) = 0, && \left\{ h : \Re^n \to \Re^m \right\} \\ &\qquad\qquad g(\mathbf{x}) \geq 0, && \left\{ g : \Re^n \to \Re^s \right\} \end{aligned} \tag{4.1}$$

In the above $\Re$ is the Euclidean Space. In the context of FEM, h(x) are the displacement-defined boundary conditions, which are handled easily, and are called the soft constraints. g(x) are the contact constraints, which are hard to deal with because of derivative of total energy is not being easily obtained at the point of contact; and called the hard constraints. When the constraints are put in the FEM context, the system of equations becomes:

$$\begin{aligned} &\text{minimize } \Pi(\mathbf{x}) = \Pi_{\text{int}}(\mathbf{x}) + \Pi_{\text{ext}}(\mathbf{x}), \\ &\text{subject to:} \\ &h_i(\mathbf{x}) = x_i - c_i = 0, && i \in I_1 \subset I \\ &g(\mathbf{x})_j = (\mathbf{x}_j^2 - \mathbf{x}^1) \cdot \mathbf{n}^1 \geq 0, && j \in I_2 \subset I, \\ &I_1 \cap I_2 = 0, \quad (I_1 \cup I_2) \subset I \end{aligned} \tag{4.2}$$

To state the above equation by words more explicitly, in the discretized world, there is an energy expression to be minimized, with displacement-defined constraints at some nodes in the index set, $i \in I_1 \subset I$, of the system analyzed. There are also the contact constraints in the index set $j \in I_2 \subset I$, but those indices are not coinciding with the displacement-defined indices (i.e. $I_1 \cap I_2 = 0$), in other words, they are disjoint sets. It should also be stated here that, in the framework of this dissertation, only normal contact is dealt, which means, there is no tangential contact force, namely the friction is ignored. That is why the notation $g_N(\mathbf{x})$ is used for normal gap here and after instead of $g(\mathbf{x})$. The body must be supported such that, it cannot undergo rigid body motion when disregarding the contact constraints.

Internal and external energy definitions for FEM are made in Chapter 2 by Eq's (2.86)-(2.89). Soft constraints $h(\mathbf{x})$ are easy to handle. References [2]-[5] may be followed for them. The inequality constraints, $g_N(\mathbf{x})$, will be explained here. Regarding $g(\mathbf{x})$ in Eq. (4.2), $\mathbf{x}^{\alpha}$ are the coordinates in current configuration mapped by $\varphi(\mathbf{X}^{\alpha})$ for body α. In the framework of this dissertation, an elastic body and a rigid

surface is dealt. Thus α=1 will correspond to the rigid surface, while α=2 will correspond to the elastic deformable body. All equations will be presented here for the 2D case, leaving the 3D case for a further work. However, in general, the contact formulations do not change much for 3D case. It is also assumed that the rigid contact surface is stationary (i.e., not moving in any direction). This situation is called *unilateral contact* in literature [1]. By using the fact that displacements will be zero for rigid stationary body, and by use of Eq. (2.2), in a more explicit form, $g_N(\mathbf{x})$ may be written in this context as:

$$
\begin{aligned}
& g_N = G_N - u_N, \qquad \text{where} \\
& G_N = (\mathbf{X}^2 - \mathbf{X}^1(\bar{\xi})) \cdot \mathbf{n}^1(\bar{\xi}) \\
& u_N = \mathbf{u}^2 \cdot \mathbf{n}^1(\bar{\xi})
\end{aligned}
\tag{4.3}
$$

In Eq. (4.3), G_N is defined as the initial normal gap, and g_N being the current normal gap. It should be observed that G_N and g_N are scalar values obtained by dot products of two vectors. It should also be stated here that $\mathbf{X}^1(\bar{\xi})$ is the nearest point, parametrically defined on the rigid surface, obtained with respect to the slave node of the elastic body at the deformed state (Figure 4.1), which will be made clear in the next section.

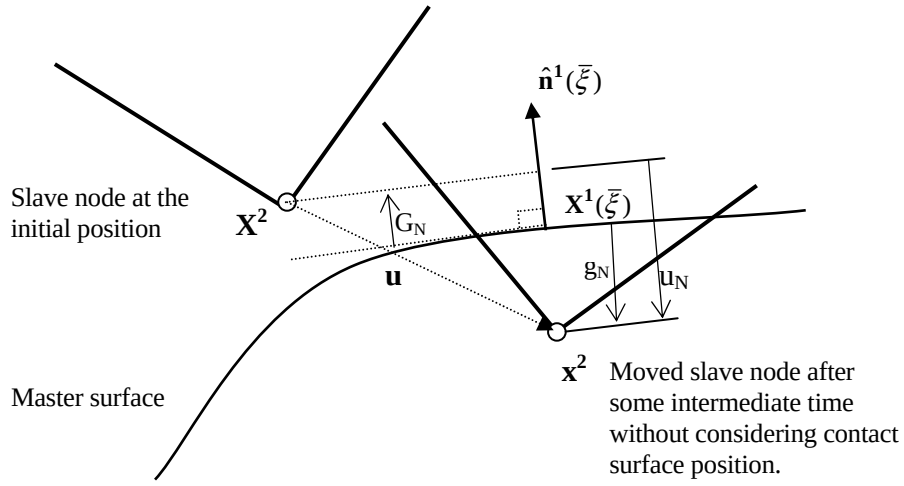


Figure 4.1:Slave node moving on the master contact surface. The figure represents the parameters involved in the gap function.

Now, with the aid of the gap function, the energy term associated with contact force should be formulated. It is plausible to treat the contact force as an external traction force in the form:

$$\Pi_c = \int_{\Gamma_c} \mathbf{u}_c^T \cdot \mathbf{t}_c d\Gamma_c \tag{4.4}$$

However, it should be mentioned in here that, at the equilibrium of the body with the interface, the contact force would be a reaction force. So, equal and opposite forces are associated to the same point. Therefore, contact forces do no work. Nevertheless, if the contact surface had not been there, the body would continue moving and deforming, thus reducing its energy and it can be said that the energy level of the body stays at a higher level due to the existence of contact interface. If a simple 1D case is assumed, a simple spring with a constraint as an example, the situation becomes more comprehensible.

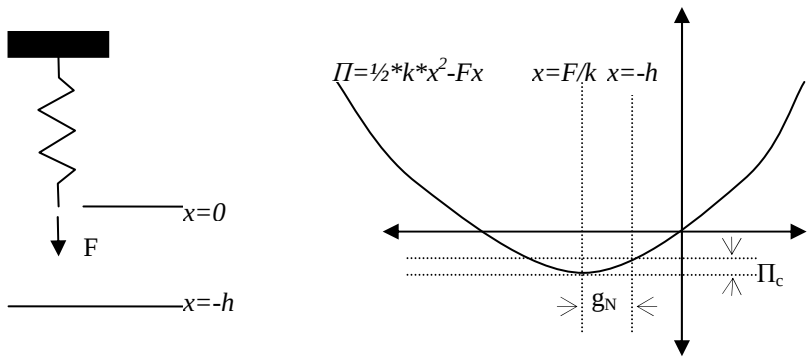


Figure 4.2: A simple 1D contact problem. A spring with an applied load and a contact constraint. The energy function and the effect of the contact interface to the energy system.

The 1D situation considered is simple but instructive for comprehension of the things happening when a contact constraint exists. For the spring-force system, if the force is small, the spring will not see the constraint. However, when the force is large enough, the spring will stop at *–h*, and will not be able to minimize the total energy and reach *x=F/k*. Therefore, some work term is associated with contact.

Though the integral in Eq. (4.4) seems simple in the first sight, the problem there is that, the contact force, and even the contact surface is not known. Eq. (4.4) is only written to state that, the problem is an interaction problem happening only at the interface of two bodies, which is represented by Γ_c, but it is to be determined somehow. In case of multi-body contact, the integral must be calculated for each body and must be added to the total energy equation. The traction vector appearing in Eq. (4.4) can be written in a different form by:

$$\mathbf{t_c} = \boldsymbol{\sigma}^2 \cdot \hat{\mathbf{n}}^{\mathbf{1}} = -p_N \cdot \hat{\mathbf{n}}^{\mathbf{1}} - p_T \cdot \hat{\mathbf{a}}^{\mathbf{1}} \tag{4.5}$$

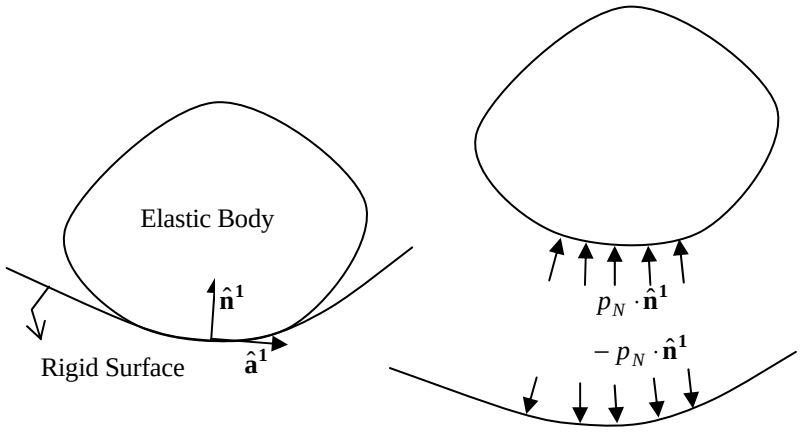


Figure 4.3: Free body diagram for contact interface.

In Eq. (4.5), p_N is the normal pressure, and p_T is the tangential shear, $\hat{\mathbf{n}}^1$ is the unit normal, and $\hat{\mathbf{a}}^1$ is the tangent defined on the master surface. It should be noted here that, p_N and p_T defined as scalars in Eq. (4.5). Tangential traction will not be dealt in the framework of this dissertation, thus p_T is assumed to be zero. Also considering that, living in the discretized world, the integral equation in Eq. (4.4) is transformed to summation for contact nodes as:

$$\Pi_c = \sum_i p_N^i \cdot g_N^i \tag{4.6}$$

In the optimization context, the inequality constraint defined in Eq. (4.2) can be reduced to equilibrium constraint with the following conditions at equilibrium:

$$\begin{aligned} &\text{minimize}\,\Pi_{\text{int}} + \Pi_{\text{ext}}, \quad \text{subject to:} \\ &g_N \geq 0, \\ &p_N \leq 0, \\ &p_N \cdot g_N = 0, \qquad (\textit{Complementary Slackness}) \end{aligned} \tag{4.7}$$

The Eq. (4.7) are called as *Hertz-Signorini-Moreau* conditions for frictionless contact [1]. In the context of optimization, they are called the *Karush-Kuhn-Tucker Conditions*. It can be said that, when $g_N \geq 0$, no contact pressure is expected. When there is contact force, g_N is zero. The first and second inequalities in Eq.(4.7) are

called *necessary optimality conditions* for inequality constraints in optimization context [8]. In the optimization context, the term p_N used here is the *Lagrange multiplier*.

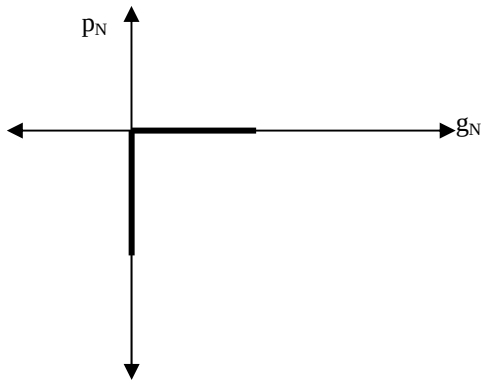


Figure 4.4: The function of p_N with respect to g_N Note the sharp change in the graph at g_N=0, which creates the major problem in optimization.

It can be noted for the complementary slackness term that the dimension of p_N is force, while the dimension of g_N is distance. Then it can be said that, the complementary slackness is associated to energy, which means that zero energy is associated with contact forces at equilibrium when g_N is zero, that is, equal and opposite forces do no work at equilibrium. However, in case of penetration, p_N and g_N being less than zero, a positive energy term is associated with contact. Else, if g_N is larger than zero, which is the gap open state, then p_N being zero. Therefore, zero energy is associated with gap open state. Complementary slackness condition in Eq. (4.7) includes both of these situations. In the following sections, the above energy form will be tackled.

4.3 RIGID SURFACE DEFINITION

In the framework of this dissertation, the rigid surface needs to be defined mathematically. As the solution technique, *master surface-slave node* technique is

used, which is the most widely used and accepted technique in the numerical treatment of contact. The definition of *Bezier surfaces* is used as in [1] and [9]. In effect, this section is based on [9]. The methods proposed results in smooth third order polynomial definition of contact surface, satisfying C^1 continuity. In [9], two types of Bezier curves have been presented as *cubic Hermite* and *cubic Bernstein* interpolations, both of which have been applied in the program developed for this dissertation. The program implementation is explained in the next chapter. They stand as an option of visualization and analysis in the program interface.

The discretized surface model is local, that is, a change in the position of a node only affects the curves corresponding to that node. Every node is associated to a single surface.

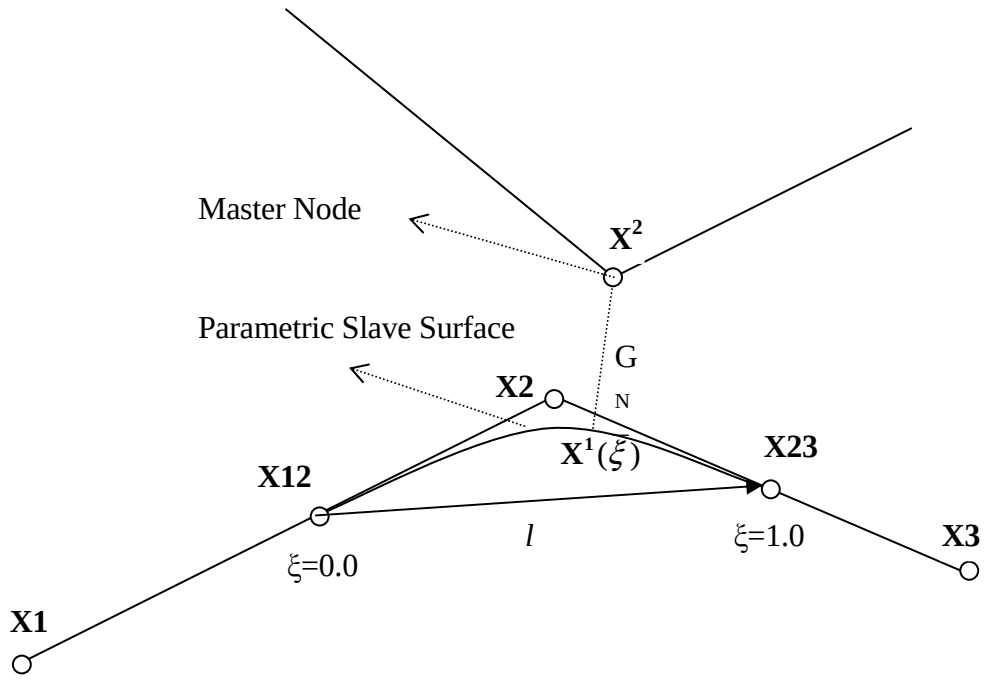


Figure 4.5: Representation of *Hermit interpolation surface* definition.

The *cubic Hermit interpolation* surface is formulated as a curve tangent to two lines drawn connecting three nodes at their mid points (Figure 4.5). The conditions of the curve may be stated as:

$$\mathbf{X}(\xi)\big|_{\xi=0} = \mathbf{X12},$$
$$\mathbf{X}(\xi)\big|_{\xi=1} = \mathbf{X23},$$
$$\left.\frac{d\mathbf{X}}{d\xi}\right|_{\xi=0} = \frac{\mathbf{X2}-\mathbf{X1}}{\|\mathbf{X2}-\mathbf{X1}\|},$$
$$\left.\frac{dX}{d\xi}\right|_{\xi=1} = \frac{\mathbf{X3}-\mathbf{X2}}{\|\mathbf{X3}-\mathbf{X2}\|}$$

(4.8)

Then the *Hermit interpolation* is given by:

$$\mathbf{X^1}(\xi) = \mathbf{X_1} + \xi \cdot (\mathbf{X_2} - \mathbf{X_1}) + w(\xi) \cdot \hat{\mathbf{e}}_\mathbf{N}$$

(4.9)

where, **X1, X2, X3** are the position vectors of points for representing the contact surface, and $\hat{\mathbf{e}}_\mathbf{N}$ is a unit vector defined normal to the line connecting **X12** and **X23** and w(ξ) is a third order polynomial defined as:

$$w(\xi) = A \cdot l^3 \cdot \xi^3 + B \cdot l^2 \cdot \xi^2 + C \cdot l \cdot \xi + D, \text{ where}$$
$$l = \|\mathbf{X23}-\mathbf{X12}\|$$

(4.10)

where, A, B, C and D are the constants to be determined for the conditions in Eq. (4.8), and *l* is the length of line connecting midpoints of two lines, **X12** and **X23**.

