SYMBOLIC-NUMERICAL OBJECT-ORIENTED FINITE ELEMENT PROGRAMMING

BY

PISITH SAM

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF SCIENCE (ENGINEERING AND TECHNOLOGY)
SIRINDHORN INTERNATIONAL INSTITUTE OF TECHNOLOGY
THAMMASAT UNIVERSITY
ACADEMIC YEAR 2015

SYMBOLIC-NUMERICAL OBJECT-ORIENTED FINITE ELEMENT PROGRAMMING

BY

PISITH SAM

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF SCIENCE (ENGINEERING AND TECHNOLOGY)
SIRINDHORN INTERNATIONAL INSTITUTE OF TECHNOLOGY
THAMMASAT UNIVERSITY
ACADEMIC YEAR 2015

SYMBOLIC-NUMERICAL OBJECT-ORIENTED FINITE ELEMENT PROGRAMMING

A Thesis Presented

By
PISITH SAM

Submitted to
Sirindhorn International Institute of Technology
Thammasat University
In partial fulfillment of the requirements for the degree of
MASTER OF SCIENCE (ENGINEERING AND TECHNOLOGY)

Approved as to style and content by

Advisor and Chairperson of Thesis Committee [illegible]
(Prof. Pruettha Nanakorn, D.Eng.)

Committee Member and
Chairperson of Examination Committee [illegible]
(Prof. Somnuk Tangtermsirikul, D.Eng.)

Committee Member [illegible]
(Assoc. Prof. Winyu Rattanapitikon, D.Eng.)

Committee Member Ekachai
(Asst. Prof. Ekachai Chaichanasiri, Ph.D.)

DECEMBER 2015

Abstract

SYMBOLIC-NUMERICAL OBJECT-ORIENTED FINITE ELEMENT PROGRAMMING

by

PISITH SAM

B.Eng. in Civil Engineering, Institute of Technology of Cambodia, 2013

There are times when closed-form analysis of solids and structures is needed. When only a structural member is considered, it is certainly possible to perform this type of analysis by hand. However, when a structure with several members is to be considered, manual calculation becomes practically impossible even when closed-form solutions are theoretically obtainable. A finite element (FE) program that can symbolically analyze elasticity problems of solids and structures can therefore be quite useful. Unfortunately, symbolic computations are usually difficult to perform when problems are complex. In these cases, numerical computations are still necessary. This study presents an object-oriented FE program that can perform both symbolic and numerical computations. The program is implemented in MATLAB. The object-oriented programming (OOP) technique is used to enhance the maintainability, extendibility, and reusability of the program. The proposed program is capable of performing symbolic and numerical finite element analysis depending on the input it receives. The obtained program is tested using some engineering problems in order to demonstrate its usefulness. The results obtained from the program are found to be satisfactory.

Keywords: Finite element programming, Closed-form analysis, Linear elasticity, Symbolic computation, Numerical computation, Symbolic-numerical computation, Object-oriented programming.

Acknowledgements

First of all, I would like to thank my advisor, Prof. Dr. Pruettha Nanakorn, for his precious guidance, encouragements, and opinions throughout my master’s degree study. It is a great honor for me that I can be involved with his research teams to study new knowledge and gain new experiences.

I deeply thank the ASEAN University Network/the Southeast Asia Engineering Education Development Network (AUN/SEED-Net) for providing me with the scholarship. Without this scholarship, I would not be able to pursue my master’s degree study in Thailand.

I would never forget to thank my sending institution, Institute of Technology of Cambodia (ITC), which has endorsed me for this scholarship. That was a good place where I started my engineering life.

I would also like to thank my host institution, Sirindhorn International Institute of Technology (SIIT), which allows me to attend the school, and offers me a half of the financial supports under the scholarship of AUN/SEED-Net program.

A special thank to my senoirs in our research team such as Mr. Wasuwat Petprakob, Mr. Kasem Theerakittayakorn, and Mr. Naga Venkata Chandrasekhar Koralla for taking time to explain me about object-oriented programming, cellular solids, and the finite element method.

To my colleagues and friends in the reseach laboratory of the School of Civil Engineering and Technology in SIIT, I thank them for their companionship and for providing a pleasurable and friendly working environment.

I would like to thank the professors in ITC who helped me during my bachelor’s degree study, and especially to those who recommended me to pursue my graduate study.

My acknowledgment would finally mention my family. I wish to express my gratitude to my parents for their finalcial supports and encourgagements untill the end of my study.

Table of Contents

List of Figures

List of Tables

Chapter 1
Introduction

1.1 General

The finite element method (FEM) is an efficient method for solving differential equations. As most physical problems can be expressed as differential equations, FEM has become a power tool for solving these problems. Finite element (FE) programs have been developed with many programming languages by procedural and object-oriented programming (OOP) paradigms. The procedural programming paradigm is a simple kind of programming technique that focuses on separating computational tasks into many systematical procedures or subroutines that can be reused. Although this procedural technique is simple and easy to program, it has some remarkable drawbacks, such as difficulties to relate with real world objects, maintenance of the codes, and management of data memories, and securities. These drawbacks are more pronounced when programs become large and complex. To deal with these data-related problems, the OOP paradigm can be used. The OOP technique treats programs as communications and interacts between objects. Programs that use OOP can be considered as a collection of interactive objects which consist of data fields associated with procedures known as methods. The OOP technique can effectively improve the maintainability, extendibility, and reusability of programs.

As aforementioned, FE programs have been written in various programming languages. Some of the popular languages for implementing FEM include FORTRAN, C, C++, C#, Java, Mathematica, Maple, MATLAB, Python, and Smalltalk. These programming languages may have different capacities and employ different programming paradigms. For instance, FORTRAN relies heavily on the procedural programming technique although some OOP capacities have been recently added. In addition, C, C++, C# and Java allow only numerical computations while Mathematica, Maple, MATLAB, Python, and Smalltalk can perform both numerical and symbolic computations. Oftentimes, symbolic computations are necessary and desirable in engineering calculations. For example, symbolic computations offer closed-form solutions for engineering problems as practical and ready-to-use formulas. However, symbolic computations are usually difficult to perform. For many complex problems, closed-form solutions sometimes do not exist at all.

Most of the time, programming languages that do not support symbolic computations are used to implement FEM such as FORTRAN, C, C++, C#, and Java. This is because of the fact that real problems are usually large and complex; therefore, they can be managed only by efficient numerical programs. However, programming languages that allow both numerical and symbolic computations have been significantly improved recently. As a result, it is now possible to develop FE programs that are capable of handling both symbolic and numerical computations. These programs are quite useful since they can give symbolic solutions when it is possible, and give numerical solutions when symbolic solutions are not possible or not required.

Closed-form analysis of simple structural problems can be solved manually by engineers. Those structures can be, for example, simple trusses and frames. Closed-form analysis is a kind of mathematical analysis that is used to determine expressions of closed-form solutions which will be later used as formulas. In finite element analysis (FEA) of structural problems, closed-form analysis can be used to find symbolic expressions of element stiffness matrices, unknown displacements, strains, stresses, and strain energy. Closed-form solutions of element stiffness have been studied and considered in many research works (Yew et al., 1995, Jiang and Wang, 2008, Shiakolas et al., 1994). Closed-form stiffness can be used to speed up FEA processes significantly since the formation of stiffness matrices in real time is no longer required. There are also some research works that deal with closed-form analysis of strain energy. For example, in the works by Zhang et al. (2007), Dai and Zhang (2009), and Wang et al. (2008), the closed forms of strain energy of unit cells of periodic cellular solids are used to determine the effective elastic properties of the solids.

1.2 Statement of the problem

FE programs are mostly written using the procedural programming paradigm, which in general cannot handle complex data structures well. As a result, these codes contain many fragmented data variables, which are accessed throughout the programs. This causes many difficulties in modifying and extending the existing codes for new uses, models, and solution procedures. To deal with these problems, the OOP paradigm becomes an interest.