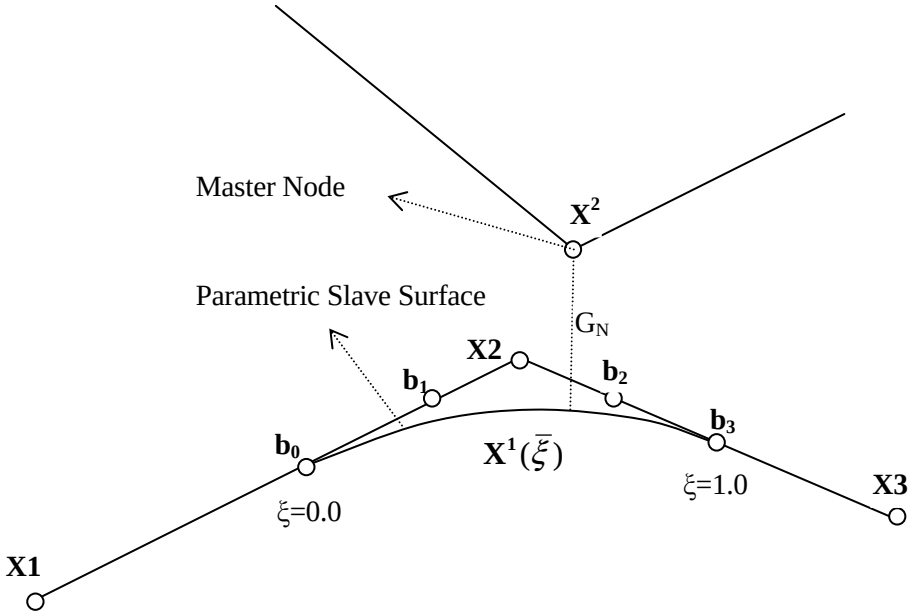


Figure 4.6: Representation of *Bernstein Interpolation Surface* definition.

In case of *Cubic Bernstein Interpolation*, two more points are needed. There are four intermediate points to represent the curve, $\mathbf{b_0}$, $\mathbf{b_1}$, $\mathbf{b_2}$, $\mathbf{b_3}$ (Figure 4.6). The function is defined as:

$$\mathbf{X^1}(\xi) = \mathbf{b_0} \cdot B_0(\xi) + \mathbf{b_1} \cdot B_1(\xi) + \mathbf{b_2} \cdot B_2(\xi) + \mathbf{b_3} \cdot B_3(\xi) \tag{4.11}$$

With the interpolating polynomials:

$$\begin{aligned} B_0(\xi) &= (1-\xi)^3, \\ B_1(\xi) &= 3 \cdot \xi \cdot (1-\xi)^2, \\ B_2(\xi) &= 3 \cdot \xi^2 \cdot (1-\xi), \\ B_3(\xi) &= \xi^3 \end{aligned} \tag{4.12}$$

Both of the formulations (*Hermite* and *Bernstein*) have been applied in the program developed as stated before. The surface type is optionally selected from the interface. For this purpose, the *inheritance* property of OOP has been utilized, which will be made clear in the next chapter. The implementation of the second form seemed

simpler and more efficient because a less number of parameters is dealt. Moreover, invariance under rotation of frame is stated as an advantage in [9].

For the contact formulation, a nearest contact position needs to be determined, which is achieved by Newton-Raphson iteration for ξ, with quadratic convergence within 4 or 5 iterations, however maximum number of iterations being fixed as 10. There are also other rigid surface definitions in literature, as NURBS curves [10]. The application of other types of curves is left for a further work. However, once more the general properties of a convenient curve definition for contact problems can be stated as:

1. Local with respect to a node on the surface, that is, moving of a node for the definition of the rigid surface should not affect the entire curve, or should result in change of the surface within a bounded region around the point that was moved.
2. Smooth, that is, easily differentiable on the entire curve; no sharp changes should occur for the normal. Although at least C^0 continuity may be enough for frictionless contact, at least C^1 continuity is required for friction solutions. Though friction is not considered for this work, for a further work, it may be implemented.

The above requirements are best discussed in [10], with the discussion of NURBS curve with C^n continuity; which have been left for a further development issue as stated previously.

4.4 VARIATIONAL FORMULATION OF CONTACT

In this section, the variational form of contact constraints should be dealt since the variational weak form is used in the FEM context. The contact constraint defined as the complementary slackness in Eq. (4.7) should be represented in the variational form, to be implemented in the general context of FEM. As stated previously in the second chapter, *variation* is the small perturbation at the equilibrium state. In case of contact, both the contact force and the penetration may be perturbed. So the variational form may simply be written as:

$$\delta\Pi_c = \sum_{i \in I} (\delta p_N^i \cdot g_N^i + p_N^i \cdot \delta g_N^i) \geq 0$$

(4.13)

However, since one is still dealing with the minimization problem, the total energy variation is defined as:

$$\delta\Pi = \delta\Pi_{\text{int}} + \delta\Pi_{ext} + \delta\Pi_c = 0$$

(4.14)

In Eq. (4.14), the first two terms are dealt previously in Chapter 2, and the third term is defined in Eq. (4.13). It is really an interesting result to be noted. At equilibrium, the total variation must be zero, though there are inequality constraints.

In Eq. (4.13), the variation of g_N is needed, which may be written as:

$$\delta g_N = (\delta \mathbf{x}^2 - \delta \mathbf{x}^1(\bar{\xi})) \cdot \hat{\mathbf{n}}^1(\bar{\xi}) + (\mathbf{x}^2 - \mathbf{x}^1(\bar{\xi})) \cdot \delta \hat{\mathbf{n}}^1$$

(4.15)

Using the fact that $(\mathbf{x}^2 - \mathbf{x}^1(\xi))$ being in the direction of $\hat{\mathbf{n}}^1$, and $\delta\hat{\mathbf{n}}^1$ being normal to $\hat{\mathbf{n}}^1$, the second term on the right vanish. In addition, since the master surface is not moving, Eq. (4.15) reduces, and it can be written as:

$$\delta g_N = \delta \mathbf{u}^2 \cdot \hat{\mathbf{n}}^1$$

(4.16)

Now, it is time to explain the methods of implementation of contact constraints to general FEM equations, which will be done in the next section.

4.5 METHODS OF SOLUTION

In this section, the methods of the solution to (4.14) will be discussed. Due to the inequality constraint, there is the differentiability problem being faced for the Π_c term defined. That needs special handling techniques to be discussed. For the linear elastic case, the differentiability problem is in general solved with active set strategies. While for the nonlinear elastic case, this method is not being so convenient, due to the linearization issues of Newton type algorithms.

4.5.1 Penalty Method

This method is the simplest to formulate and implement, and this is the oldest method for those types of problems. In the penalty method, it is assumed that contact force p_N is proportional to gap g_N. However, it is defined so that, a very large force is associated to penetration. In mathematical terms:

$$\Pi_c = \frac{1}{2}\kappa \cdot g_N^2 \tag{4.17}$$

where κ is a very large number. It should be noted in here that the contact energy function in Eq. (4.17) is very similar to energy function of a spring. The effect of Eq. (4.17) can be conceived as, a very stiff spring being active in case of penetration, such that the penetration is virtually prevented. It is not possible to say totally prevented, since this method always results in some amount of penetration. However, as κ approaches to infinity, it is expected to have zero penetration. Nevertheless, giving the value infinity to κ is not possible for the numerical reasons. It creates ill conditioning problems if too large a value is entered. For the implementation issues, Eq. (4.17) must be written in the variational form:

$$\delta\Pi_c = \kappa \cdot g_N \cdot \delta g_N \tag{4.18}$$

Writing g_N and δg_N in terms of u:

$$\begin{aligned}\delta\Pi_c &= \kappa \cdot (G_N + u_N) \cdot \delta u_N \\ &= \kappa \cdot \underbrace{(\mathbf{X^2} - \mathbf{X^1}) \cdot \hat{\mathbf{n}}^1}_{G_N} \cdot \underbrace{(\boldsymbol{\delta}\mathbf{u^2} - \boldsymbol{\delta}\mathbf{u^1}) \cdot \hat{\mathbf{n}}^1}_{\delta u_N} + \kappa \cdot \underbrace{(\mathbf{u^2} - \mathbf{u^1}) \cdot \hat{\mathbf{n}}^1}_{u_N} \cdot \underbrace{(\boldsymbol{\delta}\mathbf{u^2} - \boldsymbol{\delta}\mathbf{u^1}) \cdot \hat{\mathbf{n}}^1}_{\delta u_N}\end{aligned} \tag{4.19}$$

Since in case of unilateral contact with the master surface being stationary, Eq. (4.19) further reduces to:

$$\delta\Pi_c = \kappa \cdot (\mathbf{X^2} - \mathbf{X^1}) \cdot \hat{\mathbf{n}}^1 \cdot (\boldsymbol{\delta}\mathbf{u^2} \cdot \hat{\mathbf{n}}^1) + \kappa \cdot (\mathbf{u^2} \cdot \hat{\mathbf{n}}^1) \cdot (\boldsymbol{\delta}\mathbf{u^2} \cdot \hat{\mathbf{n}}^1) \tag{4.20}$$

Eq. (4.20) cannot be linearized due to the differentiability reasons, since it is valid only for active nodes. There is nothing partially being in contact. There is for a node

penetrating or not penetrating, which is like 0 or 1 and changing quickly and sharp in the course of the solution. Therefore, the active set strategy is only valid for linear elastic case, which is valid for small stress, small strain, and small deformation case in terms of FEM formulation. However, the penalty method still has practical applications, and further methods have been developed based on this method with some improvement, which will be dealt in the sequel. Now, the implementation of Eq. (4.20) will be explained for this section.

The general linear elastic FEM energy variational form can be written without considering contact, as:

$$\delta \mathbf{u}^T \cdot \mathbf{K} \cdot \mathbf{u} = \delta \mathbf{u}^{\mathbf{T}} \cdot \mathbf{F} \tag{4.21}$$

For the Eq. (4.21), looking at Eq. (3.19) and Eq. (3.20), the terms **K** and **F** can be deduced. Considering Eq. (4.14), and Eq. (4.20) being applied to i^{th} node with the corresponding degrees of freedoms *m* and *n*, it is obvious that Eq. (4.20) modifies Eq. (4.21) as:

$$\overline{K} = \begin{bmatrix} K_{11} & K_{12} & K_{13} & \cdots & K_{1m} & \cdots & K_{1n} & \cdots & K_{1N} \\ K_{21} & K_{22} & K_{23} & \cdots & K_{2m} & \cdots & K_{2n} & \cdots & K_{2N} \\ K_{31} & K_{32} & K_{33} & \cdots & K_{3m} & \cdots & K_{3n} & \cdots & K_{3N} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ K_{m1} & K_{m2} & K_{m3} & \cdots & K_{mm} + \kappa \cdot \hat{n}_1^1 \cdot \hat{n}_1^1 & \cdots & K_{mn} + \kappa \cdot \hat{n}_1^1 \cdot \hat{n}_2^1 & \cdots & K_{mN} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ K_{n1} & K_{n2} & K_{n3} & \cdots & K_{nm} + \kappa \cdot \hat{n}_2^1 \cdot \hat{n}_1^1 & \cdots & K_{nn} + \kappa \cdot \hat{n}_2^1 \cdot \hat{n}_2^1 & \cdots & K_{nN} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ K_{N1} & K_{N2} & K_{N3} & \cdots & \cdots & \cdots & \cdots & \cdots & K_{NN} \end{bmatrix} \tag{4.22}$$

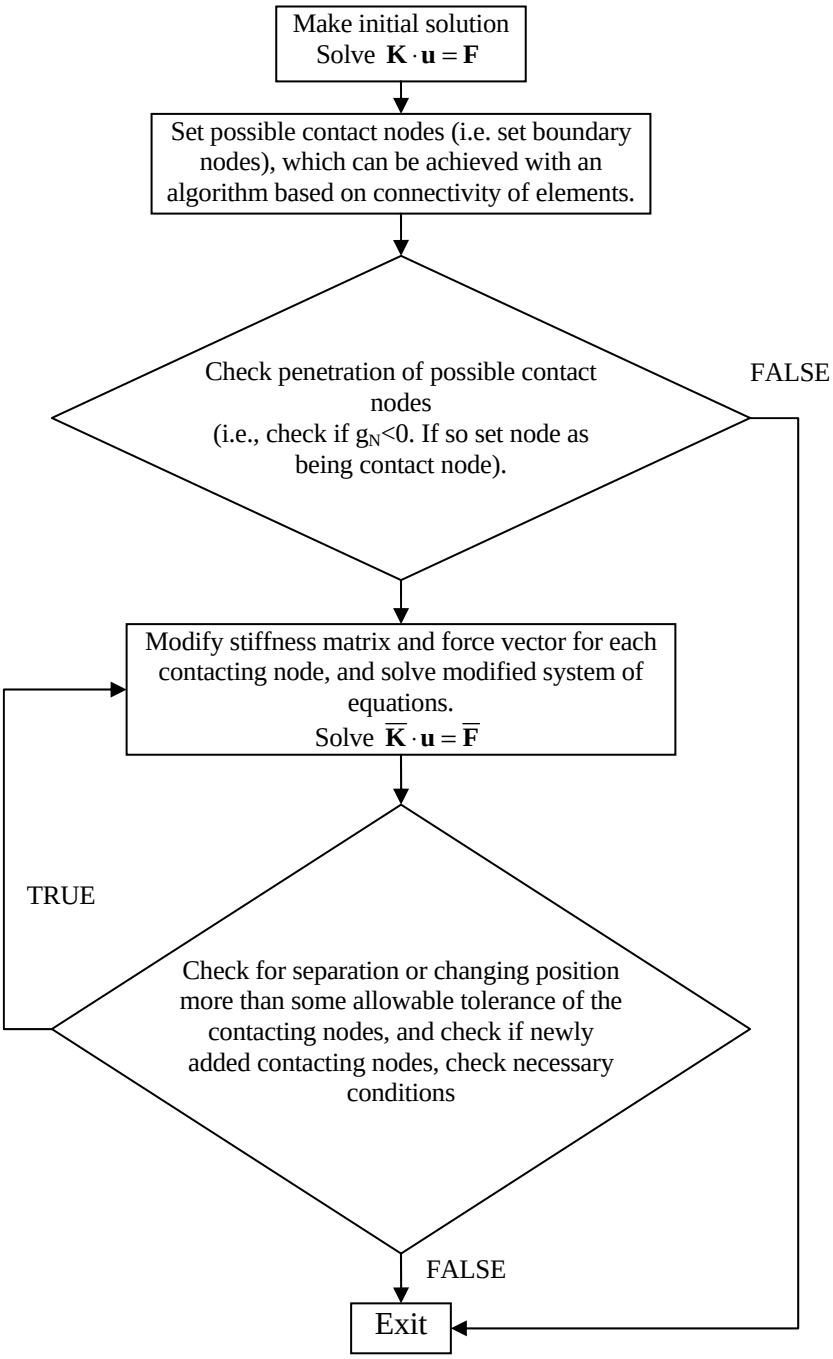
and

$$\overline{F} = \begin{bmatrix} F_1 \\ F_2 \\ F_3 \\ \vdots \\ F_m - \kappa \cdot G_N \cdot \hat{n}_1^1 \\ \vdots \\ F_n - \kappa \cdot G_N \cdot \hat{n}_2^1 \\ \vdots \\ F_N \end{bmatrix} \tag{4.23}$$

As it is obvious from the modifications in the stiffness and the force terms, the first part on the right hand side of Eq. (4.20) modifies the force vector, while the second part modifies the stiffness matrix. The modification is applied for every contacting node. Nevertheless, initially one cannot know which nodes are contacting. Therefore, it is not known prior to the solution which indices to modify. There should be a way of guessing which nodes are contacting, and that makes the linear simple problem complicated and non-linear. The general algorithm may be summarized as in Algorithm 2.

This method in general permits some small amount of penetration. As the penalty term κ increases, the amount of penetration decreases. However, the penalty term cannot be increased without bound. Too high a penalty term results in ill-conditioning of system of equations. A reasonable choice would be the largest number at the diagonal of the stiffness matrix. In the program interface, it is permitted to enter different values for being able to make tests. The entered value from the program interface is multiplied by the largest number at the diagonal of the stiffness matrix.

Advantages of this method can be summarized as being simple to apply, and easy to implement. Disadvantages can be listed as ill conditioning, and giving approximate results in the infeasible region.