Under the OOP paradigm, software is organized as a collection of discrete and distinguishable objects that incorporate both data structures and behavior. The key idea of this approach is to consider tasks in software as communications between and operations within various objects. As aforementioned, the concept greatly improves maintainability, extendibility, and reusability of software. There exist large quantities of research works that are concerned with the development of object-oriented FE programs (Forde et al., 1990, Abdalla and Yoon, 1992, Mackie, 1998, Archer et al., 1999, Mackie, 2002, Mackie, 2007). When the OOP technique is used in implementation of FEM, the process of FEA must be thought of as interactions between various entities. These entities can be classified into various classes of objects whose data and behavior are clearly defined. Common examples of these classes include nodes, elements, and materials. As seeing each entity as an object can be achieved quite naturally by programmers, programming FEM using the OOP paradigm becomes a smooth and natural process.

Most FE programs are only implemented for numerical computations. In fact, it is generally sufficient to solve engineering problems numerically. However, there are some problems that symbolic computations are necessary or preferred (Shiakolas et al., 1994, Pavlovic, 2003, Shiakolas et al., 1993). By using symbolic computations in FE programs, closed-form solutions of FE problems can be obtained and can be later used as formulas. Thus, developing a symbolic FE program that can handle both symbolic and numerical FEA can be beneficial. In order to develop a symbolic FE program, a powerful programming language that can manipulate symbolic computations is necessary. In addition, in order to utilize the OOP paradigm, the programming language must also have the OOP capabilities. One of the powerful programming languages that allow symbolic and numerical computations and have the OOP capabilities is MATLAB.

1.3 Objectives of the study

The main objective of this study is to develop a symbolic-numerical object-oriented FE program. To achieve this objective, the following research goals are warranted:

- To develop a symbolic-numerical object-oriented FE program by using the MATLAB programming language.
- To demonstrate the usefulness of the obtained program by using it to symbolically solve some engineering problems whose closed-form solutions are useful and preferable to numerical solutions.

1.4 Scope of the study

As the objectives of study are clearly identified, the scope of this study is set to specify the limitations and the sub-purposes of this study. The scope of the study is as follows:

- The obtained program solves only linear static problems of solids and structures.
- The analysis results from the program include displacements, strains, stresses, and strain energy.
- For finite element problems that do not possess closed-form solutions, the program provides only numerical solutions.
- Due to the limitation of MATLAB, it may not be possible to obtain closed-form solutions of complex FE problems.
- The input of the program is in the text format.
- The output of the program is in both text format and graphical format, when required.

Chapter 2
Literature Review

2.1 General

According to a bibliography by Mackerle (2000), there are many research works on object-oriented FE programming. Most of the programs in these research works are developed for numerical computations. Only a small number of them are implemented for symbolic computations. Mackerle (2000) stated that symbolic computations are still the difficult tasks to perform due to the lack of symbolic manipulation efficiencies in most of the conventional programming languages. Pavlovic (2003) discussed some available programming languages that can be used to perform symbolic FE programming such as Maple and Mathematica.

2.2 Numerical FE programs

Forde et al. (1990) mentioned that the capability of the object-oriented FE approach to create expandable application frameworks is probably singularly responsible for its popularity. In their work, different shape function classes are used to define different types of shape function. Each shape function class provides the strain-displacement matrix, also commonly known as the **B** matrix, determined from its designated shape functions. The shape function classes are individually employed in different element classes to create different types of element.

Abdalla and Yoon (1992) presented an object-oriented approach to integrate both FE and graphical application programs. The objective of this work is to develop a general data-translation facility by using C++ for translating data among FE and graphics based programs. To transform the data form a graphical program, AutoCAD, an initial graphical exchange standard (IGES), which is a standard format supported by the drawing applications, is used. Finally, they investigated the effectiveness and viability of object-oriented approaches to integrate programs with different data formats.

Kwon and Bang (2000) introduced a numerical FE program by using the MATLAB language. The main purpose of their work is to demonstrate how to program FEM in

MATLAB. Moreover, they provided examples of the conventional FE codes for structural and mechanical problems.

Patzák and Bittnar (2001) developed an object-oriented FE program by using C++ programming language. Their program covers linear, nonlinear, static, and dynamic problems. In addition, the concepts of a kernel structure are presented in order to illustrate the environments of their program.

Akin and Singh (2002) used the OOP paradigm in the implementation of the P-adaptive method in FEM. In their work, they introduced the advantages of the OOP paradigm in FORTRAN 90 and FORTRAN 95 that can be used to improve the flexibility, maintainability and extensibility of the program. Moreover, the error estimator called "P-adaptive method" has been developed within the program to improve the solutions in FEM.

Martha and Parente Jr (2002) developed an object-oriented framework for a FE program called "FEMOOP." They presented some principle views of the consumer-supplier techniques in order to explain the OOP and conventional programming paradigms. In their work, the computational tasks of analysis problems are grouped into three distinct levels, namely the structural level, element level, and integration level. The structural level deals with the algorithms used to analyze the problems of different types, such as linear or nonlinear problems. The element level deals with computing the element vectors and matrices, such as force vectors and stiffness matrix. The integration level used to handle and compute the strain and stress vectors.

Heng and Mackie (2009) presented a design pattern of an object-oriented FE program. In their study, the program is classified into two specific subsystems, i.e. modeling and analyzing. The modeling subsystem is used to handle the classes of node, element, boundary condition, and material properties of the program while the analyzing subsystem is used to manage the solvers for solving the equations in FEM. A unified modeling language (UML) is used to present the concepts of the program in order that the design can be used with different programming languages. Finally, the graphical user interface (GUI) is introduced to the program.

Piedade Neto et al. (2013) proposed an object-oriented class design for a generalized FEM (GFEM) by using the Python language. GFEM is based on generating new shape functions called "enriched shape functions," which are obtained by multiplying a partition of unit shape functions and special functions. As a result, new classes are created and inserted into the conventional FE program to create GFEM. The program is successfully created to numerically solve linear elastic mechanical problems, and future development for nonlinear problems is also discussed.

Alves et al. (2013) also developed an object-oriented program for GFEM. In their study, the program is implemented in Java language, which is different from the previous programming language used by Piedade Neto et al. (2013). The new program can be used to numerically solve linear, nonlinear, static, and dynamic structural problems.

Rahman and Valdman (2013) depicted a fast technique for FE programming, called array operations, to numerically assemble the element stiffness matrices for two-and three-dimensional elements by using MATLAB.

Zander et al. (2014) presented an object-oriented toolbox for a finite-cell method in MATLAB called "FCMLab." The method is used to apply mesh generations by using P-adaptive and H-adaptive refinements over fictitious and real physical domains. Due to various capacities of the FCMLab toolbox, it allows new algorithms to be easily and quickly added without affecting other codes in the program.

2.3 Symbolic FE programs

Yew et al. (1995) wrote a symbolic FE program for analysis of 2D beam structures by using Mathematica language. In their work, the closed-form integration of element stiffness matrices, which are derived from the mixed-formulation functional based on the Hellinger-Reissner principle, is employed. Unlike the conventional FEM based on the displacement functional, whose displacement fields are the primary variables, FEM based on the Hellinger-Reissner functional, whose stresses and displacements are the unknown fields, can give results that are more accurate. However, this method can be applied only in linear elasticity when the complimentary strain energy is equivalent to the strain energy.

Jiang and Wang (2008) introduced a symbolic FE program in plasticity written in Mathematica language. In their study, the closed-form expressions of stiffness matrices of the 2D plane strain elements in plasticity are determined. The Newton-Raphson method is used to solve the nonlinear equations of the problems.

Eyheramendy and Zimmermann published many research works about symbolic OOP using Smalltalk programming language. The first publication has intension to determine the symbolic integrations of the element stiffness matrices for linear elastodynamic problems (Zimmermann and Eyheramendy, 1996). The second publication is about a symbolic object-oriented FE program for linear elastodynamic problems as well (Eyheramendy and Zimmermann, 1996). In this publication, the detailed descriptions of symbolic derivations and automatic principles are presented. Moreover, the symbolic computations are derived from the initial-boundary-value problems into matrix form in a quasi-automatic environment. The third publication aims to improve capacities and automatic environments of the second publication (Eyheramendy and Zimmermann, 1996). In the third publication, the program is tested using several problems, such as thermal, elastodynamic, dynamic uniaxial bar problems, and Navier-Stoke flow problems. Some weak points relating to the performances of the symbolic computations of shape functions, in which the automatic environments for symbolic computations are still limited, are pointed out. The fourth publication accomplishes the automatic environments of the program that can combine symbolic mathematical manipulations, symbolic computations, and automatic programming (Eyheramendy and Zimmermann, 1998). The last publication aims to improve the capacities of the previous program by introducing new concepts of nonlinear analysis (Eyheramendy and Zimmermann, 2001).

2.4 Symbolic-numerical FE programs

Cheng (1991) developed a symbolic FE program for heat transfer problems by using hybrid techniques. The techniques are implemented by both Mathematica and C languages. The objective of his work is to determine the symbolic closed-form solutions of the Nusselt number, which is a ratio of convection and conduction in heat transfer problems. In his work, the symbolic derivations are first implemented in Mathematica in order to identify all the coefficients for systems of polynomials. Due to the available

Gauss elimination algorithm in C, the proposed hybrid techniques are used to link the symbolic computations in Mathematica and the numerical computations in C.

Shiakolas et al. (1993) presented a FE program for closed-form analysis of a Zienkiewicz-Zhu (ZZ) error estimator for linear and quadratic strain tetrahedron elements. The ZZ error estimator is firstly suggested by Zienkiewicz and Zhu, and it is different from adaptive mesh refinement methods. The basic ideas of this method are to introduce an error of computed stress, which is obtained from the difference between the smoothed stress distribution and the computed nodal stress, into the errors of the energy-norm formulations. In addition, Shiakolas et al. (1994) determined the closed-form expressions of element stiffness matrices for these elements as well. In their work, the symbolic derivations are implemented in Mathematica. Finally, the numerical results are obtained by FORTRAN via its existing Gauss numerical algorithms.

Tummarakota and Lieh (1996) proposed a symbolic FE model of structural systems. The purpose of their work is to determine the symbolic equations of motions for 2D multibody systems from the Lagrange's method. Moreover, the symbolic derivations are written in Maple programming language. As the systems of equations are obtained, a well-known numerical method "Runge-Kutta-Fehlberg" is then used. Finally, the numerical computations are implemented in FORTRAN.

Cameron (1997) presented symbolic computations to evaluate multivariate polynomials in FEM. In his work, the polynomial expressions of 2D and 3D isoparametric elements are considered. To obtain those polynomial expressions, two procedures of Horner's method are presented. One is employed with the loop operations and the other is used with nested bracket operations. As it is difficult to derive the symbolic expressions of the polynomials with the nested implementations, the loop method is preferred. Finally, the symbolic derivations and numerical computations are written in the Maple and C languages, respectively.

Korelc (1997) developed an automatic nonlinear FE program by simultaneous optimizations of the expressions. The purpose of his work is to determine the closed-form expressions of the gradient and Hessian, which are necessary for applications in nonlinear analysis. Moreover, the symbolic derivations are implemented in Mathematica language. However, the symbolic derivations became a difficult task due to the lack of efficiency of `simplify` commands in Mathematica. A stochastic evaluation algorithm is used to effectively simplify the symbolic expressions. Finally, the numerical computations are implemented in FORTRAN.

Lee and Hobbs (1998) presented closed-from analysis of element stiffness matrices for 2D plane stress elements by using mixed formulations based on the Hellinger-Reissner functional. In their work, there is no mention of the programming language used to derive their symbolic computations of the problems. However, the numerical computations are written in FORTRAN.

Zimmermann et al. (1998) presented an integrated environment of an object-oriented FE program by using Smalltalk and C++. The objective of their work is to develop a user-friendly interface for FE developers. The interface consists of a graphical interface, operations for symbolic mathematical derivations, and FE formulations that can handle both symbolic and numerical computations. Moreover, new element classes of some structural problems are also created and added to their previous research works.

Eriksson and Pacoste (1999) introduced a new technique of symbolic computations in a FE program that can be implemented by using symbolic languages such as Maple and Mathematica. In their work, the symbolic derivations of elements stiffness and rotational-transformation matrices for 3D beam and 2D plane stress elements are considered. Moreover, the symbolic computations are presented in both Maple and Mathematica. Finally the numerical results are evaluated in FORTRAN.

Eyheramendy (2000) proposed an object-oriented FE program that employs hybrid symbolic and numerical approaches. As the symbolic derivations in the author's previous work (Zimmermann et al., 1998) are derived from the Galerkin method, the author created new object classes in the existing FE program by using the variational methods. The symbolic derivations and numerical computations are implemented in Smalltalk and C++, respectively.

McCaslin et al. (2012) improved the work by Shiakolas et al. (1994). They aim to determine the closed-form solutions of elements stiffness for tetrahedral elements by increasing the degree of the shape functions. The symbolic derivations are implemented in Mathematica and the numerical computations are implemented in FORTRAN.

Chapter 3
Theoretical Background

3.1 FE formulations

For the finite element method based on a functional of displacements, the total potential energy of an element, whose domain is V_{el} and boundary is S_{el}, can be expressed as

$$\Pi_{el} = \frac{1}{2}\int_{V_{el}} \boldsymbol{\varepsilon}^T \boldsymbol{\sigma} dV - \int_{V_{el}} \mathbf{u}^T \mathbf{b} dV - \int_{S_{el}} \mathbf{u}^T \bar{\mathbf{T}} dS, \tag{1}$$

where $\boldsymbol{\varepsilon}$, $\boldsymbol{\sigma}$, $\mathbf{u}$, $\mathbf{b}$, and $\bar{\mathbf{T}}$ represent the strain, stress, displacement, body force, and prescribed traction vectors, respectively. The displacements in **u** are interpolated from the nodal displacements as

$$\mathbf{u} = \mathbf{N}\mathbf{q}, \tag{2}$$

where $\mathbf{N}$ and $\mathbf{q}$ are the shape function matrix and the nodal displacement vector, respectively.

From the strain-displacement relations, the strain vector can be expressed as

$$\boldsymbol{\varepsilon} = \mathbf{L}\mathbf{N}\mathbf{q} = \mathbf{B}\mathbf{q}. \tag{3}$$

Here, **L** is the strain displacement operator matrix. For linear elasticity, the stress vector **σ** can be written as

$$\boldsymbol{\sigma} = \mathbf{D}\boldsymbol{\varepsilon} = \mathbf{D}\mathbf{B}\mathbf{q}, \tag{4}$$

where **D** is a constitutive matrix.

By substituting the Eqs. (2)-(4) into Eq. (1), the total potential energy can be expressed as

$$\Pi_{el} = \frac{1}{2}\mathbf{q}^T \mathbf{K}_{el} \mathbf{q} - \mathbf{q}^T \mathbf{F}_{el}, \tag{5}$$

where $\mathbf{K}_{el}$ and $\mathbf{F}_{el}$ denote the element stiffness matrix and force vector, respectively. They are expressed as

$$\mathbf{K}_{el} = \int_{V_{el}} \mathbf{B}^T \mathbf{D} \mathbf{B} dV, \tag{6}$$

$$\mathbf{F}_{el} = \int_{V_{el}} \mathbf{N}^T \mathbf{b} dV + \int_{S_{el}} \mathbf{N}^T \bar{\mathbf{T}} dS. \tag{7}$$

By employing the principle of stationary potential energy to the Eq. (5), an element stiffness equation can be obtained as

$$\mathbf{K}_{el}\mathbf{q} = \mathbf{F}_{el}. \tag{8}$$

After all elements are assembled by considering their connectivities, the following equations can be obtained

$$\Pi = \frac{1}{2}\mathbf{Q}^T\mathbf{K}\mathbf{Q} - \mathbf{Q}^T\mathbf{F}, \tag{9}$$

$$\mathbf{K}\mathbf{Q} = \mathbf{F}. \tag{10}$$

Here, Π is the total potential energy of the whole domain. In addition, **K**, **Q** and **F** are the global stiffness matrix, the global displacement vector, and the global force vector, respectively. In FEM, some displacement degrees of freedom and forces are known. As a result, the Eq. (10) can be rearranged into the following form, i.e.