Algorithm 2: Pseudo algorithm for contact solution with the *Penalty method.*

4.5.2 Lagrange Multiplier Method

This method is applicable to linear elastic problems with small stress, small strain, and small deformation problems like the previous penalty method. For non-linear elastic solutions, it is prohibitively difficult to apply. It has advantages and disadvantages compared to the penalty method. A Lagrange function defined as:

$$L(u,\lambda) = f(x) + \lambda \cdot g(x),\ (\mathrm{x},\lambda) \in \Re^{\mathrm{n}} \times R^{m} \tag{4.24}$$

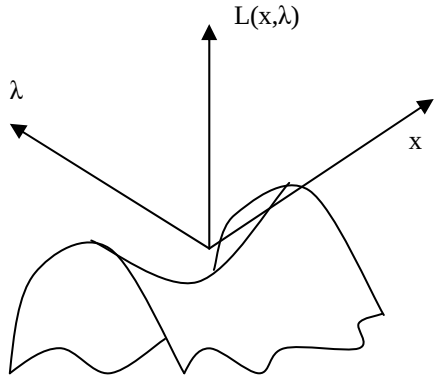


Figure 4.7: Lagrange Function L(x,λ).

where $\Re^n$ is the Euclidean space, $f:\Re^n \to \Re$, $x \in X \subset \Re^n$, $\lambda \in \Re^m : \lambda \le 0$, and $g:\Re^n \to \Re_+$, $\Re_+$ being the positive octant, while λ is defined in the negative octant.

The primal problem is defined as [8]:

$$L^*(x) = \max_{\lambda \le 0} L(x,\lambda) = \begin{cases} f(x), \text{if } \mathrm{g(x)} \ge 0 \\ \infty, \quad \text{otherwise} \end{cases} \tag{4.25}$$

For the Eq. (4.24), *L(x)* will be maximized when λ is zero. If g(x)<0 the *Lagrange function L* will increase without limit. Then the min-max problem is:

$$\underset{x\in X}{\text{minimize}}\, L^{*}(x,\lambda) = \underset{x\in X}{\text{minimize}}\, f(x)$$

(4.26)

The dual function is defined as:

$$L_{*}(\lambda) = \min_{x\in X} L(x,\lambda)$$

(4.27)

and the dual problem is defined as:

$$\underset{\lambda\le 0}{\text{maximize}} \quad \min_{x\in X} L(x,\lambda)$$

(4.28)

The uniqueness of the optimum (x,λ) is proved by the duality theorems for convex functions [8]. The uniqueness of the solution requires the optimal point being the saddle point of the Lagrangian function defined in Eq. (4.24). As it may be obvious, λ is a new unknown parameter to be determined for the minimization problem. The system of equations to be solved takes the form:

$$\begin{bmatrix} \nabla_{xx} f(x) & \nabla_{x} g(x) \\ \nabla_{x} g(x) & 0 \end{bmatrix} \begin{bmatrix} x \\ \lambda \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

(4.29)

The preparations are complete for application of the technique. Now turning to the solution of the contact problem, one needs to solve Eq. (4.14). In that case, $\delta\Pi_c$ can be written as:

$$\delta\Pi_c = \int_{\Gamma_c} \delta\lambda \cdot g_N(x) \cdot d\Gamma_c + \int_{\Gamma_c} \lambda \cdot \delta g_N(x) \cdot d\Gamma_c$$

(4.30)

Since in the discretized world one is to impose constraints onto the nodes, Eq. (4.30) can be written in a more explicit form for nodes as:

$$\delta\Pi_c = \sum_{i\in I} \delta\lambda^{i} \cdot (G_N^{i} + u_N^{i}) + \sum_{i\in I} \lambda^{i} \cdot \delta u_N^{i}$$

(4.31)

Eq. (4.31) can be imposed on the general FEM equations for the i^{th} node with corresponding degrees of freedom being *m* and *n* as:

$$\overline{K} = \begin{bmatrix} K_{11} & K_{12} & \cdots & K_{1m} & \cdots & K_{1n} & \cdots & K_{NN} & 0 \\ K_{21} & K_{22} & \cdots & K_{2m} & \cdots & K_{2n} & \cdots & K_{2N} & 0 \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots & 0 \\ K_{m1} & K_{m2} & \cdots & K_{mm} & \cdots & K_{mn} & \cdots & K_{mN} & \hat{n}_1^1 \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots & 0 \\ K_{n1} & K_{n2} & \cdots & K_{nm} & \cdots & K_{nn} & \cdots & K_{nN} & \hat{n}_2^1 \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots & 0 \\ K_{N1} & K_{N2} & \cdots & K_{Nm} & \cdots & K_{Nn} & \cdots & K_{NN} & 0 \\ 0 & 0 & 0 & \hat{n}_1^1 & 0 & \hat{n}_2^1 & 0 & 0 & 0 \end{bmatrix} \tag{4.32}$$

and,

$$\overline{F} = \begin{bmatrix} F_1 \\ F_2 \\ \vdots \\ F_m \\ \vdots \\ F_n \\ \vdots \\ F_N \\ -G_N \end{bmatrix} \tag{4.33}$$

The above modification in stiffness matrix and the force vector must be performed for each contacting node. Therefore, the size of the stiffness matrix and the force vector increase by one for each contacting node. For speed considerations, in the program implementation, the global stiffness matrix is stored, and whenever an update is necessary, it is called and copied to new larger matrix and modified stiffness matrix is created. That doubles the storage needs. Or, else, every time the global stiffness matrix had to be regenerated. The advantage of this method may be that it is an exact solution, that is g_N=0 is exactly satisfied. The disadvantage may be the need for a larger storage, every time changing size of the stiffness matrix becomes expensive. As stated before, this method is applicable to linear elastic case. For non-linear elasticity problems, the use of this method is prohibitive. The algorithm can be summarized as in Algorithm 2, except the method of calculating the modified stiffness matrix. The algorithm also includes some index tracking for the *Lagrange multiplier*.

4.5.3 Augmented Lagrange Multiplier Method

This method combines the *penalty method* and the *Lagrange multiplier method* to utilize advantages of both. It is nearly exact, but criticized for slow in convergence, being strongly dependent on penalty term, and offering not much advantage compared to *Lagrange Multiplier method* for the linear elastic case [11][15]. However, this method is applicable for non-linear elastic solutions. It is also applicable to non-linear frictional solutions [11], [12]. Actually, this section will be based on [11]. This method has not been applied in the program developed since a better method is proposed by the same authors in [15]. This method will briefly be introduced for completeness.

In this method, an initial penalty solution is performed. The resultant contact forces are applied as external forces and the system is solved again for contact with the penalty method. The procedure continues until the gap reduces to a reasonable value. In mathematical terms, the method can be written as:

$$\Pi_c = \sum_{i \in I} \frac{1}{2} \kappa \cdot g_N^{i\,2} + \lambda^i \cdot g_N^i \tag{4.34}$$

Looking at the above equation carefully, it is obvious that if λ=0, penalty formulation is recovered. The stiffness matrix and the force matrices are modified in the same way as in the penalty method, but only one more modification to force matrix is performed due to the *Lagrange Multiplier* term. One begins with λ=0 and augments it continuously. As reaching to convergence, the effect of penalty term approaches to zero.

Eq. (4.34) should be written in variational form for implementing it in general FEM equations:

$$\delta\Pi_c = \kappa \cdot g_N \cdot \delta g_N + \lambda \cdot \delta g_N \tag{4.35}$$

Note that in Eq. (4.35), there is no variation of λ, since it is determined previously and not vary for an increment. It comes from the previous solution.

For implementing this method in Newton type algorithms, one also needs the linearization of Eq. (4.35), which can be written as:

$$D(\delta\Pi_c) = \kappa(Dg_N \cdot \delta g_N + g_N D\delta g_N) + D\lambda \cdot \delta g_N + \lambda \cdot D\delta g_N \tag{4.36}$$

Eq. (4.36) is written in a more explicit form in [11] as:

$$\begin{aligned}&(\nabla^2\Pi)^T \cdot \Delta x + c\sum_{i\in I}\left(g_N^i(\nabla^2 g_N^i)^T\Delta x + c\cdot\nabla g_N^i\cdot(\nabla g_N^i)\cdot\Delta x\right)+\\&\sum_{i\in I}\left(\lambda_i\cdot(\nabla^2 g_N^i)^T\cdot\Delta x + \nabla g_N^i\Delta\lambda_i\right) = -\left(\nabla\Pi + c\cdot\sum_{i\in I}\left(\nabla g_N^i\cdot g_N^i + \nabla g_N^i\cdot\lambda^i\right)\right)\end{aligned} \tag{4.37}$$

where $\nabla^2\Pi$ is the *Hessian matrix*. This is previously called the tangential stiffness matrix and defined as $\mathbf{K_T}$ in Chapter 2, of the general FEM equations, and $\nabla\Pi$ is the gradient of the general FEM equations being the residual term, which also defined in Chapter 2. The above may be written in a more compact form as [11]:

$$\nabla^2 L\cdot\Delta x + \nabla g\cdot\Delta\lambda = -\nabla L \tag{4.38}$$

There is one more equation to be included in the system, which is g_N being equal to zero, from the linearization of which, one can get [11]:

$$\nabla g_N^i\cdot\Delta x = -g_N^i \tag{4.39}$$

Combining them, the equation needed to solve becomes [11]:

$$\begin{bmatrix}\nabla^2 L(x,\lambda) & \nabla g(x)\\ \nabla g(x) & 0\end{bmatrix}\begin{bmatrix}\Delta x\\ \Delta\lambda\end{bmatrix} = \begin{bmatrix}-\nabla g(x)\\ -g(x)\end{bmatrix} \tag{4.40}$$

Eq. (4.40) seems equivalent to the standard *Lagrange Multiplier Method*, but not completely. Since this method has not been applied, most of the details are skipped. The interested reader should consult to [11] or [12] for the details. Actually, in [11], a method for speeding the algorithm has been proposed, but though it needs much user intervention, and does not seem much preferable compared to other methods. Here, it

should be mentioned that this method is applied in most of the commercial FEM programs handling contact, like ABAQUS, ANSYS, and MARC.

4.5.4 Barrier Method

This method is designed in the contact mechanics for non-linear elasticity problems. However, it can also be applicable to linear problems in this framework. In this method, instead of setting nodes being active or inactive, and changing activity of constraint continuously, all the boundary nodes are set as being active, and that does not change in the course of the solution. This is a penalty kind of method, but the difference is that the penalty term κ is a function of the gap. This section is based on [13].

It should be mentioned here that, this method is not being applied to the program developed for this dissertation either. In [9] it is criticized and a better approach is suggested. it is mentioned here for the completeness of the subject matter.

Turning again to the contact problem, the energy term may be defined this time as:

$$\Pi_c = \mu \sum_{i \in I} \lambda^i \ln\left(1 + \frac{d^i(x)}{\mu}\right) \tag{4.41}$$

In Eq. (4.41), $d(x)$ is written instead of g_N since it is defined also in the positive side, which means distance instead of penetration. Also in the same equation, $\mu>0$ but small, is the barrier parameter to be entered to the program. In addition, $\lambda^i<0$ are the fixed estimates of the *Lagrange multipliers*. It should be noted that, as $d^i(x)$ approaches $-\mu$ and considering $\lambda \le 0$, Π_c approaches infinity. This causes ill conditioning problems. For that reason after some small amount of penetration, a parabola is fit to the barrier function in Eq. (4.41). To speed up the convergence, also a scaling method is proposed [13].

The smooth function in Eq. (4.41) can easily be linearized and implemented into the general non-linear tangential stiffness. An initial linear elastic solution with the standard penalty method can be performed. This way λ^i's may be initiated. Contact tangential stiffness is to be imposed into general tangential stiffness. Newton

iterations can be performed until convergence, and λ^i's may be reset and the Newton iterations may be repeated. Continue until λ^i's change by some allowable limit.

This method is also an approximate one. It is criticized for always being in the feasible region, which means preventing penetration totally and causing some small gap. It also needs one more parameter μ to be entered to the program. Details of the method are not elaborated since it is not implemented in this dissertation. The algorithm seemed complicated for implementing.

4.5.5 Constraint Function Method

This section is based on [14]. The method is offered for non-linear elastic problems. In this method, since at g_N=0 there is the differentiability problems, a smooth function approximately satisfying the complementary slackness Eq. (4.7) is offered (Figure 4.8). Again all possible contact nodes are set active and this constraint function is applied to all of them.

The constraint function is defined as [2], [14]:

$$w(g,\lambda)=\frac{g_N-\lambda}{2}-\sqrt{\left(\frac{g_N+\lambda}{2}\right)^2+\varepsilon} \tag{4.42}$$

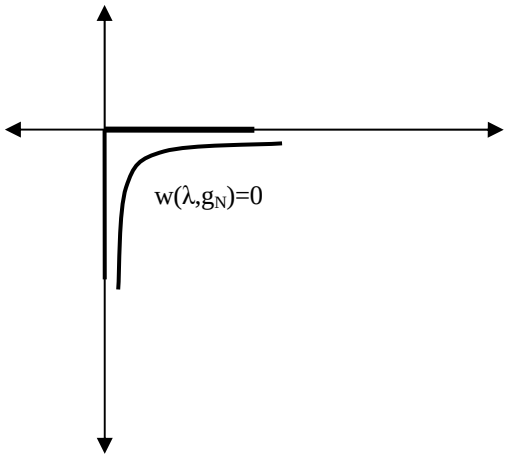


Figure 4.8: w(λ,g_N) approximately satisfying *complementary slackness*.

where, $g_N \geq 0$ is the gap, $\lambda \leq 0$ is the *Lagrange multiplier* term, being the contact force, and ε>0 is a small number. The constraint *w(g,λ)*=0 is imposed on the general FEM equations. For this purpose, of course Eq. (4.42) must be linearized first. It should be noted here that unknown extra parameters to be determined, λ, enter into the equations. Considering all possible contact nodes being active, and iterative Newton solution being applied, this method seems costly and is not preferred. It is also mentioned and criticized in [9]. References [2] and [14] offer a function also for friction solutions with the method.

The details are not elaborated because this method is mentioned for the completeness of the subject, and is not applied in the program developed for this dissertation.

4.5.6 Cross Constraint Method

This method is applied nicely in the program developed. It has super-linear convergence rate and nicely adaptable to *Newton algorithm*. This section is based on [15], in which the *barrier method* and the *constraint function method* have also been criticized, and this method is submitted as a new and a better approach. However, nothing is mentioned about friction. Nevertheless, it is very convenient for the purposes of this dissertation and fits nicely into the context, unilateral frictionless contact.

The approach in [15] is similar to the *barrier method* or the *constraint function method* in the sense that no distinction is made for the gap status (i.e. gap open or gap closed), all boundary nodes being active and a continuous function with respect to gap is defined. However, different from the *constraint function method*, stiffness matrix size is not increased, and different from the *barrier method*, the function in here is defined in both feasible and infeasible regions (i.e. giving possibility of gap being open or closed for a contacting node).

Now coming to the details of the method, the contact energy is defined as an exponential function in the form:

$$\Pi_c = \int_{\Gamma_c} \left(\mu e^{a \cdot g_N} + \beta e^{b \cdot g_N} \right) d\Gamma_c$$

(4.43)

where μ, β, a, b are constants to be determined from conditions of contact surface and contact force. In addition, contact force and the contact stiffness are defined as:

$$F = \nabla \Pi_c,$$
$$K = \nabla(\nabla \Pi_c)$$

(4.44a,b)

With the conditions $\lim_{g_N \to \infty} \Pi_c = 0$, $\lim_{g_N \to 0} F = \hat{F}$ and $\lim_{g_N \to 0} K = \hat{K}$, constant parameters are determined with, $\hat{F} < 0$ being some approximation to contact force, and $\hat{K} > 0$ being an approximation to contact stiffness. It should also be mentioned here that F is the gradient of the potential, and K is the Hessian of the same potential with respect to g_N and Eq. ((4.43) is defined on both sides of the constraint surface (i.e. when g_N is positive or negative).