$$\begin{bmatrix} \mathbf{K}_{I,I} & \mathbf{K}_{I,II} \\ \mathbf{K}_{II,I} & \mathbf{K}_{II,II} \end{bmatrix} \begin{Bmatrix} \mathbf{Q}_I \\ \mathbf{Q}_{II} \end{Bmatrix} = \begin{Bmatrix} \mathbf{F}_I \\ \mathbf{F}_{II} \end{Bmatrix}. \tag{11}$$

Note that $\mathbf{Q}_{II}$ contains the prescribed displacements while $\mathbf{F}_I$ contains the prescribed forces. In addition, $\mathbf{Q}_I$ and $\mathbf{F}_{II}$, respectively, contain the unknown displacements and the unknown forces. The unknown displacements in $\mathbf{Q}_I$ can be easily obtained as

$$\mathbf{Q}_I = \mathbf{K}_{I,I}^{-1}(\mathbf{F}_I - \mathbf{K}_{I,II}\mathbf{Q}_{II}). \tag{12}$$

As all displacement degrees of freedom are now known, the strain energy U can be obtained from

$$U = \frac{1}{2}\mathbf{Q}^T\mathbf{K}\mathbf{Q}. \tag{13}$$

3.2 FE programming

3.2.1 OOP concepts

The OOP paradigm is more advantageous than the procedural programming technique. To further understand the OOP concept, Martha and Parente Jr (2002) have introduced a consumer-supplier concept in the OOP paradigm that can help to effectively improve the reusability, maintainability and extendibility of the FE program. The principle concept of OOP is generally based on classes, objects, encapsulation, inheritance, and polymorphism. A class is a blueprint which can represent the characteristics of objects, namely their properties and behavior. A class contains data members and function members of objects. An object is an instance of a class. It means that a class can create many possible objects from its existing blueprint. Encapsulation defines types of specialized access to members of classes. Encapsulation allows the details of the implementations of objects to be hidden from the users. Inheritance allows subclasses or derived classes to be derived from a parent or base class and inherits the characteristics of the base class. Inheritance removes the necessities for repeating the codes that represent the inherited characteristics in the derived classes. This inheritance concept can help improve the extendibility of codes. Polymorphism is a consequence of inheritance that allows the same operations to behave differently in different classes (Glasser, 2009).

3.2.2 Object modeling technique

An object modeling technique (OMT) is a kind of diagram languages used for software modeling and designing. The OMT has been first introduced by Rumbaugh et al. (2004). This technique allows relationships between classes and objects to be summarily and clearly presented. Moreover, the OMT allows different types of entity relationship, such as associations (link), aggregations and compositions (has-a), generalizations (is-a), dependencies, and realizations, to be clearly shown.

Associations describe reference-based relationships which allow objects to comminute with each other. Roles that specify the purposes of relationships can be specified in associations. In addition, each end of an association can have a multiplicity value, which indicates how many objects on one side can relate to other objects on the other side. For example in Fig. 3.1, an `Element` class connects to a `Node` class. In fact, many elements can be connected to many nodes. Hence, the multiplicity values of this relationship can be zero or more, which are represented by darkened circles at the ends of the connection. A multiplicity value can also be expressed as a specific-number representation.

Aggregations and compositions are special kinds of association. They represent types of relationships in which objects are part of other objects in a system. These relationships are "has-a" relationships. For instance, in Fig. 3.1, many elements and many nodes in `Element` and `Node` classes are part of a `FESys` class. An aggregation implies a relationship where an object being used can exist independently of another object using the former object. A composition implies a relationship where an object being used cannot exist independent of another object using the former one. Aggregations and composition are represented by diamond ends at classes that own or use objects of other classes. Diamonds for aggregations are unfilled while those for compositions are filled.

Generalizations and specializations present the inheritance concept where classes are derived from their base classes. They represent “is-a” relationships. For example in Fig. 3.1, the `Element` class is a generalization of `ELine2`, `ETri3`, and other classes represented by the dots. On the contrary, `ELine2`, `ETri3`, and other classes are specializations of the `Element` class. A generalization is represented by a triangle symbol.

Dependencies describe dependent or weaker relationships of objects. For example, changes in one object can affect other objects in a system. Realizations describe implementations of functionalities defined in one class by other classes.

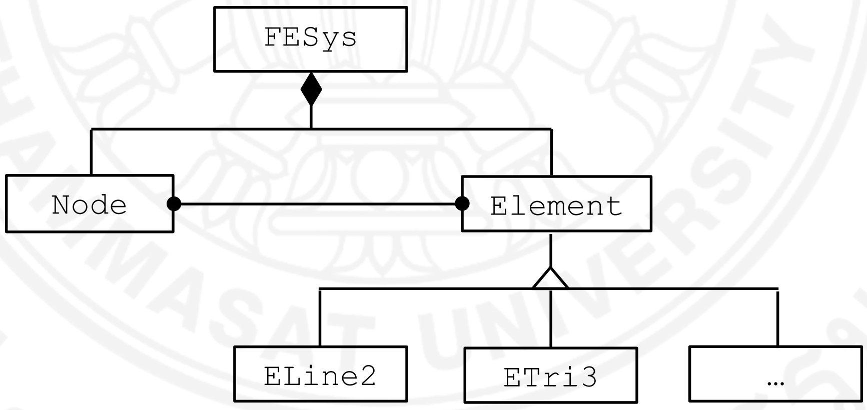


Fig. 3.1 Simple relationship diagram of class entities

Chapter 4
Symbolic-Numerical Object-Oriented Finite Element Programming

4.1 Object-oriented FE programming

According to Heng and Mackie (2009), a good design pattern of an object-oriented FE program should decompose the program into two main subsystems, namely the model and analysis subsystems. In this study, the model subsystem represented by the `Object` class in Fig. 4.1 focuses on defining model classes such as the classes for nodes, boundary conditions, elements, geometry sets, and materials: `Node`, `BounCond`, `Element`, `Geometry` and `Material`. The analysis subsystems in this study include the `Solver` class, which is a generalization of solvers, and the `FESys` class, which is the main class of the proposed symbolic FE system. The two analysis classes are responsible for forming and solving the equations in FEM. The class dependencies in the proposed program is shown in Fig. 4.1.

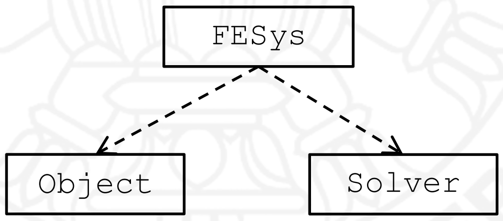


Fig. 4.1 Dependency relationship of a model-analysis pattern

In detail, the `Object` class is an abstract class which is used to derive the model classes such as `Node`, `BounCond`, `Element`, `Geometry` and `Material` as shown in Fig. 4.2.

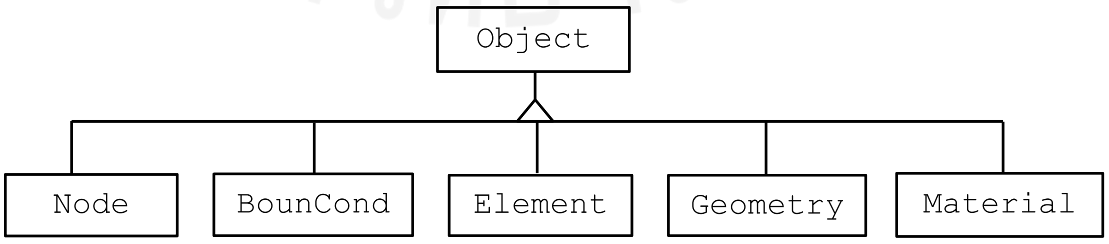


Fig. 4.2 Object class relationship

As MATLAB consists of many available solvers for solving not only linear but also nonlinear equations, MATLAB's solvers are directly used in this study. A `Solver` class is used as a generalization of all MATLAB's solvers. This `Solver` class has a simple relationship or an association with the `FESys` class. The overall class relationship of the program is expressed in Fig. 4.3.

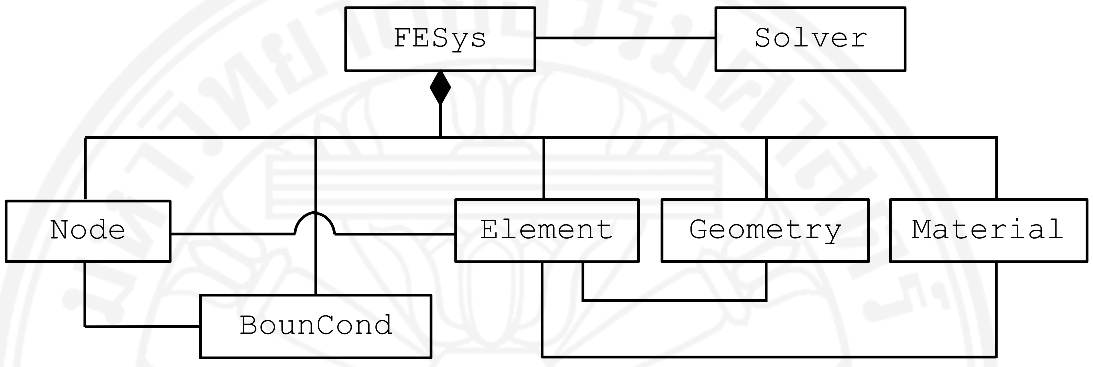


Fig. 4.3 Overall class relationship

As aforementioned, `FESys` is the main class of the symbolic FE system proposed in this study. This class is basically used to store the objects of the `Node`, `Element`, `Material`, `Geometry`, and `BounCond` classes. The main tasks of this class are to read the inputs from the text files, to assemble the global element stiffness matrices and force vectors, to compute the unknown nodal displacements and reaction forces, to update the nodal displacements, to determine the strain energy and force resultants of the structures, and to print out the outputs in the text files.

The `Node` class stores the nodal coordinates from the input file. It is a composition of the `FESys` class and has associations with the `Element` and `Boundary` classes. It also provides some methods to receive the number of degrees of freedom and the updated nodal displacement results.

The `Element` class is basically used to create the elements from the nodal connectivity. It is a composition of the `FESys` class and has an association with the `Node`, `Material`, and `Geometry` classes. The main tasks of this class are to store the objects of the `Node`, `Material`, and `Geometry` classes. It also provides several methods to generate the shape function matrices, mapping function matrices, and the element stiffness matrices in FEM. This class is used to derive many derived classes such as `Eline2` for 2D and 3D truss and beam elements, `ETri3` for 2D-three-node solid elements, `EQuad4` for 2D-four-node solid elements, and `ETet4` for 3D-eight-node solid elements. The specialization relationship of the `Element` class is shown in Fig. 4.4. In addition, to implement this `Element` class, there are some additional classes that are required such as the `Interpolation` and `MathModel` classes. The `Interpolation` class is responsible for storing various kinds of shape functions or interpolation functions. The `MathModel` class is used to store the mathematical formulations of different types of element in FE domains. The composition relationship of the `Element` class is illustrated in Fig. 4.5.

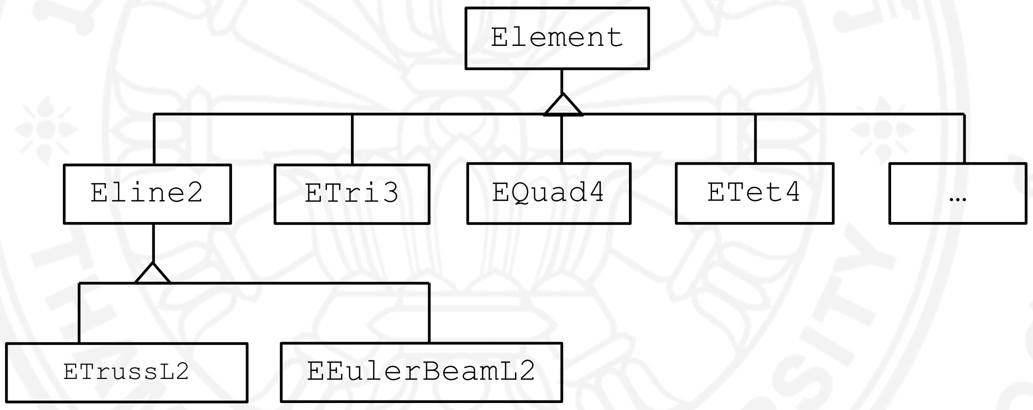


Fig. 4.4 Element class design relationship

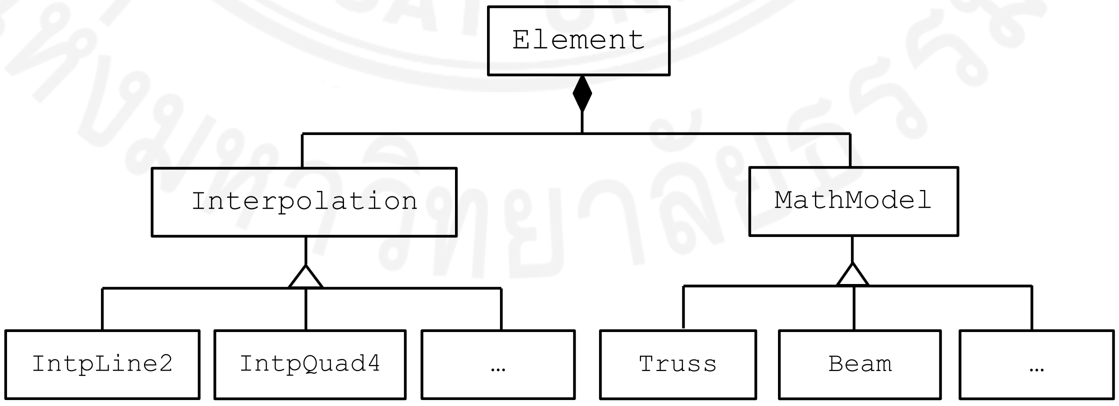


Fig. 4.5 Composition relationship of the `Element` class

The `Material` class provides a generic interface to handle different types of material in the program. The relation between this class and `FESys` is of a composition type. The `Material` class has an association with the `Element` class. It is used to store the material properties such as Young's modulus (E) and Poisson ratio (v) , and to provide methods to query these properties. Different types of material can be specialized by using derived classes as shown in Fig. 4.6. In the figure, the `MLElasIso` class represents linear elastic isotropic materials.

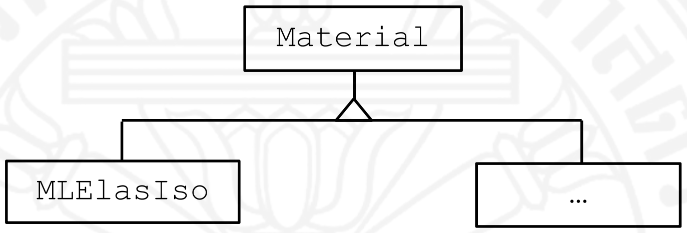


Fig. 4.6 Material class relationship

The `Geometry` class is used to store the geometry properties of structural elements. Examples include the cross sectional area (A), the moment inertia (I), and the torsional constant (J) for beam elements. This class is a composition of the `FESys` class, and has an association with the `Element` class. The class is used to derive some classes such as `GLine` for truss and beam elements and `GPlane` for 2D solid elements. The class relationship of the `Geometry` class is illustrated in Fig. 4.7.

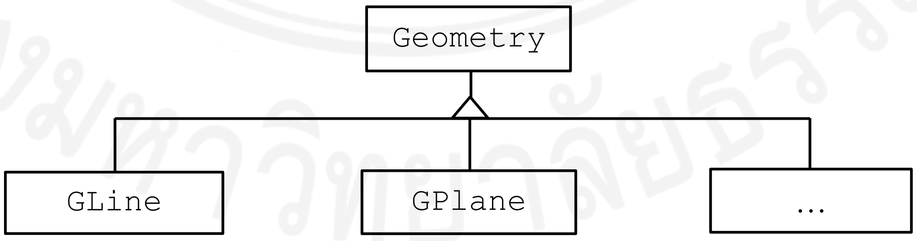


Fig. 4.7 Geometry class relationship

The `BounCond` class is used to store loads and prescribed displacement values from the input. This class is a composition of the `FESys` class, and also has an association with the `Node` class. Moreover, the class provides some methods to query the information of the boundary conditions in `FESys`. There are some classes that are derived from this class such as `NodalForce` for nodal forces and `NodalDisp` for nodal displacement values. The class relationship of the `Boundary` class is shown in Fig. 4.8.