Then after some manipulations, the potential, force, and the contact stiffness becomes for node i as:

$$\Pi_c^i = \frac{\hat{F}^{i2}}{\hat{K}} \cdot e^{\frac{\hat{K}}{\hat{F}^i} \cdot g_N^i},$$
$$\mathrm{F^i} = \hat{\mathrm{F}}^{\mathrm{i}} \cdot \mathrm{e}^{\frac{\hat{\mathrm{K}}}{\hat{\mathrm{F}}^{\mathrm{i}}} \cdot g_N^i},$$
$$\mathrm{K^i} = \hat{\mathrm{K}} \cdot \mathrm{e}^{\frac{\hat{K}}{\hat{F}^i} \cdot g_N^i}$$

(4.45a,b,c)

Looking at Eqs. (4.45a,b,c), as g_N becomes negative, that is in case of penetration, the contact stiffness, which is the penalty term becomes large and may cause ill-conditioning of the general matrix equation. However, as g_N becomes positive, the contact stiffness gets smaller. In the penetration case, to prevent ill conditioning due to K being large, a parabola is fitted smoothly to the contact potential at $g_N=0$. The derivative of the parabola with respect to g_N being the force term, and the second derivative being the contact stiffness term, which can be written for contacting node i as:

$$\Pi_c^i = \frac{\hat{F}^{i\,2}}{\hat{K}} + \hat{F}^i \cdot g_N^i + \frac{\hat{K}}{2} g_N^{i\,2},$$
$$\mathrm{F}^{\mathrm{i}} = \hat{\mathrm{F}}^{\mathrm{i}} + \hat{\mathrm{K}} \cdot \mathrm{g}_{\mathrm{N}}^{\mathrm{i}},$$
$$\mathrm{K}^{\mathrm{i}} = \hat{\mathrm{K}}$$

(4.46a,b,c)

Note that in Eq. (4.46a,b,c), the standard penalty is recovered in case of penetration. For application to general FEM equations, Π_c must be written in variational form:

$$\delta\Pi_c^i = \begin{cases} \hat{F}^i \cdot e^{\frac{\hat{F}^i}{\hat{K}} g_N^i} \cdot \delta g_N^i & \text{if} \quad \mathrm{g}_{\mathrm{N}}^{\mathrm{i}} \geq 0 \\ (\hat{F}^i + \hat{K} \cdot g_N^i) \cdot \delta g_N^i & \text{if} \quad \mathrm{g}_{\mathrm{N}}^{\mathrm{i}} < 0 \end{cases}$$

(4.47)

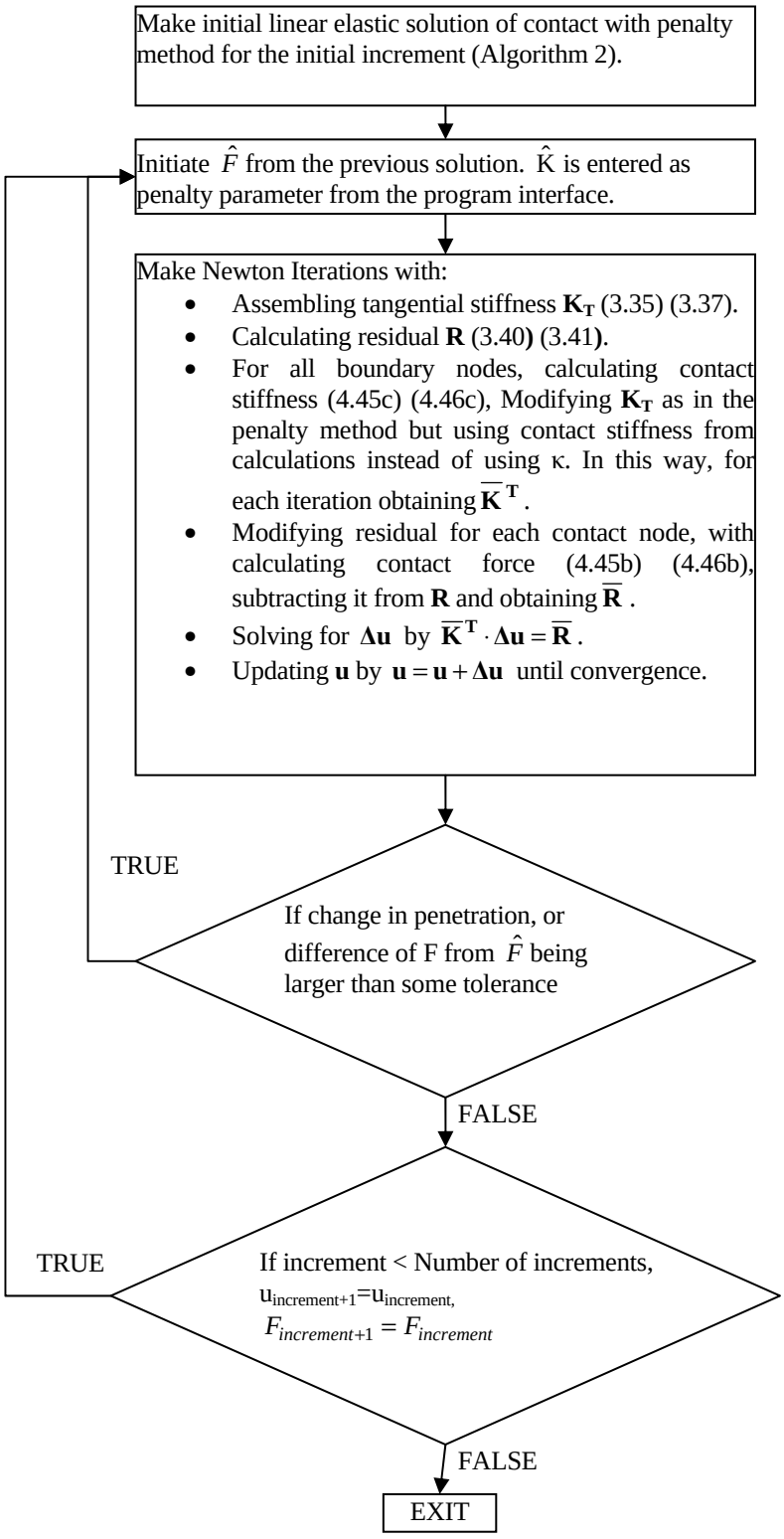
For applying Eq. (4.47) to non-linear equations, also needed is the writing of it in linearized form.

$$D(\delta\Pi_c^i) = \begin{cases} \underbrace{\delta g_N^i \cdot \hat{K} \cdot e^{\frac{\hat{F}^i}{\hat{K}} g_N^i} \cdot \Delta g_N^i}_{K_c} + \underbrace{\hat{\mathrm{F}}^{\mathrm{i}} \cdot e^{\frac{\hat{F}^i}{\hat{K}} g_N^i} \cdot \delta\Delta g_N^i}_{0} & \text{if } \mathrm{g}_{\mathrm{N}}^{\mathrm{i}} \geq 0 \\ \underbrace{\delta g_N^i \cdot \hat{K} \cdot \Delta g_N^i}_{K_c} + \underbrace{\hat{\mathrm{F}}^{\mathrm{i}} \cdot \Delta\delta g_N^i}_{0} & \text{if } \mathrm{g}_{\mathrm{N}}^{\mathrm{i}} < 0 \end{cases}$$

(4.48)

The contact linearization in Eq. (4.48) can be adapted to the general *Newton algorithm* into the tangential stiffness term as it is done in the linear elastic case, with instead of using κ, using the tangential stiffness terms in Eq. (4.48). For the residual, the update is done by adding contact forces as written in Eq. (4.47).

To explain the method briefly, make initial penalty solution. Initialize $\hat{F}$ for every boundary node. If contact force for a boundary node is zero, initialize it to some small number to prevent division by zero. $\hat{K}$ is entered from the program interface as penalty parameter. Usually 0.1 is being convenient for multiplying with largest diagonal. Still too large a penalty parameter may cause ill conditioning, while too small a penalty results in long computation time. Assemble tangential stiffness defined with Eqs. (3.35) or (3.37) and modify it with contact stiffness K_c in Eq. ((4.48), and modify residual term defined with Eqs. ((3.40) or (3.41), with Eq. (4.47). Repeat this until convergence. If F is too different than $\hat{F}$, update $\hat{F}$ and repeat the full process (Algorithm 3).



Algorithm 3: Cross constraints method.

4.6 CONTACT SEARCH, AND SURFACE DETECTION

Although there are not many contacting surfaces in the case considered in the framework of this dissertation, in case of multi-body contact problems involving complicated surface structure and having large deformation, contact search takes considerable CPU time and should be mentioned for any program developed for the solution of contact. For that purpose, two papers [16], [17] will be referenced since some inspiration has been gained, although implementation of the methods mentioned has not been possible.

"*Contact searching is to detect and keep in trace the contact points in a deformation system, where contact and discontact phenomena occur frequently. This is one of the fundamental abilities required to conduct FE simulation. Usually it includes the local and global search processes. The former is to roughly find all the possible candidates around a specific point. The latter is to find exactly the contact point after the global searching* [16]."

In this framework, contact search process is split into local and global searches. Global search in general involves some index operations to detect which node is contacting on which master surface. The local search involves the exact detection of the nearest point, which is performed with *Newton's algorithm*. According to [16], a maximum of 10 iterations is being enough in general, which is also preferred in the program developed for this dissertation.

The local search is in general the same. The main discussion point in general becomes the global detection. Many methods have been proposed for this purpose until now, but to mention the methods proposed in the two references [16] and [17], they are the inside-outside search algorithm, binary search algorithm and the bucket search algorithm. They are only mentioned for the completeness of the topic, and the details left for a further work on the area. However, the main idea, being the global search and the local search has been applied in the program developed.

In the program developed for this work, whether a contact surface is defined is checked first. If so, a boundary detection process, which will be explained in the sequel, is performed. After that, the global search is performed for each boundary node to detect the nearest contact surface node. Since the nearest surface is locally

defined, each surface node is associated to a surface segment. Thus, at this segment the local search is performed with the *Newton algorithm* to find the nearest surface position.

For the boundary detection process, the element connectivity properties are used. It is performed only once before the beginning of contact analysis. Element nodes are defined in the counter-clockwise direction, nodes know connected elements, and the elements know the connected nodes, by use of which, the boundary nodes may be detected. This function is provided in the *class* **Obj2D** since all the information is contained in it (Appendix p.120).

The algorithm can be summarized as follows:

Every node holds a list of unsigned integer for the next node information. The process has two stages:

1. For every node, connected elements are traced for the next node counter-clockwise to the node in consideration. The next node detected is pushed into the list of next node indices in the node in consideration.
2. A process of deletion of next nodes is performed. For this, again the nodes are traced one by one. If the next node indexed in the list of a node contains in its next node list, the index of the node in consideration, that index is deleted from next node's list, and the index of the next node is deleted from the node in consideration. (i.e. if the nodes are mutually next node of each other, their indices are mutually deleted from list of the other.)

At the end, every boundary node is left with node indices on the counter-clockwise to it if the node is at the external boundary. If the node is at an interior of the body, the next node list will be empty. If the node is at the boundary of an interior hole, the next node will be in clockwise direction (see Figure 4.9 and Table 4.1).

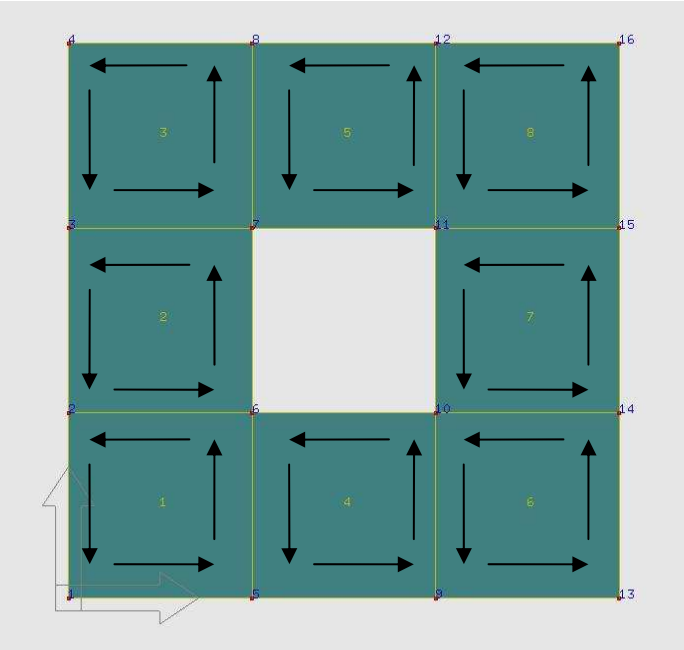


Figure 4.9: Boundary node detection system. Next nodes are entered by tracing elements in row directions in the first stage.

Table 4.1: Table of next nodes for boundary detection. The grey colored indices are the deleted ones in the second stage.

Node	Next Node List		
1	5		
2	1	6	
3	2	7	
4	3		
5	6	9	
6	2	7	5
7	3	8	11
8	4	7	
9	10	13	
10	6	9	14
11	12	10	15
12	8	11	
13	14		
14	10	15	
15	11	16	
16	12		

CHAPTER 5

IMPLEMENTATION ISSUES

5.1 INTRODUCTION

FEM programming has evolved in the last two decades with the evolution of the computer programming techniques. Introduction of OOP techniques have given possibility to develop general robust and structured programming solutions to complicated problems in the last decade.

"*Recent developments in software engineering and in the field of object oriented C++ programming have made it possible to model physical processes and mechanisms more expressively than ever before*" [23].

"*Much of the early research on FEM implementation has focused on the speed of execution or equation solving. However, as the complexity of finite element programs increase, it is obvious that improving the maintainability, extendibility and reusability of the software is equally important*"[24].

Similar ideas are also written and explained in [25], which gives a bibliography of OOP FEM programming. The differences of OOP from traditional FORTRAN programming and concepts of OOP are elaborated with some detail.

In this chapter, the modern techniques of FEM programming are presented briefly. This chapter includes OOP programming concepts and techniques with C++, and the application to contact solution in 2D. In this regard, to give some references, Strustrup [18] is the monumental book for learning C++. Nevertheless, it would not be enough to develop a complicated FEM program. To give some other references of C++ programming, the references [19], [20], [21], [22]may be offered. Especially for the OOP FEM programming techniques, papers [23], [24], [25], [26], [27], [28] could be followed. An implementation program has been written in C++ by use of the general concepts developed from those books and papers. The papers present their

own approach on their own implementation issues, not being the same, but not being much different in the programming point of view. Now, the details are ready to be presented for the implementation issue in consideration.

5.2 CLASS STRUCTURE

A class is *user-defined type*, where *type is a concrete representation of a concept* [18]. In the FEM context, there are nodes, elements, domains, Gauss points… All these have specific data and specific relations with each other. OOP provides them to be represented conceptually as they are, and organize the data and functions conveniently in a structured manner. It also provides separate compilation and error detection mechanisms. For large problems, with complicated organization, those considerations become indispensable.

In a FEM program, there is the domain, which is defined in the program by *class* **Obj2D**, and it is decomposed into sub domains called elements, which are declared in the program by *class* **El2D**. In addition, there are the nodes, represented by *class* **Node2D** to define element edges. The Gauss points, where the integrations and most of the calculations on the element are performed, are handled in the program by *class* **GsPt2D**. For the contact solution, it is also needed to define classes related to contact surface. For that purpose, *class* **Surface2D** is written for the contact surface, and the *class* **CntNd2D** for the contact node. Most of the data have been organized in these classes.

Those classes handle jobs specific to themselves by functions declared in them, called methods. The methods may have common properties, among classes of the same type, handling common jobs by some means, or in all means distinct jobs intrinsic to the object itself. Some common functions may be declared in a virtual abstract base class, and child classes may be derived from it. For instance, all of the mentioned classes are graphical objects, which can be drawn into the graphical environment. All of them may be captured from the graphical interface with a mouse click etc... Those properties are declared in the base *class* **FEGrObj**. Some of the functions may be handled in the base class, or some may need specific handling in the class itself, or else some common part of the function may be implemented in the base and the rest

specific part to be handled in the child. Therefore, other then the classes declared above, there might be base classes in the lower order, and child classes derived from the mentioned ones, which are of higher order.

To give another instance for class relations, consider the element class. As it has been stated in chapter 2, there are two types of planar elements dealt in here, triangular and rectangular. Though they differ in some ways, they have common properties too. For instance, both are composed of Gauss points, nodes, etc… However, the ways they handle drawing or calculating stiffness differ considerably. So two classes *class* **Tri2D**, and *class* **Rec2D** are derived from *class* **El2D.** The class relations are represented simply in Figure 5.1.

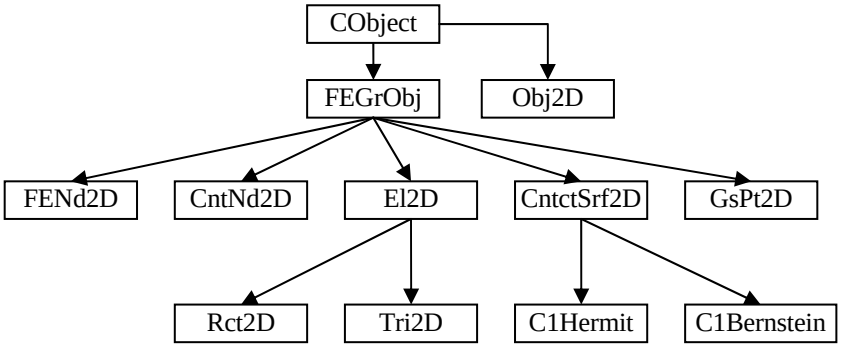


Figure 5.1: Diagram representing class hierarchy of FEM objects. Lower order are the *child classes* derived from higher *parent classes*.

Below given are some brief explanations of the classes in Figure 5.1. They are only to give some idea of how the things handled. The class definition headers are given in the Appendices. For more details, refer to related appendix.