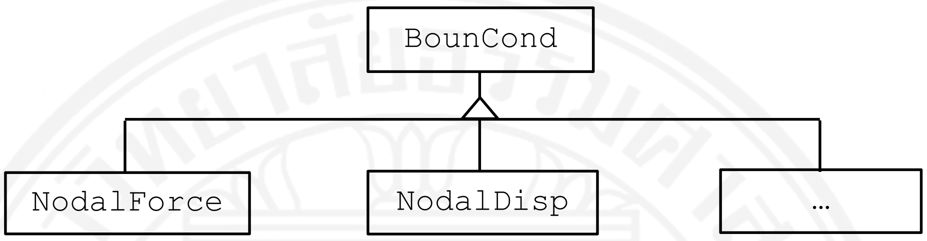


Fig. 4.8 Boundary class relationship

4.2 Symbolic computations

In order to perform symbolic computations, the employed computation platform must allow symbolic variables to be created. Naturally, it must also allow mathematical operations to be performed symbolically on these symbolic variables. The symbolic object-oriented FE program in this study is written in the symbolic language of MATLAB. In MATLAB, symbolic variables can be created by using commands `syms` or `sym('var')`. For example, to create symbolic variables for a sectional area `A`, a moment of inertia `I`, a beam length `L`, and Young's modulus `E`, the following statement can be used:

```
syms A I L E;
```

By using symbolic variables, it is straightforward to create symbolic data types in classes. Consider, for example, the `Material` and `MLElasIso` classes shown below. The `Material` class is the base class of all material types while the `MLElasIso` class represents linear elastic isotropic materials and is derived from `Material`. Parts of the two classes are shown below:

```
classdef Material
% This class is the base class for material classes
   properties (SetAccess = protected)
       number    = 0;          % Material ID number
       para      = {};         % Material parameters
   end
   methods
```

```
       ...
   end
end
```

```
classdef MLElasIso < Material
% This class is the class for linear elastic
% isotropic materials
   properties
       ...
   end
   methods
       % Constructor ------------------------------
       function s = MLElasIso(varargin)
           syms E Nu positive      % Young's modulus,
                                   % Poisson's ratio
           if nargin == 0          % No parameter
               s.para = {E, Nu};   % Default symbolic
                                   % E and Nu
           elseif nargin == 1      % One parameter:
                                   % ID number
               s.number = varargin{1};
               s.para = {E, Nu};   % Default symbolic
                                   % E and Nu
           elseif nargin == 2      % Two parameters:
                                   % E and Nu
               for i = 1:2
                   if isnumeric(varargin{i})    % If
                                     % numerical data
                       s.para{i} = varargin{i};
                   else                         % Else
               s.para{i}=sym(varargin{i},'positive');
                   end
               end
           end
       end
       % Function for setting Young's modulus-----
       function s = SetE(s,E)
           if isnumeric(E)   % If numerical data
               s.para{1} = E;
           else              % Else
               s.para{1} = sym(E,'positive');
           end
       end
       % Function for retrieving Young's modulus--
       function e = E(s)
            e = s.para{1};
       end
       ...
```

```
45:     end
46: end
```

The `Material` class has `number` for storing the material ID number, and `para` for storing material parameters. The modification access to these data is set to `protected`, which means that only modifications from within the class itself and from within derived classes are allowed. As the `MLElasIso` class is derived from `Material`, it automatically inherits `number` and `para` from `Material`. The `MLElasIso` class keeps only two material parameters, which are Young's modulus and Poisson's ratio. Young's modulus is kept in `para{1}` while Poisson's ratio in `para{2}`. If an object of type `MLElasIso` is created with no argument, then the `MLElasIso` class sets a symbolic `E` as Young's modulus and a symbolic `Nu` as Poisson's ratio. Functions can be prepared for setting and retrieving Young's modulus and Poisson's ratio from objects of type `MLElasIso`. For example, the functions `SetE(s,E)` in lines 33-39 of `MLElasIso` and `E(s)` in lines 41-43 are provided for setting and retrieving Young's modulus kept in `para{1}`.

An example below demonstrates how an object of type `MLElasIso` can be created and used:

```
>> m1 = MLElasIso();
>> m1.E()*m1.E()

ans =

E^2
```

Here, `m1` is an object of type `MLElasIso` created with no initial argument. As a result, the default symbolic `E` and `Nu` are used as Young's modulus and Poisson's ratio of `m1`. After that, as an example, the square of Young's modulus is symbolically computed. Young's modulus of `m1` is retrieved by using the function `m1.E()`. Note that, in MATLAB, the first argument of a member function of a class is always the object that is calling the function. For example, `s` in `E(s)` is the calling object itself. If the calling object is `m1`, then it can be passed into the function `E(s)` by writing `m1.E()` or `E(m1)`.

4.3 Symbolic argument passing

Simple symbolic computation capabilities can be found in high-end calculators. These calculators allow mathematical operations to be performed on symbolic variables. However, they generally do not allow symbolic argument passing in their programming platforms. As a result, complicated symbolic programs cannot be created. In MATLAB, symbolic arguments can be passed into a functions in the same way as numerical arguments. For example, if `m1` in the previous example is to be created with initial symbolic Young's modulus `E1` and Poisson's ratio `Nu1`, the following statements can be used:

```
>> syms E1 Nu1
>> m1 = MLElasIso(E1,Nu1);
>> m1.E()*m1.E()

ans =

E1^2
```

It can be seen that the square of Young's modulus correctly becomes `E1^2`. Technically, a copy of `E1` is created in `m1`. The function `SetE(s,E)` can be used to change Young's modulus from its current value, i.e.

```
>> syms E2
>> m1 = m1.SetE(E2);
>> sqrt(m1.E())

ans =

E2^(1/2)
```

Here, Young's modulus of `m1` is set to `E2`. This time, the square root of Young's modulus is symbolically computed. The correct symbolic result obtained confirms that the symbolic variable `E2` is successfully passed into the function `SetE(s,E)`. Note that it is also possible to set Young's modulus of `m1` to be numerical, i.e.

```
>> m1=m1.SetE(10.);
>> sqrt(m1.E())

ans =

    3.1623
```

It can be seen from lines 33-39 of `MLElasIso` that the function `SetE(s,E)` first checks whether the variable `E` received is a numerical variable or not. If it is a numerical variable, its value is copied to `para{1}`. If it is not, the function will create a symbolic positive real variable from `E` and copy it to `para{1}`. In addition to passing symbolic variables directly to a function, when an object is passed into a function, all of its member data, including its symbolic variables, are automatically passed into the function as internal parts of the object.

Another excellent example that shows why symbolic argument passing is essential to FE programming is a function for computing the local element stiffness matrix of a 2D Euler beam element. Let `EEulerBeamL2_2D` be the class that represents 2D Euler beam elements. In the `EEulerBeamL2_2D` class, the function `CalKel(s,EE,A,II,L)` shown below can be created to compute the local element stiffness of a 2D Euler beam element:

```
function s = CalKel(s,EE,A,II,L)
% The local element stiffness of a 2D Euler beam is
% computed here
    s.Kel(1,1) = (A*EE)/L;
    s.Kel(1,2) = 0;
    s.Kel(1,3) = 0;
    s.Kel(1,4) = -(A*EE)/L;
    s.Kel(1,5) = 0;
    s.Kel(1,6) = 0;
    s.Kel(2,1) = 0;
    s.Kel(2,2) = (12*EE*II)/L^3;
    ...
end
```

Besides the calling object itself, the function takes Young's modulus `EE`, the sectional area of the beam `A`, the moment of inertia `II`, and the length `L` as its arguments. Here, the variable `Kel` is the local stiffness matrix of a 2D Euler beam element and it is a member variable of `EEulerBeamL2_2D`. The statements below demonstrates how `CalKel(s,EE,A,II,L)` actually works:

```
>> syms E1 A1 I1 L1
>> myBeam = EEulerBeamL2_2D;
>> myBeam = myBeam.CalKel(E1,A1,I1,L1);
>> myBeam.Kel(2,2)
```

```
ans =

(12*E1*I1)/L1^3
```

Here, an object of type `EEulerBeamL2_2D`, called `myBeam`, is created to represent a 2D Euler beam element. The function `CalKel(s,EE,A,II,L)` of `myBeam` is executed with Young's modulus, the sectional area, the moment of inertia, and the beam length equal to `E1`, `A1`, `I1`, and `L1`, respectively. After that, as a demonstration, `Kel(2,2)` of `myBeam` is retrieved. Normally, the whole local element stiffness matrix will be operated upon and it can be obtained simply through `myBeam.Kel()`.