5.2.1 class CObject

In the above diagram (Figure 5.1), *class* **CObject** is the virtual base of all of the classes. It is actually a Microsoft Foundation Class Library (MFC) specific class, and

virtually all of the MFC specific codes are derived from it. It handles the runtime type checking, serialization (saving and reading binary data), and runtime checking of objects for debugging purposes. (For more specific details refer to MSDN Library of Microsoft Visual C++.Net.)

5.2.2 class FEGrObj

This class is derived from *class* **CObject**, and it handles mouse events, like left button down, left button up, mouse move etc… It also has a static variable, pointer to *class* **FEGrObj**, to handle mouse captures called *m_pCaptured*. Since only one mouse would be active in the graphical interface, a static pointer of the same type is defined in the class, this way capturing a single object of this type from the graphical interface. For example, in the event of left button click, the function handling this event in the view class searches for clicked object, which can be a node, an element, a contact node, etc… In case a hit occurs, the address of the hit object is assigned to *m_pCaptured*. Actually, this technique is called *late* or *dynamic binding* [28], which will be elaborated in the sequel for other issues.

5.2.3 class FENd2D

This class is derived from the above *class* **FEGrObj**, and it holds data related to node, like its index, initial coordinates, final coordinates, nodal results etc... Furthermore, it also handles mouse events and drawing of itself into the view. In a graphical environment, a node should know connected elements for stress averaging and error analysis [28], though in old style of FEM programming, only elements knowing connected nodes would be sufficient for assembling purposes. It also needs to store results related to itself and serve those results to the elements connected to itself. It needs to reach to the domain body it is connected to, for reaching elements, which it belongs to, for stress averaging etc… This is achieved by keeping a pointer to *class* **Obj2D**.

5.2.4 class CntNd2D

This class is also derived from *class* **FEGrObj**, as well as *class* **FENd2D**. It holds data related to contact node, like its fixed coordinates, etc. It also handles mouse events and drawing of itself into the view. It needs to reach to the rigid surface definition it is connected to, since a node is associated to a segment of curve, and curve is defined with respect to neighboring contact nodes. This is achieved by keeping a pointer to *class* **CntctSrf2D**.

5.2.5 class El2D

This class is also derived from *class* **FEGrObj**, but it still serves as a virtual base class for specific element definitions (i.e. rectangle, triangle or other element types may be derived). Common data are stored in the class and some common functionality is performed other than pure virtual declaration of some other functions. It stores connected nodes as both pointers and indices, which makes code maintenance easier. Since elements need to reach nodes frequently in drawing issues, element deletion issues, stiffness calculations, assemblage processes, etc., it becomes an urge to reach connected nodes directly, which is achieved by keeping pointers of nodes in elements. For serialization issues, setting degrees of freedoms etc., it needs the node indices.

Elements also hold *Gauss points* for calculating necessary data. An element object has functions to calculate element stiffness namely linear and non-linear stiffness. Linear stiffness is calculated by Eq. (3.19) for linear elastic analysis and by first integrations in Eq's. (3.35), or (3.37) for non-linear elastic case. The non-linear stiffness is calculated by Eq. (3.31) or second integration in Eq's (3.35), or (3.37). However, the calculation of element stiffness is specifically performed differently for specific kind of element. Thus, *class* **El2D** should have the declaration of common functions only. The Implementation of those functions is specific to the derived child class. It also has to deduce solution specific parameters from the data of *class* **Obj2D**. For instance, it has index of the material definition of itself, and material definitions are actually kept at *class* **Obj2D**. Thus, it should also hold the address of the connected body.

5.2.6 class Rct2D and class Tri2D

These classes are derived from *class* **El2D**. They handle specific jobs related to themselves. The intrinsic implementation of pure virtual functions defined in *class* **El2D** is performed in those classes separately. For instance actual implementation of how they will be drawn into the graphical interface, how they will respond to mouse events specifically is written and defined in them. They contain all properties of *class* **El2D**, that is those classes inherit from it all the variables, and methods.

5.2.7 class CntctSrf2D

This class is also derived from *class* **FEGrObj**, since it is a graphical object to be drawn into the view and to be captured from the view. It serves as a base class for discrete contact surface definitions. Finding nearest contact node for a FEM node is a common functionality to be supported. In addition, finding nearest surface point, drawing are purely virtual functions to be declared in here.

5.2.8 class C1Hermite2D

This class is derived from *class* **CntctSrf2D**. It implements the functionality defined in Section 0 for the *Hermit surface*, defined by the functions in Eqs. (4.8), (4.9), and (4.10). It handles specific drawing and mouse handling issues for itself. Furthermore, it contains the implementation of finding the nearest surface point for a FEM node.

5.2.9 class C1Bernstein2D

This class is also derived from *class* **CntctSrf2D**. It implements the functionality defined in Section 0 for the *Bernstein surface*, defined by Eqs. (4.11) and (4.12). It handles specific drawing and mouse handling issues for itself. It contains the implementation of finding the nearest surface point for a FEM node specifically for this surface definition.

5.2.10 class GsPt2D

This class handles most of the calculations. Every element stores as many of this class as the integration point number. For instance a rectangle with four nodes must have four integration points for exact evaluation of the integrations, while a triangle having three nodes must have three integration points for the same purpose [2][3][4]. It has the local coordinates of itself, which are defined on the master element (Figure 3.1), and the global coordinates defined on the real element patch. It also has graphical coordinates for drawing issues. It serves the functions for calculating stresses, strains, Jacobian (Eq. (3.5 a, b)), deformation gradient (Eq. (3.8)), determinant of deformation gradient, derivative matrices (Eqs. (3.10), (3.27), (3.32)), etc… It also needs to keep the address of the element it is belonged to, which is achieved by holding a pointer of *class* **El2D**. However, for calculations, it needs to know the connected element type, which is achieved by *runtime type information* (RTTI) checking. For instance, rectangular element has different base function definitions than triangle. Base functions are defined as global functions in the program. Therefore, it needs to know which type of element is connected to itself for correctly calling the base functions. Due elements knowing the connected domain, it has indirect access to connected domain defined by *class* **Obj2D**.

5.2.11 class Obj2D

This class is derived from *class* **CObject**. It stores the elements, and nodes constituting itself. How elements and nodes kept in the class is a very technical issue. For this purpose Standard Template Library (STL) is utilized [21][22].

There are many ways of storing data in a container, namely, vector, list, map, multi-map, set, multi-set, etc… It would be a challenging task to select which type of a container to use. This is one of the first issues to be resolved to write a good FEM program. A user should easily interact with the program, that is, delete, or add nodes or elements in the preprocessor. Also left as a further task, on the runtime, mesh refinement possibilities must be considered. STL serves as a perfect tool to do such tasks. In the program developed, vector, holding pointer to objects is selected as best suiting in the framework of this dissertation. Knowing STL helps making judicious

decisions on container types. In some of the implementations, list is preferred, while in some others, map implementation is selected. Both have been tried during the development but the implementation of a vector of pointers is judiciously selected as best appealing for giving most flexibility and functionality.

5.3 IMPLEMENTATION DETAILS

Here, some minor specific details of calculation in the OOP environment will be elaborated. They are important and without understanding them, FEM programming becomes painful for the program developer. The issues considered in this section are critical to be able to implement all the things mentioned up to now.

5.3.1 Copy Constructors, Assignment Operators and Destructors

Since the class structure is so much interconnected with keeping addresses of each other, address tracking is important and should be given special attention. For instance, in element copy process, since ***class GsPt2D*** keeping the address of the element it is connected to, copied Gauss points must be initialized to new element address. In addition, this kind of details should be given special attention in the assignment operators. Another special subject to be shared is the destruction of *class* ***Obj2D***. In the destructor, it must delete all the elements and nodes it stores as pointers in the vectors. Those issues are slightly touched and mentioned in [28]. Constructors and destructors must be designed carefully in order to design strong codes, not crashing on the run time.

5.3.2 Element Transformations

The implementation of Jacobian J is exactly formulated as in Eqs. (3.5 a, b) in a single function in *class* ***GsPt2D,*** and ξ, η not being arguments, but as private variables in ***class GsPt2D***. For calculation of **J,** the argument to be entered is the current increment number, which for the linear elastic case entered as zero. The calculation of *deformation gradient* **F** is implemented in the same way as in Eq. (3.8) in *class*

GsPt2D, but with two variables being the reference increment and the current increment numbers, and ξ, η are held in the class itself as private variables. The inverse is achieved by switching the reference and current increment numbers for the arguments in the same function. A Gauss point should be initialized to the element it is connected to, and to the master coordinates where it belongs to, before its any attempt of use. When initialized, its global position is set automatically in the initialization function.

5.3.3 Late or Dynamic Binding

Now consider how contact calculations are performed. Boundary nodes are detected and constraint is applied only at those nodes. However, for application of contact constraints to nodes, several parameters are necessary as the surface normal, nearest surface position normal gap etc… To keep and handle all those parameters in nodes would be expensive, since they are needed only for possible contact nodes (i.e. boundary nodes). In addition, keeping all those parameters in a structured manner becomes crucial. To achieve that, a class to provide those variables is created called *class* **CntParamVals**. Instead of keeping all those variables in nodes, nodes are only provided with a pointer of type *class* **CntParamVals**. A pointer is of size 32 bits, being much cheaper compared to keeping all those values in all the nodes. This pointer is initially assigned to NULL. In the case of contact solution, when boundary nodes are detected, a new object of type *class* **CntParamVals** is allocated on the run time for each boundary node, and assigned to these boundary nodes's pointer variable. This type of binding is called the *late* or *dynamic binding*. In [28], this name is mentioned but any detail is not given as in here. In the FEM context, this method may be very efficient for different types of problems. For instance in plastic analysis, Gauss points need to hold plastic deformation history in case plastic deformations to occur. Though plastic analysis is not implemented in the context of this dissertation, this method may be very convenient and efficient in that case too. This issue stands as a further work to develop in the authors mind.

5.3.4 Program Interface

The program interface must be designed such that, the user would be prevented from erroneous selections or entering wrong parameters to the program as much as possible. A text based FEM program would have an input file and when the program is executed, it would generate an output file. In a graphical interfaced program as the one developed here for solution of contact, the program must have preprocessor and post processor. In the case considered, both are implemented in the same environment. However, the interaction with the program should change according to status of the solution environment (i.e., preprocessing, post processing). In addition, drawing of the body, capturing of elements of drawing, (i.e. nodes, elements, contact nodes etc…) must be done efficiently, effectively, and fast. For this purpose, *Open Graphics Library* (OpenGL) is utilized [29] [30]. *OpenGL is designed as a streamlined, hardware-independent interface to be implemented on many different hardware platforms* [29]. It provides coordinate transformation, selection and feedback utilities, drawing of lines and polygons. It has also the 3D support for further development of the FEM program; however, some primitive knowledge of OpenGL is enough for the drawing issues considered in here.

In the program developed, the switch between preprocessor and the post processor is achieved by a flag, handling if the analysis is complete. In case of preprocessing, there is the solution parameters dialog bar being active, and any change to the body in consideration is possible, like moving nodes, element addition or deletion, node addition or deletion, etc. In the analysis parameters bar, there are the options of which kind of analysis is to be performed (plane stress, plane strain, axi-symmetric), whether non-linear analysis is to be performed, non-linear analysis parameters, contact solution parameters, etc. (Figure 5.2).

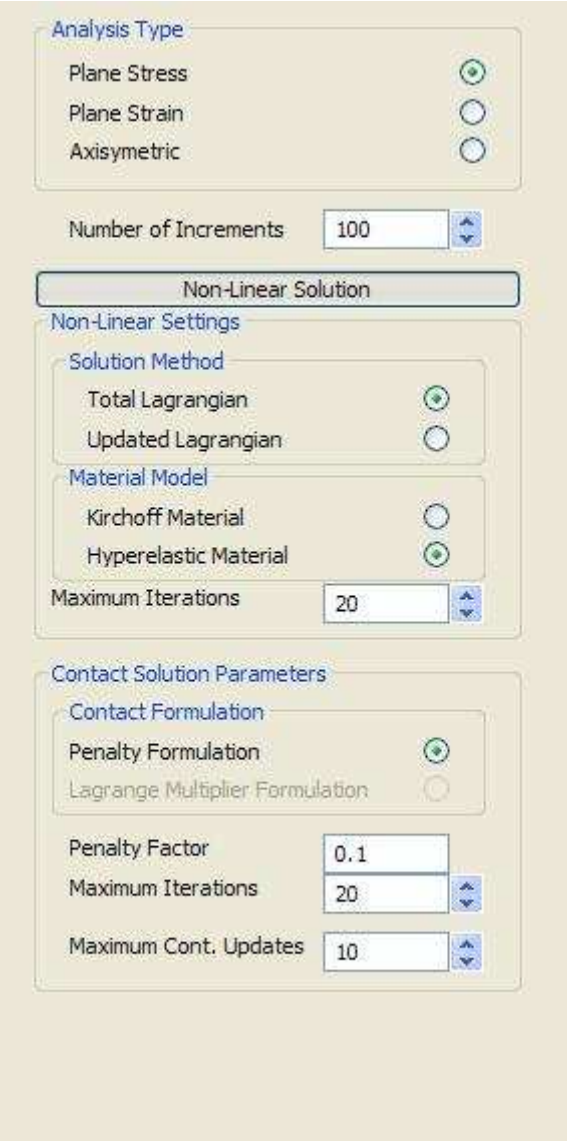


Figure 5.2: Input SideBar dialog.

However, in case when the analysis is complete, those options must be automatically disabled, and the sidebar must change automatically from solution parameters to results parameters (Figure 5.3). In the result parameters bar, there are the options of viewing the object in displaced shape, selection of displaying deformations, stress components, strain components, color ranging.

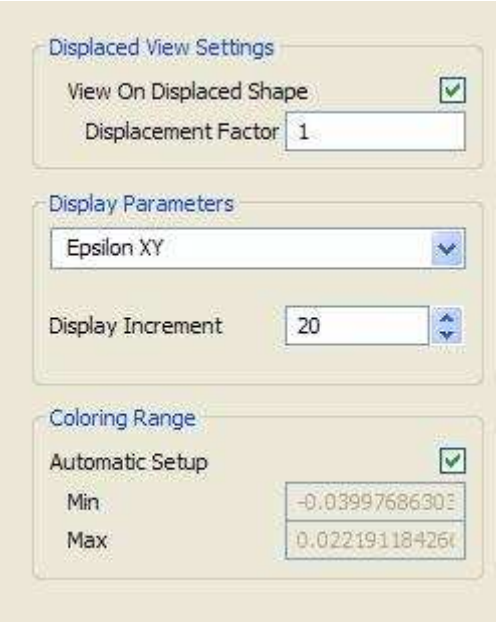


Figure 5.3: Result parameters SideBar dialog.

There are common functions of both preprocessor and the post processor, being the zooming options, zoom in, zoom out, dynamic zooming, window zooming, going to previous zoom are the provided functionality.

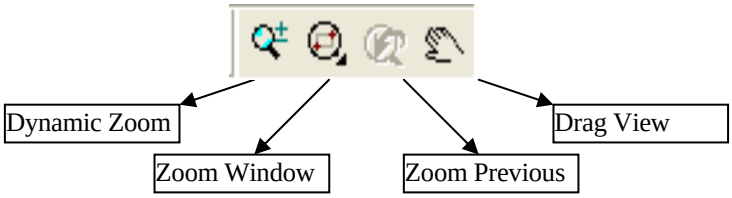


Figure 5.4: Zoom ToolBar providing interface for zoom functions.

Other than those, viewing options, like node sizes, text sizes, color selections are provided with the view settings *SideBar*. *Menu bar* is provided for further optional selections as is used in today's modern software. Snapping to grid, setting grid sizes,

setting window sizes, regular mesh generation, etc. are other functionalities provided. *StatusBar* provides aid in understanding functions of buttons in the program, writes coordinates of mouse position, and when analysis is complete provide with interpolated scalar values on the element on which mouse is moving.

It is not possible to provide information about all functions and abilities of the program here, but the usage is straightforward to understand when playing with the program interface for those familiar with graphical interfaced programs. A setup program and sample files are provided with the CD attached.

CHAPTER 6

TEST PROBLEM COMPARISONS AND BENCMARK PROBLEM

6.1 INTRODUCTION

In this section, some benchmark and test problems will be solved to verify the results. First, the results without contact, the internal consistency of the program for different solution selections will be tested, and the results will be interpreted. Then some comparisons with commercial program ABAQUS will be given, with and without contact.

6.2 NON-LINEAR BUCKLING

Here, a cantilever loaded axially at the tip with a small perturbation lateral force is selected as a test problem. The length of the bar is 800 units, section depth of 100 units and section thickness of 1 unit is selected. Elastic modulus is 1000, and the Poisson's ratio is 0.3. The bar is subdivided into 20 by 4 elements. Support conditions are as seen on the figure. This beam should buckle around 321.28 units of axial tip load according to Euler Beam Theory. In the analysis axial tip load entered as 300/100 units per unit length and tip shear load 10/100 units per length (Figure 6.1). Maximum tip deflections are given in (Table 6.1).