4.4 Simple user interface environments

A simple user interface environment allows the users to use the proposed program conveniently. It is illustrated using a simple realization diagram in Fig. 4.9.

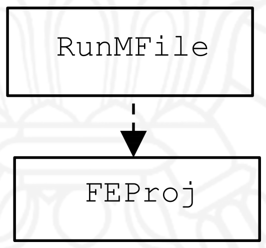
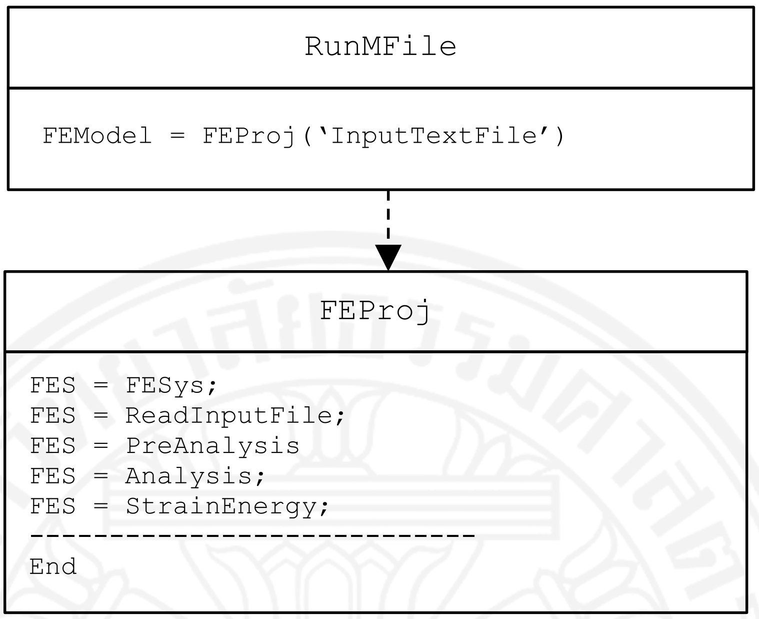


Fig. 4.9 Relationship of user interface environments

The interface environment of this program is created as a relationship between two files that are `RunMFile` and `FEProj` as shown in Fig. 4.10. `RunMFile` is an M-file used to run the program. The users can also use this file to extend the codes for other applications. `FEProj` is a function that performs all finite element processes from the beginning to the end. As shown in Fig. 4.10, the function `FEProj` receives the name of the input file `InputTextFile`. After that, the function performs the finite element processes as instructed.



```
FEModel = FEProj('InputTextFile')
```

```
FES = FESys;
FES = ReadInputFile;
FES = PreAnalysis
FES = Analysis;
FES = StrainEnergy;
----------------------------
End
```

Fig. 4.10 Example template of the simple user interface environment

Chapter 5
Advantages of the Proposed Program

5.1 General

The program obtained in this study can be used to effectively perform both numerical and symbolic computations for finite element analysis. By providing numerical inputs, the program can give numerical solutions as shown in Fig. 5.1. If symbolic computations are considered, the program needs symbolic inputs as shown in Fig. 5.2. The obtained program can be used to solve engineering problems such as structural analysis, structural design, and mechanics of materials. Below, the advantages of the obtained program in solving different types of FE problem are briefly discussed.

$$\begin{tikzcd}\boxed{\text{Numerical Inputs}} \arrow[r] & \boxed{\text{Numerical Computations}} \arrow[r] & \boxed{\text{Numerical Outputs}}\end{tikzcd}$$

Fig. 5.1 Process of numerical computations in the program

$$\begin{tikzcd}\boxed{\text{Symbolic Inputs}} \arrow[r] & \boxed{\text{Symbolic Computations}} \arrow[r] & \boxed{\text{Symbolic Outputs}}\end{tikzcd}$$

Fig. 5.2 Process of symbolic computations in the program

5.2 Structural analysis problems

The structural engineering field consists of structural analysis and structural design fields. Structural analysis helps engineers to understand the physical behavior of structures under loads caused by the gravity, climatic conditions and ground conditions, while structural design ensures and deals with the stability and durability of structures to withstand loads.

Normally, displacements, forces, and stresses of structural members are the primary parameters for structural engineers. For example, in designing a truss structure, the stresses in its members must not exceed the allowable strength in the ultimate state, and the displacements of the truss must also satisfy the state of serviceability. By using the obtained program, it is straightforward to find the values of these stresses and displacements either by numerical or symbolic computations. Chapter 6 will illustrate some advantages of the proposed program when it is used to solve structural analysis problems.

5.3 Structural design problems

It is safe to say that structural design problems are in fact optimization problems. This is because each structural design problem is generally a problem of finding the most efficient design that results in a structure that satisfies all the required constraints. In real practice, the word “efficient’ usually means “economical.” Since the obtained program can perform closed-form analysis, the closed-form solutions of displacements, strains, forces, and stresses of structures can be determined. These symbolic results can be used to help solve structural optimization problems. For example, when a sizing optimization problem of a truss is to be solved by a conventional nonlinear programming technique, the closed-form solutions of displacements and stresses can be very useful. The closed-form displacements and stresses allow the gradients and Hessians of displacement and stress functions to be analytically determined. Since conventional nonlinear programming techniques are gradient-based methods, these analytical gradients and Hessians from the proposed program can be directly used in these optimization techniques without resorting to their numerical approximations. Chanter 7 will demonstrate the usefulness of the program in structural optimization problems.

5.4 Mechanics of materials

The capability of the proposed program in giving symbolic FE solutions can be used to derive closed-form solutions of some problems in the field of mechanics of materials. For example, the proposed program can be used to derive the closed-form effective elastic constants of some periodic cellular solids. Periodic cellular solids are made up of interconnected solid struts or plates which form the edges and faces of the cells (Gibson and Ashby, 1999). They are used in many types of structure in various scales. In many of their applications, their effective elastic properties such as the effective Young’s modulus, Poisson ratio and shear modulus are of interest. These effective elastic constants can be computed by homogenization methods from unit cells of periodic cellular solids.

Periodic cellular solids that are frame-like can be modelled accurately as frame structures using beam elements. Many researchers have manually determined the closed form effective elastic constants of frame-like periodic cellular solids. For example, Gibson and Ashby (1999) have determined the closed-form effective elastic constants of periodic cellular solids with square, triangle, and hexagon unit cells. However, manual computations are tedious and prone to errors. In addition, when the structures are too complex, manual computations are simply not possible. The proposed program can be used instead of manual computations. The determination of the closed-form solutions of the effective elastic constants of the frame-like periodic cellular solids by using the obtained program are considered in Chapter 8.

Chapter 6
Closed-Form Analysis of Simple Structures

6.1 General

Closed-form analysis can be used to determine the mathematic expressions or formula of a particular problem. There are times when closed-form analysis of structures is required or preferred. When only a simple structural member is considered, it is possible to perform this type of analysis by hand. However, when a structure with several members is to be considered, manual calculation becomes impossible even when closed-form solutions are theoretically obtainable. By using the obtained program, the closed-form solutions of simple structures, such as simple trusses and frames can be determined straightforwardly.