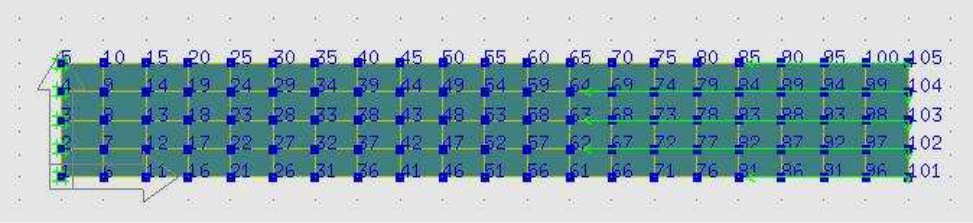


Figure 6.1: The analyzed cantilever model. The same model is analyzed with TL and UL approaches for both plane stress and plane strain cases.

Table 6.1: Analysis results for cantilever loaded axially with small perturbation lateral force for *Plane Stress* and *Plane Strain* analysis. The system analyzed by both *Total Lagrange* and *Updated Lagrange* methods and by *Kirchoff Material* and *Hyperelastic Material* models.

	Abs. Max. Tip Defl.	Plane Stress Analysis		
		Total Lagrange	**Updated Lagrange**	**Abaqus Result**
Kirchoff Model	**U**	33.23	33.23	
	V	148.01	148.15	
Hyperelastic Model	**U**	29.38	29.64	29.04
	V	136.40	137.17	135.30
	Abs. Max. Tip Defl.	**Plane Strain Analysis**		
		Total Lagrange	**Updated Lagrange**	**Abaqus Result**
Kirchoff Model	**U**	15.65	15.73	
	V	84.07	84.44	
Hyperelastic Model	**U**	14.58	14.69	20.19
	V	79.60	80.12	103.80

The above test is performed to check the consistency of the TL and UL approaches for both *Kirchoff* and the *hyperelastic models*. From the above table, it is obvious that TL and UL approaches give approximately same results. The results seem different from the ABAQUS results especially for Plain Strain case. However, the Hyperelastic model used in ABAQUS is different. There are numerous *hyperelastic models* in literature, which are devised for different material characteristics. For the details of the *hyperelastic model* used in ABAQUS, refer to ABAQUS help manuals. It is known that Kirchoff model may give unrealistic results in large strain case, but for slender beam, it can handle non-linear behavior and give realistic results.

6.3 BEAM ON RIGID FOUNDATION

This time, same mesh is analyzed for *plane strain* case with same material constants as in the previous section, but this time with a rigid linear base, with different boundary conditions on the left side and different loading conditions. The left bottom edge is fixed, while other nodes on the left side are only supported in x-direction. Instead of tip loading, it is loaded at the left top along 25% of its length with 1.0 units/length of downward force (Figure 6.2). It is expected that, right edge is elevated and contact is lost, while on the left edge along the contact surface, there is the contact reaction.

The beam is analyzed as *linear-elastic* and *hyperelastic*. The results are compared to ABAQUS. In ABAQUS, contact is modeled with *augmented Lagrange approach*. In the program developed for this dissertation, *Lagrange multiplier approach* is preferred for linear elastic case while *cross constraints method* is used for *hyperelastic* non-linear elastic case.

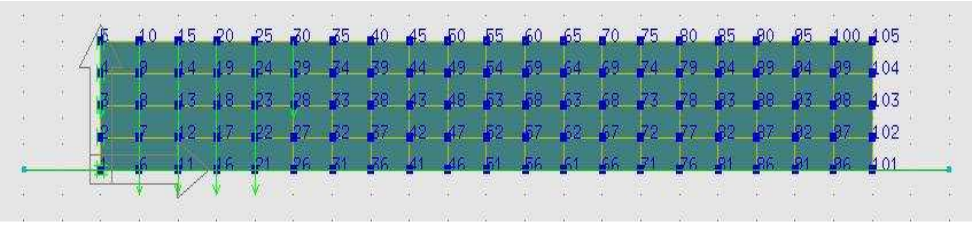


Figure 6.2: Beam on rigid foundation. This is at the preprocessor stage of new program developed (i.e. not analyzed yet).

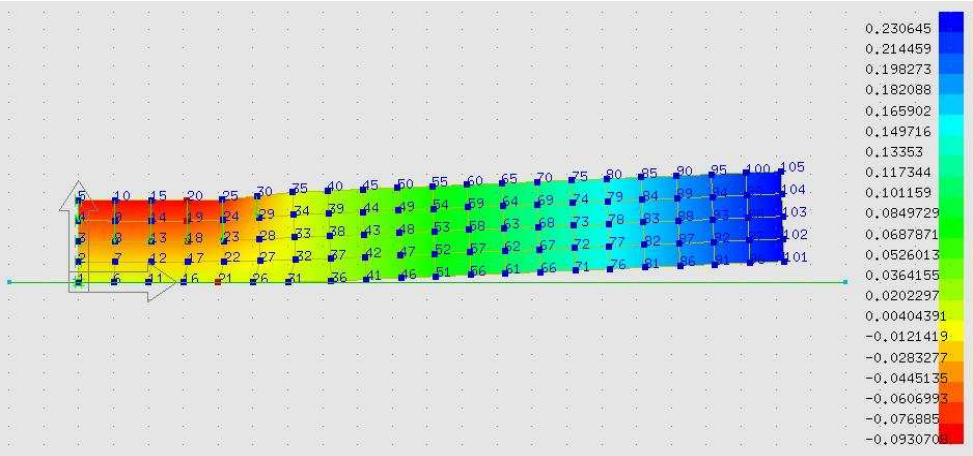


Figure 6.3: Beam on rigid foundation analyzed with the program developed. *Linear Elastic* case with *Lagrange Multiplier Approach* is considered. Vertical displacements are pictured.

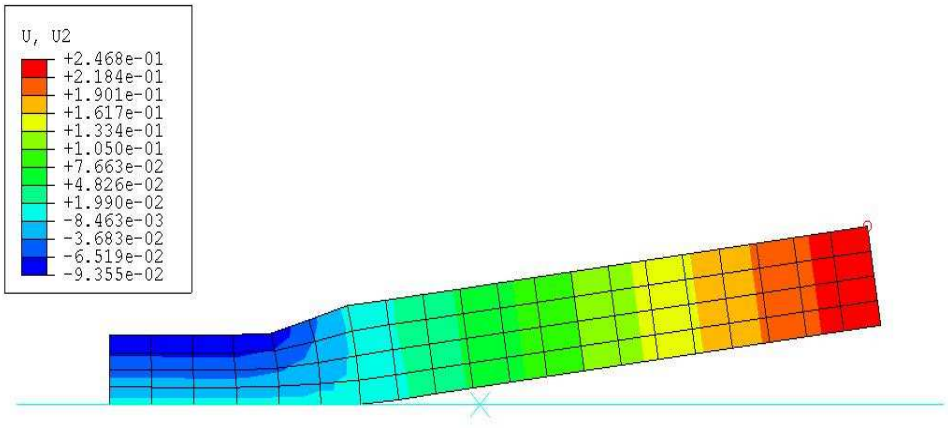


Figure 6.4: Beam on rigid foundation analyzed with the ABAQUS commercial program. *Linear Elastic* case with *Augmented Lagrange Approach* is considered. Vertical displacements are pictured.

In the above, (Figure 6.3) and (Figure 6.4), vertical displacements are plotted. From the figures, it is obvious that the results are comparable for the linear elastic case. The

small difference may be due to different contact handling technique used in the new program developed and the commercial program ABAQUS.

6.4 CIRCULAR DISK ON RIGID FOUNDATION

This time the same mesh as in the previous section (Figure 6.2) with the same material constants is analyzed, but with the axisymmetric analysis option. The same loading conditions and the same boundary conditions on the left side have been applied. Result is compared to ABAQUS and comparable results have been obtained.

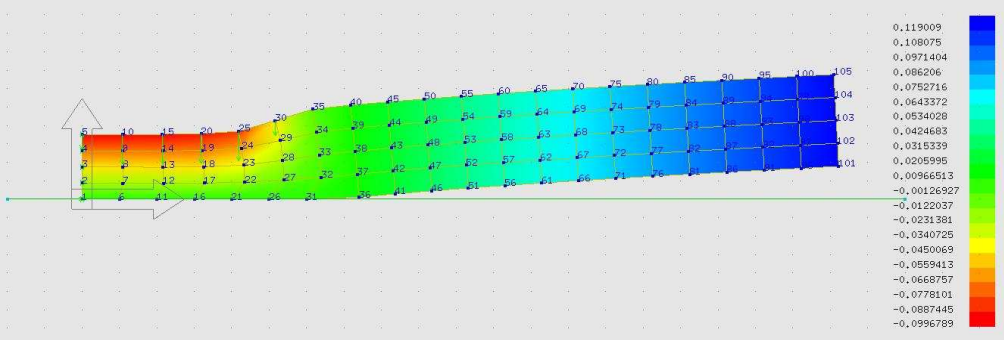


Figure 6.5: Circular disk interacting with rigid foundation. Axisymmetric analysis with the new program developed for this dissertation. Linear elastic case analyzed with *Lagrange multiplier approach*.

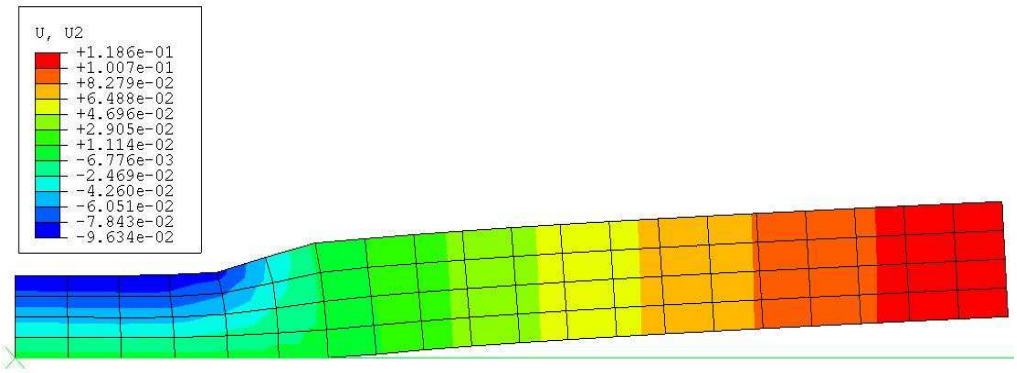


Figure 6.6: Circular disk interacting with rigid foundation. Axisymmetric analysis performed with the commercial ABAQUS program. Linear elastic case with *augmented Lagrange approach* is the analysis options.

In the linear elastic case, comparable results have been obtained. The small difference in displacements may be attributed to different approaches in handling the contact conditions.

6.5 THE BENCHMARK PROBLEM

The benchmark problem is inspired from an industrial application, the analysis of Ericsson cell phones charging plug (Figure 6.7). A model is tried visually, not by measure. Only half bottom is modeled because of the symmetry for the upper part. This analysis is performed only to show that, the program can handle this kind of an interesting complicated analysis, and industrial applications may be performed with the program.

For the model, both the triangular and the rectangular element formulations are used coherently, with *Hermit Interpolation surface* representing the phone side. It is pressed 4.25 mm against the contact surface in x-direction. Non-linear Hyperelastic analysis performed in 100 increments of Newton iterations. For application of contact, cross constraints method is used. Solution took around 1 hour. The result at the 60^{th} increment is represented in Figure 6.8.

The problem is interesting for an analytic solution is expected to be impossible for that complicated problem, and it represents a challenging engineering application. It does not seem possible to compare it with other programs.

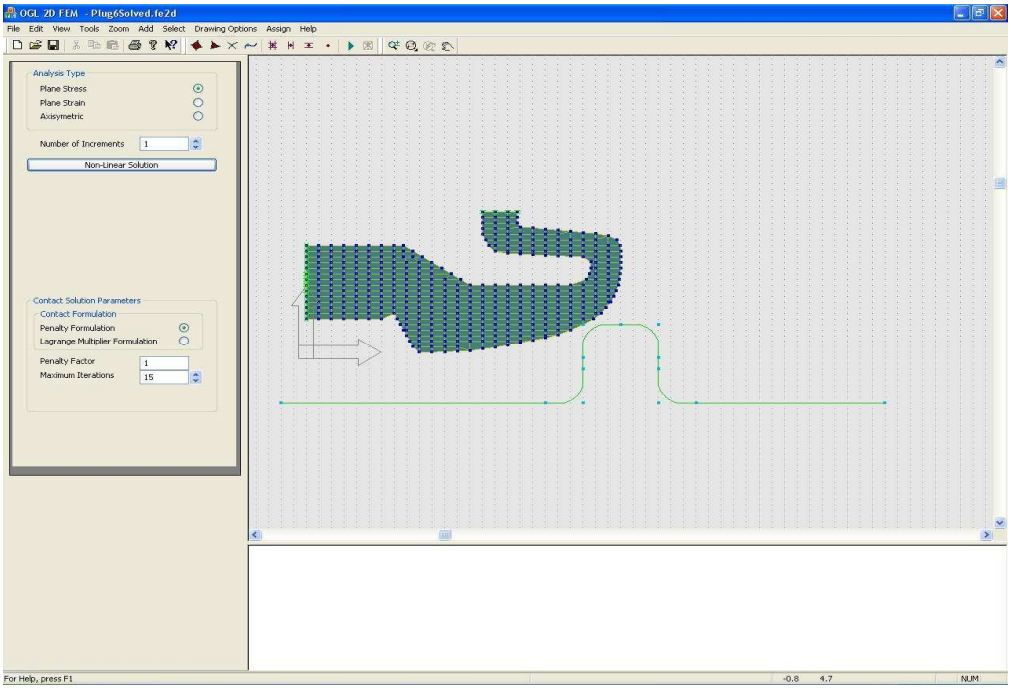


Figure 6.7: The benchmark problem. Plug in the preprocessor stage. This is the model entered from the graphical interface. The *Dialog Bar* on the left is in the preprocessor state.

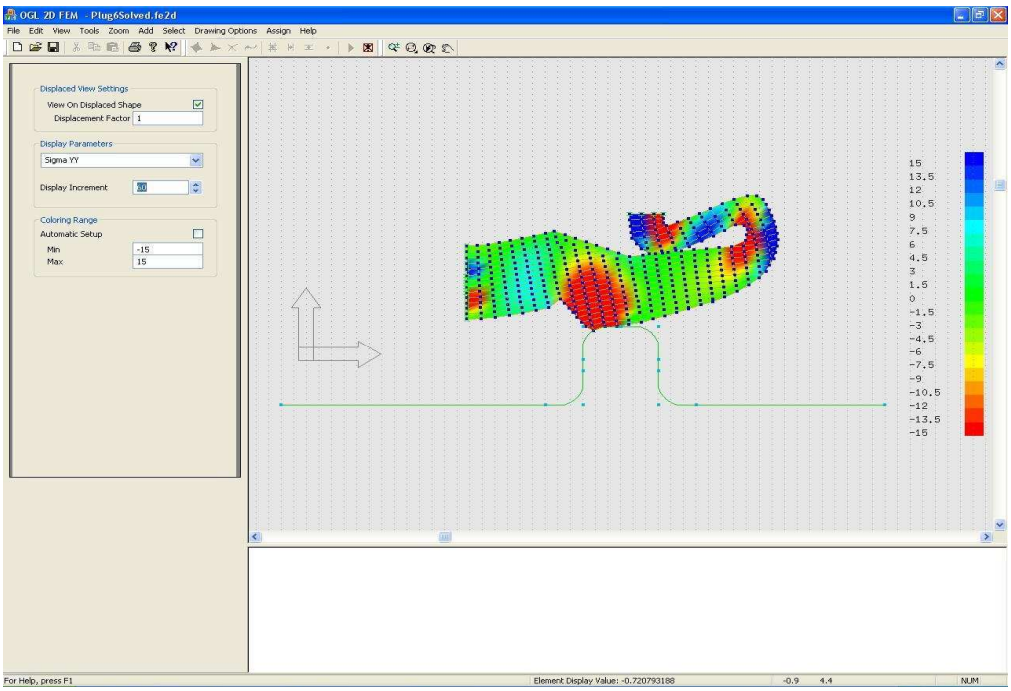


Figure 6.8: The benchmark problem. Plug in the post processor stage. This is the result of the analysis representing y-direction Cauchy's stress distribution when pushed against to the contact surface in x-direction. The *Dialog Bar* on the left is in the post processor state.

6.6 TESTING WITH ANALYTIC RESULTS

In general, it is difficult to test this kind of a problem in a good way with analytic results, due the method developed in here is being numerical approximation, and the it includes different complicated aspects of the problem. .In [31], result for infinite elastic beam resting on elastic half-space is given (Figure 6.9). To compare the results, model mesh in (Figure 6.2) is used. In [31], two ratios for elastic layer and rigid half-space, $\lambda_{cr} = P / Pe \cdot h$, is given for critical elevating load *P*, being tensile and compressive as 1.088 and 44.139. In the model, for the program developed, around 1.2 and 45, the separation occurs. The result obtained is comparable to the result in the considered paper.

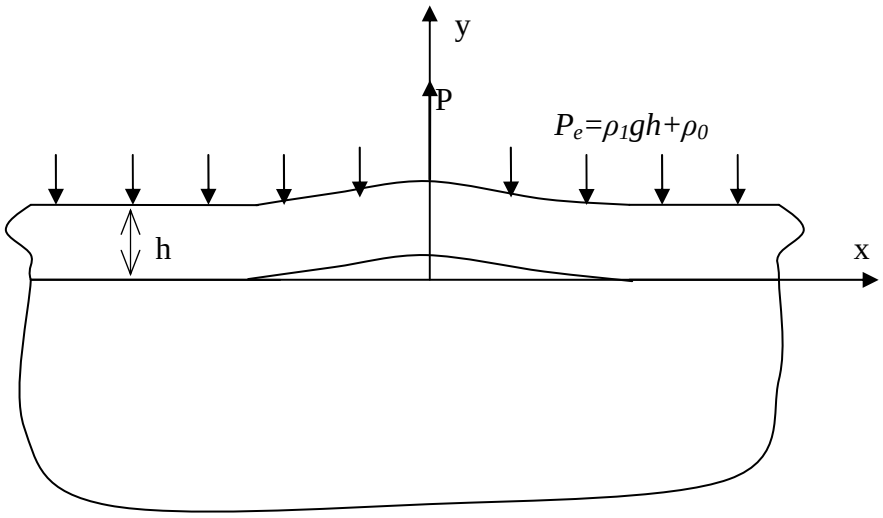


Figure 6.9: Beam on elastic half-space. ρ_0 is the load per unit length, ρ_1 is the load per unit volume, g is the gravitational constant.

CHAPTER 7

FURTHER REMARKS AND CONCLUSION

7.1 INTRODUCTION

In this chapter, further development issues and the achievements with the program developed is discussed.

7.2 FURTHER DEVELOPMENT ISSUES

At the very beginning, the aim is declared as writing an extendible program, which solves contact problems. The program is developed in view of this aim, considering further development issues. Though writing a perfect program handling every aspect of a problem at once is never possible in consideration of the extent of FEM, some near future development issues may be foreseen. Most of the technical details would stand valid, and as program evolves, by the gained expertise, the art of science can be flourished with the evolved new ideas.

As the first attempt, different surface definitions; i.e NURBS curves [10] as mentioned before, and other analytic surface definitions, may be implemented. The program may further be developed for contact solution by considering self-contacting of the body, i.e a boundary surface may be fitted onto the detected boundary nodes of the body. Therefore, in that case, surface would also be moving, and that must be handled in a convenient way. Multiple body definitions may be implemented, and interactions of them may be formulated. For that purpose, more efficient global search methods, i.e. bucket search, binary search [16], [17] as mentioned in Chapter 3, may become an obligation. Friction is also a challenging problem, which stands as a further development issue.

Automatic mesh generation is indispensable in the modern FEM programs. Good automatic mesh generators, i.e. *advancing front, Delaunay triangulation* can be implemented.

In the current work only two material models, namely the Kirchoff material model and the Hyperelastic model have been implemented. Other material models may be searched and investigated.

In the future, for a PhD. work, the program may further be developed to handle impact problems, in which dynamic effects must be considered. Plastic analysis may also be implemented. That way metal-forming process can be simulated.

The program written up to now is a good shell core code, around which many other functions and abilities may be woven.

7.3 CONCLUSION

In the framework of this dissertation, unilateral discontinuous contact for an elastic body moving and deforming in space, interacting with a rigid surface only in the normal direction to that surface have been solved numerically, by applying FEM. For the solution of the problem, OOP is seen as crucial instrument, and the OOP programming techniques have been devised for this purpose. By use of OpenGL, a good graphical interface has been created. The problem is solved in 2D for the *plane stress*, *plane strain* and *axisymetric* cases.

For the rigid surface definition, two discrete 2 dimensional surface models have been implemented effectively, namely the *Hermit* and *Bernstein surface* models.

First, a test problem to check the internal consistency of the program is generated for the non-linear analysis, comparing TL and UL approaches. For this purpose, a beam is loaded near to Euler buckling load with a small perturbing tip force. Nice results have been obtained with little numerical deviations for both approaches. It is observed that *Kirchoff material model* and *Hyperelastic material model* give different results as expected. However, results differ from commercial ABAQUS program due to different material models are used.

Some test problems have been solved and compared to the commercial ABAQUS program. For the linear elastic, *small stress, small strain* type of problems, comparable results have been obtained for both plane strain and axisymmetric cases. In that case, the small differences are attributed to the differences in the contact formulation. For the linear elastic case, the new program developed used *Lagrange multiplier approach*, which satisfies contact constraints exactly.

For the non-linear elastic case, due to the use of different hyperelastic models, larger differences have been observed compared to the results of ABAQUS program. ABAQUS program used *Augmented Lagrange Approach* for the solution of contact in all cases. In addition, strain models differ from that program.

It is not possible to check every aspect of the program developed with the commercial programs, since it is devised independently, without consideration of other programs. The program is self-contained as much as possible, proving itself with the application of different methods of solutions. It contains similarities and differences compared to other commercial programs. For instance, *Bezier Curve* is not implemented in ABAQUS though it is in one of the other popular FEM program MARC. Nevertheless, the implementation in MARC is not the same. No test is performed for MARC. ABAQUS does not have surface detection algorithm, since boundary is entered at the very beginning and automatic meshing is performed. After meshing, the user intervention is a bit restricted in most of the popular FEM programs. In the program developed, a regular mesh is directly created, and it can easily be modified in the preprocessor stage. This approach may have advantages and disadvantages compared to other package programs.

In general, the program developed gives good and comparable results, and serves as a perfect core shell for possible further development. The contact problem is solved efficiently for both linear-elastic and non-linear elastic cases for the unilateral frictionless case with that program.

REFERENCES

[1] Wriggers, Peter, *Computational Contact Mechanics*., John Wiley & Sons, England (2002).

[2] Bathe, K. J., *Finite Element Procedures*., Prentice Hall, Englewood Cliffs, New Jersey (1996).

[3] Cook, Robert D., Malkus David S., Plesha Michael E., *Concepts and Applications of Finite Element Analysis*., John Wiley & Sons, New York (1989).

[4] Reddy, J. N., *An Introduction to the finite element method*., McGraw-Hill, Inc., Singapore (1993).

[5] Belytschko, T., Liu, W., K., Moran, B., *Nonlinear Finite Elements for Continua and Structures*., John Wiley & Sons, New York (2000).

[6] Simo, J. C., Hughes, T.J.R., *Computational Inelasticity*., Springer-Verlag, New York, Berlin Heidelberg (1997).

[7] Bonet, Javier., *Nonlinear continuum mechanics for the finite element analysis*., Cambridge University Press, USA (2000).

[8] Nash, Stephen G., Sofer Ariela., *Linear and Nonlinear Programming.*, The McGraw-Hill Companies, Inc., Singapore(1996).

[9] Wriggers P., Krstulovic-Opara L., Korelc J., "Smooth C^1-interpolations for two-dimensional frictional contact problems.", *Int. J. Numer. Meth. Engng.*, 51:1469-1495 (2001).

[10] Stadler M., Holzapfel G. A., Korelc J., "C^n continuous modeling of smooth contact surfaces using NURBS and application to 2D problems.", *Int. J. Numer. Meth. Engng.*, 57:2177-2203 (2003).

[11] Zavarise G., Wriggers P., "A superlinear convergent augmented Lagrangian procedure for contact problems.", *Engineering Computations*, 16:88-119 (1999).

[12] Pietrzak G., Curnier A., “Large deformation frictional contact mechanics: continuum formulation and augmented Lagrangian treatment” *Comput. Methods Appl. Mech. Engrg.*, 177:351-381 (1999).

[13] Kloosterman G., van Damme R. M. J., van den Boogaard A. H., “A geometrical-based contact algorithm using a barrier method.”, *Int. J. Numer. Meth. Engng.*, 51:865-882 (2001).

[14] Bathe K. J., Bouzinov P.A., “On the constraint function method for contact problems.”, *Computers & Structures*, 64:1069-1085 (1997).

[15] Zavarise G., Wriggers P., Schrefler B. A., “A method for solving contact problems.”, *Int. J. Numer. Meth. Engng.*, 42:473-498 (1998).

[16] Wang S. P., Nakamachi E., “The inside-outside contact search algorithm for finite element analysis.”, *Int. J. Numer. Meth. Engng.*, 40:3665-3685 (1997).

[17] Heinstein M. W., Mello F. J., Attaway S. W., Laursen T. A., “Contact-impact modeling in explicit transient dynamics.”, *Comput. Methods Appl. Mech. Engrg.*, 187:621-640 (2000).

[18] Stroustrup, Bjarne., *The C++ Programming Language*., Addison-Wesley, Massachussets, (1998)

[19] Ford, W., Topp, W., *Data Structures With C++*, Prentice Hall, Inc., Englewood Cliffs, New Jersey, (1996)

[20] Murray, W. H., Pappas C. H., *Visual C++.NET: The Complete Reference*, McGraw-Hill, California, (2002)

[21] Austern, M. H., *Generic Programming and the STL: Using and Extending The C++ Standard Template Library*, Addison-Wesley Longman, inc., Massachussets, (1998)

[22] Musser, D. R., Saini, A., *STL Tutorial and Reference Guide: C++ Programming with the Standard Template Library*, Addison-Wesley, Reading, MA, (1996)

[23] Zabaras N, Srikanth A., "An object-oriented programming approach to the Lagrangian FEM analysis of large inelastic deformations and metal forming processes.", *Int. J. Numer. Meth. Engng.*, 45:399-445 (1999).

[24] Yu L., Kumar A. V., "An Object-Oriented Modular Framework For Implementing The finite Element Method", *Computers & Structures*, 79:919-928 (2001).

[25] Mackerle Jaroslav, "Object-oriented programming in FEM and BEM: a bibliography (1990-2003).", *Advances in Engineering Software*, 35:325-336 (2004).

[26] Pantale O, Caperaa S, Rokotomala R., "Development of an object-oriented finite element program: application to metal-forming and impact simulations.", *Journal of Computational and Applied Mathematics*, 341-351 (2004).

[27] Gil Lluis, Bugeda G., "A C++ object-oriented programming strategy for the implementation of the finite element sensitivity analysis for a non-linear structural material model.", *Advances in Engineering Software*, 32:927-935 (2001).

[28] Balopoulos V. Abel J. F., "Use of shallow class hierarchies to facilitate object-oriented nonlinear structural simulations.", *Finite Elements in Analysis and Design*, 38:1047-1074 (2002).

[29] Woo, Mason., Neider, Jackie., Davis, Tom., Shreiner, Dave., OpenGL Architecture Review Board, *OpenGL Programming Guide*, 3rd ed., Addison Wesley Longman, Inc., Massachusetts (1999).

[30] Hearn, Donald., Baker, Pauline M., *Computer Graphics with OpenGL*, 3rd Ed., Pearson Prentice Hall, Pearson Education Inc., USA, (2004).

[31] Geçit M. R., "A Tensionless contact without friction between an elastic layer and an elastic foundation", *Int. J. Solids Structures*, 16:387-396, (1980).

APPENDIX

FEGrObj.h

```
class FEGrObj : public CObject
{
public:
        FEGrObj();
        virtual ~FEGrObj();
        static FEGrObj* m_pCaptured;   //For mouse capturing;

        BOOL m_bSelected;              //Bool to hold object selection state

        virtual void Draw(CFEMGLView* pView)=0;

        virtual void OnLButtonDown   ( CFEMGLView* pView,
                                       UINT nFlags, CPoint point) = 0;
        virtual void OnLButtonUp     ( CFEMGLView* pView,
                                       UINT nFlags, CPoint point) = 0;
        virtual void OnMouseMove     ( CFEMGLView* pView,
                                       UINT nFlags, CPoint point)=0;
};
```

FENd2D.h

```
class FENd2D : public FEGrObj
{
public:
	DECLARE_SERIAL(FENd2D)
	//Constructors
	FENd2D( double X=0.0, double Y=0.0,
		unsigned Idx=0, class Obj2D *Ob=NULL);
	FENd2D(const FENd2D& Nd);

	Obj2D* Obj;	//Pointer to the connected domain

	virtual ~FENd2D();
	double X;
	double Y;
	double x;
	double y;

	BOOL m_bUxdefined;
	BOOL m_bUydefined;
	double m_dBCX;	//Boundary Condition in Y
	double m_dBCY;	//Boundary Condition in X
	double m_dKx;	//Spring constant in X
	double m_dKy;	//Spring constant in Y
	double Rx;	// Reaction force in X
	double Ry;	// Reaction Force in Y

	vector<class NodalRes2D> m_VResults;	//Vector of results structure
	vector<unsigned> ElLst;			//Connected element list

	//List for boundary node detection
	//Firs index stores previous node, second index stores next node!
	list< pair<unsigned, unsigned> > Neighbours;

	static GLint m_nDisplySz;
	static CFont  m_NdFnt;
	static BOOL   m_bVwNodes;	//If to draw nodes on the screen;
	static BOOL   m_bVwIdx;	//If to draw nodal indices on the screen;
	static BOOL   m_bVwNdLd;	//Boolean to view node on the display
	static BOOL   m_bVwNdBC;	//Boolean to view node boundary condition
	static float m_fLdDispFc;	//Load display factor
	static unsigned m_nNdTxtSz;	//Text height for drawing indices

	void SetIdx(unsigned i);	//Setting index of the node in the node list.
	unsigned GetIdx() const
	{return Idx;}

	void operator =  (const FENd2D &N); //Assignment operator

	virtual BOOL   operator == (const FENd2D &N)const;
	virtual BOOL   operator != (const FENd2D &N)const;
```

```
        void Draw(CFEMGLView* pView);
        void DrawBdry(CFEMGLView* pView);
        void DrawSprng(CFEMGLView* pView);

        //Mouse Function declarations
        void OnLButtonDown  ( CFEMGLView* pView,
                              UINT nFlags, CPoint point);
        void OnLButtonUp    ( CFEMGLView* pView,
                              UINT nFlags, CPoint point);
        void OnMouseMove (CFEMGLView* pView,
                              UINT nFlags, CPoint point);

        virtual void Serialize(CArchive& ar);   //Handle saving and reading from file issues


private:
        unsigned Idx;

#ifdef _DEBUG
        void Dump(CDumpContext &dc) const;
#endif
};
```

CntNd2D.h

```
class CntNd2D:public FEGrObj
{
public:
        DECLARE_SERIAL(CntNd2D)
        //Constructors
        CntNd2D ( double X=0.0, double Y=0.0,
                        class CntctSrf2D* pSurf=NULL);
        CntNd2D(const CntNd2D& Cont);

        //Address of the connected surface
        CntctSrf2D* pSrf;

        //Fixed coordinates of contact node
        double X, Y;

        static GLint m_nDisplySz;

        static CFont m_NdFnt;
        static BOOL m_bVwCntNds; //View Contact Nodes

        void operator =  (const CntNd2D &Cont);
        BOOL    operator == (const CntNd2D &N)const;
        BOOL    operator != (const CntNd2D &N)const;

        //Function to handle drawing issues of contact node
        void Draw(CFEMGLView* pView);

        //Functions to handle mouse events for contact node
        void OnMouseMove   ( CFEMGLView* pView,
                             UINT nFlags, CPoint point);
        void OnLButtonDown( CFEMGLView* pView,
                             UINT nFlags, CPoint point);
        void OnLButtonUp    ( CFEMGLView* pView,
                             UINT nFlags, CPoint point);

        virtual ~CntNd2D(){};    //Destructor
        virtual void Serialize(CArchive& ar);

#ifdef _DEBUG
        void Dump(CDumpContext &dc) const;
#endif
};
```

Element2D.h

```
class El2D:public FEGrObj
{
public:
        static ofstream fout;
        El2D (  class Obj2D *Ob=NULL, //Default Constructor
         unsigned Idx=0,
         double thck=1.0,
         unsigned short MtIdx=0);

        El2D (const El2D& El);    //Copy constructor

        Obj2D* Obj;       //Pointer to connected domain

        virtual double Dimension(void)=0;
        virtual void operator =(const El2D &N);
        virtual bool operator ==(const El2D &N);
        virtual bool operator != (const El2D &N);

        //Liner stiffness matrix of element for both linear anaysis,
        //total and updated lagrangian formulations.
        virtual matrix K_Lin()=0;
        //Tangential stiffnes matrix of element for Newton solution
        //style for both Total & Updated Lagrange formulations
        virtual matrix K_NL()=0;

        //Calculate Body Loading
        virtual colvec Fb()=0;

        //Calculate load from initial displacement effect.
        //Used for residual calculation.
        virtual colvec F_DisplEff  ( unsigned short m_nCurInc=1,
                                     unsigned short m_nRefInc=0)=0;

        //Get local displacements and incremental from connected domain.
        virtual colvec   U_Loc(unsigned inc=0);
        virtual colvec D_U_Loc(void);

        vector<struct NdIdxing> NdLst; //List of connected nodes.

        virtual void IdxtoPtr(void);
        virtual void PtrtoIdx(void);

        unsigned short m_nMtIdx;

        vector<class GsPtData2D> GsPt;
        vector<class GsPtData2D> RedGs;

        double Thick;    //Thickness of Element
        double m_dRho;  //Mass Density of Element (kg/m3)
```

```
        //Internal body loading per unit mass in X(N/m3)
         double m_dFx;
        //Internal body loading per unit mass in Y(N/m3)
        double m_dFy;

        virtual void OnMouseMove    ( CFEMGLView* pView,
                                      UINT nFlags, CPoint point);
        virtual void OnLButtonDown ( CFEMGLView* pView,
                                      UINT nFlags, CPoint point);
        virtual void OnLButtonUp     ( CFEMGLView* pView,
                                      UINT nFlags, CPoint point);

        static unsigned m_nElTxtSz;

        virtual unsigned GetIdx();
        virtual void SetIdx(unsigned i);

        virtual void InitGs(void)=0;

        virtual ~El2D();   //Destructor
        unsigned short GetType(){return TYPE;};
protected:
        virtual void Serialize(CArchive& ar);

        unsigned m_nIdx;
        unsigned short m_nIntOrd;
        unsigned short m_nMinNdNumbr;
        unsigned short m_nMaxNdNumbr;
        unsigned short TYPE;
#ifdef _DEBUG
        void Dump(CDumpContext &dc) const;
#endif
};
```

Rct2D.h

```
class Rct2D:public El2D
{
        public:
        DECLARE_SERIAL(Rct2D)
        //Constructors
        Rct2D  ( class Obj2D *Ob=NULL, unsigned Idx=0,
                   double thick=1.0,unsigned short MtIdx=0);
        Rct2D (const Rct2D& El);
        friend matrix RctBase_2D ( double ksi, double eta,
                                   const vector<unsigned int> LocIdx);

        //Function for calculating linear stiffness matrix for rectangle
        matrix K_Lin();
        //Function for calculating non-linear stiffness matrix for rectangle
        matrix K_NL();
        colvec Fb  (); //Calculate Body Loading
        //Calculate load from displacement effect for residual calculation
        colvec F_DisplEff( unsigned short m_nCurInc=1,  unsigned short m_nRefInc=0);

        double Dimension();  //Area of rectangle
        //Function to handle mouse move message for rectangle
        virtual void OnMouseMove ( CFEMGLView* pView,
                UINT nFlags, CPoint point);

        //Function to handle drawing for rectangle.
        void Draw(CFEMGLView* pView);
        void InitGs(void);
        virtual void Serialize(CArchive& ar);
        virtual ~Rct2D(); //Destructor for rectangle
private:
        unsigned short m_nInt_x, m_nInt_y;
#ifdef _DEBUG
        void Dump(CDumpContext &dc) const;
#endif
};
```

Tri2D.h

```
class Tri2D:public El2D
{
public:
        DECLARE_SERIAL(Tri2D)
        //Constructors
        Tri2D   ( class Obj2D *Ob=NULL, unsigned Idx=0,
                  double thick=1.0,unsigned short MtIdx=0);
        Tri2D (const Tri2D& El);      //Copy constructor

        friend matrix TriBase_2D (  double ksi,  double eta,
                                    const vector<unsigned int> LocIdx);

        //Function for calculating linear stiffness matrix for triangle
        matrix K_Lin();
        //Function for calculating non-linear stiffness matrix for triangle
        matrix K_NL();
        colvec Fb();  //Calculate Body Loading

        //Calculate load from displacement effect for residual calculation
        colvec F_DisplEff( unsigned short m_nCurInc=1,
            unsigned short m_nRefInc=0)  ;
        double Dimension();  //Area of triangle
        //Function to handle mouse move message for rectangle
        void OnMouseMove ( CFEMGLView* pView,
                           UINT nFlags, CPoint point);
        //Function to handle drawing for triangle.
        void Draw(CFEMGLView* pView);
        void InitGs(void);
        virtual void Serialize(CArchive& ar);
        virtual ~Tri2D();  //Destructor for triangle
#ifdef _DEBUG
        void Dump(CDumpContext &dc) const;
#endif
};
```

CntctSrf2D.h

```
class CntctSrf2D:public FEGrObj
{
public:
        //Constructors
        CntctSrf2D(void);
        CntctSrf2D(const CntctSrf2D & Cont);
        CntctSrf2D(CntctSrf2D* const Cont);

        //List of contact nodes
        vector<class CntNd2D*> NdList;

        COLORREF Node_Clr;    //Contact node color
        COLORREF SlctNd_Clr;    //Selected contact node color

        COLORREF Surf_Clr;   //Contact surface color
        COLORREF SlctSrf_Clr;    //Selected contact surface color

        static BOOL  m_bVwCntSrf;
        static BOOL  m_bClosedCnt;

        virtual bool DeleteNode(CntNd2D* pNode);
        virtual void CalcParam(FENd2D* pNd)=0;

        //Find nearest contact node to a FEM node.
        virtual unsigned Nearest  (FENd2D* const pNd);

        //Pure virtual function to calculate nearest
        //parametric point to a FEM node.
        virtual void Ksi_bar    (FENd2D* pNd)=0;

        //Saving and reading of surface specific data from the binary file
        virtual void Serialize(CArchive& ar);

        //Functions to handle mouse events
        void OnLButtonDown  ( CFEMGLView* pView,
                              UINT nFlags, CPoint point);
        void OnLButtonUp      ( CFEMGLView* pView,
                              UINT nFlags, CPoint point);
        void OnMouseMove    ( CFEMGLView* pView,
                              UINT nFlags, CPoint point);

        virtual  ~CntctSrf2D();
        unsigned short GetType(){return TYPE;};

protected:

        //Used for drawing purpose!
        virtual void CalcParam  (CntParamVals*)=0;
        virtual unsigned Nearest  (const double x, const double y);
        virtual double Ksi_bar(const double x, const double y)=0;
```

```
        //Stores type of contact surface (Bernstein, Hermite, etc.)
        unsigned short TYPE;
#ifdef _DEBUG
        virtual void Dump(CDumpContext &dc) const;
#endif
};
```

C1Hermit.h

```
class C1Hermit :public CntctSrf2D
{
public:
        DECLARE_SERIAL(C1Hermit)
        C1Hermit(void);
        C1Hermit(const C1Hermit & Cont);
        C1Hermit(CntctSrf2D*  const Cont);

        //Calculate nearest contact node to a FEM node.
        unsigned Nearest  (FENd2D* const pNd);
        //Calculate nearest parametric point to a FEM node.
        void Ksi_bar    (FENd2D* pNd);

        void CalcParam(FENd2D* pNd);

        //Find nearest contact node to point.
        unsigned Nearest  (const double x, const double y);
        //Calculate nearest parametric point to point.
        double Ksi_bar     (const double x, const double y);

        void CalcParam(CntParamVals*);

        //Handle drawing issues of Hermit Bezier Curve
        void Draw(CFEMGLView* pView);

        //Handle mouse events for hermite surface
        void OnLButtonDown  ( CFEMGLView* pView,
                              UINT nFlags, CPoint point);
        void OnLButtonUp       ( CFEMGLView* pView,
                              UINT nFlags, CPoint point);
        void OnMouseMove     ( CFEMGLView* pView,
                              UINT nFlags, CPoint point);

        //Handle saving and reading of Hermite Surface
        virtual void Serialize(CArchive& ar);

        ~C1Hermit(void);  //Destructor

#ifdef _DEBUG
        virtual void Dump(CDumpContext &dc) const;
#endif
};
```

```
class C1Bernstein :public CntctSrf2D
{
public:
        DECLARE_SERIAL(C1Bernstein)

        //Constructors
        C1Bernstein(void);
        C1Bernstein(const C1Bernstein & Cont);
        C1Bernstein(CntctSrf2D*  const Cont);

        //Calculate nearest contact node to a FEM node.
        unsigned Nearest  (FENd2D* const pNd);
        //Calculate nearest parametric point to a FEM node.
        void Ksi_bar    (FENd2D* pNd);

        void CalcParam(CntParamVals* pPrm);

        //Calculate nearest contact node point to point.
        unsigned Nearest  (const double x, const double y);

        //Calculate nearest parametric point to point.
        double Ksi_bar  (const double x, const double y);

        void CalcParam(FENd2D* pNd);

        //Function to handle drawing issues
        void Draw(CFEMGLView* pView);

        //Functions to handle mouse events for Bernstein surface
        virtual void OnLButtonDown(  CFEMGLView* pView,
                 UINT nFlags, CPoint point);
        virtual void OnLButtonUp ( CFEMGLView* pView,
                UINT nFlags, CPoint point);
        virtual void OnMouseMove ( CFEMGLView* pView,
                UINT nFlags, CPoint point);

        virtual void Serialize(CArchive& ar);
        ~C1Bernstein(void);
private:
#ifdef _DEBUG
        virtual void Dump(CDumpContext &dc) const;
#endif
};
```

GsPt2D.h

```
class GsPt2D: public FEGrObj
{
public:
        DECLARE_SERIAL(GsPt2D)

        //Constructors
        GsPt2D  ( El2D *El=NULL,
            double ksi=0.0,
            double eta=0.0,
            double Coeff=2.0);
        GsPt2D  ( const GsPt2D &GsPt  );

        bool Init ( El2D *El, double ksi,
            double eta,  double Coeff);

        //Jackobian matrix (Default referred to initial state)
        matrix J  ( unsigned short m_nInc=0);
        double detJ ( unsigned short m_nInc=0);

        //Constitutive matrix
        matrix C ( unsigned short m_nInc=0);

        //For calculating derivatives (Default referred to initial state
        matrix B(unsigned short m_nInc=0, bool m_bCurent=FALSE);
        //Always referred to initial state (Calculated only for Total Lagrangian Solution)
        matrix BL1(unsigned short m_nInc =1);

        //For calculating Non_Lin part of stiffness matrix.
        //For TL Approach should always refer to initial state (=0)
        //For UL Approach should refer to curent state.
        matrix BNL( unsigned short m_nInc=1,
          bool m_bCurent=FALSE  );
        //For calculating mass matrix or internal body forces
        matrix H();

        //Calculates strains.
        //For TL approach Green's Strain.
        //For UL approach Almansi's strain
        matrix E ( unsigned short m_nCurInc=1,
                    BOOL m_bVForm=TRUE,
         unsigned short m_nRefInc=0);

        //Cauchy Stress
        colvec CST(unsigned short m_nCurInc=1,unsigned short m_nRefInc=0);

        //Deformation Gradient
        matrix F(unsigned short m_nCurInc=1, unsigned short m_nRefInc=0);  //Deformation
        //Determinant of deformation gradient matrix
        double detF(unsigned short m_nCurInc=1, unsigned short m_nRefInc=0);

        //Second Piola Kirchoff Stress
        PKST2(unsigned short m_nCurInc=1,  unsigned short m_nRefInc=0);
```

```
        // Set ksi and eta local coordinates
        void SetLocal(double ksi, double eta, unsigned short m_nInc);
        //Calculate Global coordinates
        void To_Glob (unsigned short m_nInc = 0);

        void operator =  (const GsPt2D &GsPt);

        BOOL    operator == (const GsPt2D &N)const;
        BOOL    operator != (const GsPt2D &N)const;


        ~GsPt2D(){};    //Destructor

        virtual void Serialize(CArchive& ar);
        double Coeff;    //Effect pf GsPt on Element Integration

        El2D* m_pEl;
        static GLint m_nDisplySz;
        double  x,  y;  //Coords of GsPt on Global Elem in curent coordinates
        double  X, Y; //Coords of GsPt on Global Elem in reference coordinates
        double Displ_X, Displ_Y;

        virtual void Draw(CFEMGLView* pView);

        //Functions to handle mouse events
        virtual void OnLButtonDown(CFEMGLView* pView, UINT nFlags, CPoint point);
        virtual void OnLButtonUp(CFEMGLView* pView, UINT nFlags, CPoint point);
        virtual void OnMouseMove(CFEMGLView* pView, UINT nFlags, CPoint point);

private:
        double ksi, eta;//Coords of GsPt on Master Elem

public:
#ifdef _DEBUG
        void Dump(CDumpContext &dc) const;
#endif
};
```

Obj2D.h

```
class Obj2D: public CObject
{
public:
	DECLARE_SERIAL(Obj2D)

	//Constructors
	Obj2D (void);
	Obj2D (const Obj2D & Ob);

	friend class std::basic_ostream;
	friend class std::basic_istream;

	//For output to file
	static ofstream fout;

	//*******************************
	//Solvers

	//This function also called from Newton solution for an initial
	//solution to Newton iteration.
	//When making incremental solution for NonLinear solution
	//option, it does not make incrementation.
	//Only makes a single solution at the curent incrementation.
	virtual BOOL Lin_Solve    (const unsigned short inc);

	//This function handles both Total Lagrangian and
	//Updated Lagrangian solutions.
	virtual BOOL Nwt_Solve  (void);
	//*************************************
	double TotFx, TotFy;
	//*************************************
	//Containers:
	std::vector < class FENd2D*>GlNodes; //Node Container
	std::vector < class El2D*  >Elements;  //Element Container
	std::vector<class Lin_Mat> Mater;    //Material Container
	//*****************************************

	static Solution_Parameters SolnPrm;  //Solution Parameters
	static Results_ParametersResPrm; //Results Parameters

	//Find maximum number of free nodes.
	//(Assembler() must be called first)
	unsigned maxff()const;
	//Find maximum number of restrained nodes.
	//(Assembler() must be called first)
	unsigned maxrr()const;
	//Set matrix Dof for assemblage
	virtual int Assembler();

	//Assemble global stiffness matrix
```

```
virtual int Assemble_K(  matrix & Kff,
      matrix & Kfr,
      matrix & Krr,
      BOOL m_bLinear=1);

//Assembledisplacement vector at the incremental time
virtual Assemble_Ur(colvec &Ur, unsigned m_nInc=1) const;

//Assemble free node force vector at the incremental time
virtual Assemble_Ff (colvec &Ff, unsigned m_nInc=1) const;

//Calculates internal element loads at the increment
virtual Assemble_El_Fb( colvec &Ff,  colvec &Fr,
      unsigned m_nInc=1) const;

//Calculates loads from displacement effects from elements at the increment;
virtual Assemble_El_Fd  ( colvec &Ff,  colvec &Fr,
                              unsigned m_nCurInc=1,  unsigned m_nRefInc=0) const;

virtual void ResetNdIdx();
virtual void ResetElIdx();

//Reset indices after node iterator
virtual void ResetNdIdx(FNdit);
//Reset indices after element iterator
virtual void ResetElIdx(FElit);

//**************************
//Coloring Parameters
//Node Coloring
COLORREF Node_Clr;
COLORREF SlcNd_Clr;

COLORREF Load_Clr;
COLORREF Supp_Clr;
COLORREF Reac_Clr;
COLORREF Sprng_Clr;

//Element Coloring
COLORREF Element_Clr;
COLORREF Element_Slctd_Clr;
COLORREF Element_Frm_Clr;

//Element VwPrms
static BOOL m_bVwElEdges;
static BOOL m_bVwElIdx;
static BOOL m_bVwGsPts;
//************************

 colvec Ur, D_Ur, Uf, D_Uf;

//*******************************
//Real display items
vector<double> Disp_Real_Vals;
//To display color values
vector <COLORREF> Disp_Col_Val;
//*********************************
CMatrix<unsigned> Dof;//Stores Degrees of Freedoms
```

```
        BOOL m_bAnalysed; //For checking state of analysis

        void DeleteElem(const unsigned ElIdx);
        bool DeleteElem(El2D* pElem);
        void DeleteNode(const unsigned NdIdx);
        bool DeleteNode(FENd2D* pNode);

        //Set boundary nodes next boundary node index table
        void SetBoundary(void);

        //Set global U, V, Fx, Fy for curent increment number
        bool SetColorTable();

        int Construct;
        virtual ~Obj2D();

        virtual void Serialize(CArchive& ar);

protected:
        //Penalty solution returns number of iterations
        unsigned PenaltyContact(  const matrix & m_Kff,
              const matrix & m_Kfr,
              const matrix & m_Krr,
              const colvec & m_Ur,
              const colvec & m_Ff);

        //Lagrange multiplier solution returns number of iterations
        unsigned LagMultContact(const matrix & m_Kff,
              const matrix & m_Kfr,
              const matrix & m_Krr,
              const colvec & m_Ur,
              const colvec & m_Ff);

#ifdef _DEBUG
        void Dump(CDumpContext &dc) const;
#endif
};
```