The obvious advantage of the obtained program when compared with other numerical FE programs is that the proposed program is capable of performing both numerical and symbolic FEA. When numerical computations are considered, the obtained program can be used in the same way as numerical FE programs. However, when symbolic computations are to be considered, the proposed program becomes distinctively useful. This chapter demonstrates the advantage of the obtained program in providing symbolic FE solutions by solving some FE problems of simple truss and frame structures. In order to validate the proposed program, the closed-form solutions obtained from the program are numerically compared with those from a commercial FE program.

6.2 FEA of simple trusses and frames

The truss and frame structures in Fig. 6.1 and Fig. 6.2 are analytically analyzed by the proposed program. The program is used to determine the closed forms of the nodal displacements, support reactions, total strain energy, internal forces, strains, and stresses for the structures as functions of the following variables:

A_i = The cross-sectional area of structural member i,

I_i = The moment inertia of structural member i,

L = The length of the structure,

H = The height of the structure,

E_e= Young’s modulus of the material,

P_i = The force value at node i.

D_x= The prescribed displacement value,

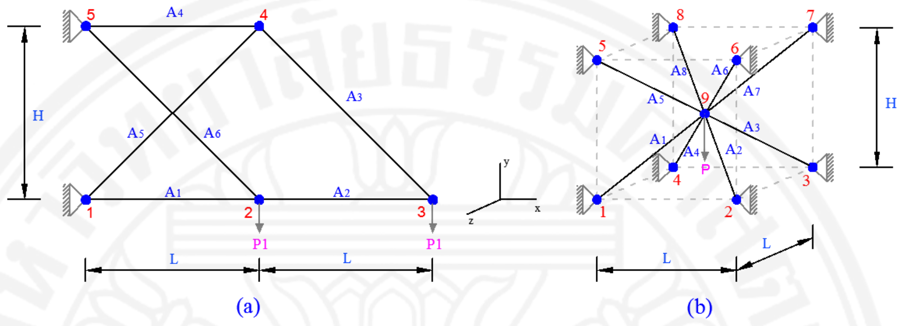


Fig. 6.1 2D and 3D trusses

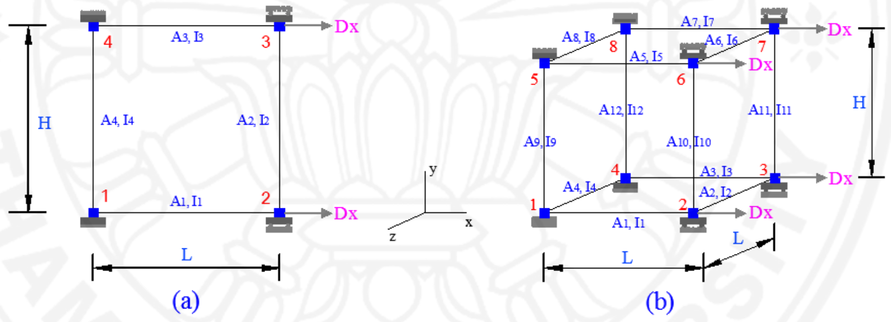


Fig. 6.2 2D and 3D frames

6.3 Results

The symbolic inputs and symbolic outputs of the program are shown in the Appendix A and Appendix B, respectively. The comparisons of the numerical results from the obtained closed forms and MSC.Marc Mentat are provided in Appendix C.

6.4 Discussions

The comparisons of the numerical results from the obtained closed forms and the commercial FE program are satisfactory. Since the numbers can be kept as symbolic variables during computations in the proposed program, the numerical errors are reduced as well. Due to the obtained closed-form solutions, the engineers do not have to run the program many times.

Chapter 7
Truss Optimization Problems

7.1 General

Various techniques of optimization have been used to achieve the optimal weights of truss structures. Truss optimization problems can be categorized into three different problems, namely sizing, shape and topology optimization problems (Christensen and Klarbring, 2008). Generally, the objective of a truss optimization problem is to minimize the total weight of the truss, which is subject to displacement and stress constraints. Most of the time, truss optimization problems are nonlinear problems that can be convex or non-convex problems. To deal with these problems, two well-known groups of methods have been used, namely the conventional nonlinear programming (NLP) approaches and metaheuristic algorithms. The conventional NLP methods are gradient-based methods. This means that the information on the gradients and Hessians of stress and displacement functions with respect to member areas is generally required. If an ordinary numerical FE program is used, the gradients and Hessians have to be numerically approximated. The lack of the gradients and Hessians of the constraint functions naturally encourages the use of metaheuristic optimization methods, which are non-gradient-based methods. Most of these methods are derived from observations of natural phenomena. Examples of these methods include genetic algorithms (GAs), particle swarm optimization (PSO), firefly algorithms (FAs), ant system (AS) algorithms, harmony search (HS) algorithms, and water cycle algorithms (WCAs).

By using the proposed program, the closed-form solutions of displacements and stresses can be obtained. As a result, the conventional NLP methods can be used to deal with truss optimization problems because the gradients and Hessians of the displacement and stress functions can be exactly computed from the symbolic displacements and stresses.

In this chapter, sizing optimization of 2D truss structures by using a conventional NLP method in MATLAB is presented. Some benchmark problems of 2D truss structures in the literature are tested and compared.

7.2 Truss optimization problems

The objective of truss optimization in this study is to minimize the weight of the truss with respect to its member areas that are continuous variables. The problem can be expressed mathematically as

$$\begin{aligned} &\text{Minimize} && W = \textstyle\sum_{i=1}^{n} \rho_i A_i l_i \\ &\text{Subject to} && \sigma_a - |\sigma_i(A_1, \dots, A_n)| \geq 0, \quad i = 1,2, \dots, n \\ &&& \delta_a - |\delta_j(A_1, \dots, A_n)| \geq 0, \quad j = 1,2, \dots, m \\ &&& A_i^{min} \leq A_i \leq A_i^{max}, \quad i = 1,2, \dots, n, \end{aligned} \tag{14}$$

where

W	=	The overall weight of the truss structure.
ρ_i	=	The weight density of the material of member i.
l_i	=	The length of member i.
A_i	=	The area of member i.
σ_i	=	The stress of member i.
δ_j	=	The degree of freedom j.
σ_a	=	The allowable stress.
δ_a	=	The allowable displacement.
A_i^{min}	=	The minimum area of member i.
A_i^{max}	=	The maximum area of member i.
n	=	The number of members.
m	=	The number of degrees of freedom.

7.3 MATLAB optimization toolbox

MATLAB optimization toolbox provides a command `fmincon` to deal with constrained nonlinear optimization problems. Under `fmincon`, different optimization methods, such as interior-point, active-set, sequential-quadratic-programming and trust-region-reflective algorithms, can be selected for use. In this study, the command `fmincon` with the interior-point algorithm is used. The options available in `fmincon` to specify whether the gradients and Hessians of the objective and constraint functions are available for the optimization or not are also utilized. The syntax of `fmincon` can be show as follow:

```
[x,fval] = fmincon(fun,x0,A,b,Aeq,beq,lb,ub,nonlcon,options)
```

Here, `fun` is the objective function. The constraint functions and their gradients, if they are considered, are specified via `nonlcon`. In addition, `options` is used to specify the set of optimization options. The set of options when the gradients and Hessians are not used can be specified as

```
options = optimoptions('fmincon','Algorithm',...
'interior-point','GradObj','off','GradConstr','off')
```

Moreover, when the gradients and Hessians are considered, the set of options can be specified as below expression, where the Hessians are specified through `myhess`.

```
options = optimoptions('fmincon','Algorithm', ...
'interior-point','GradObj','on','GradConstr','on', ...
'Hessian','user-supplied','HessFcn',myhess)
```

7.4 Results

7.4.1 Truss with six members and five nodes

Fig. 7.1 shows a truss with six members and five nodes to be optimized. The problem parameters are as follows: Young's modulus $E = 10000\ ksi$, the weight density $\rho = 0.1\ lb/in^3$, the allowable stress $\sigma_a = 25\ ksi$, the allowable displacement $\delta_a = 2\ in$, and the limit cross sectional areas $A^{min}, A^{max} = 0.09, 35\ in^2$.

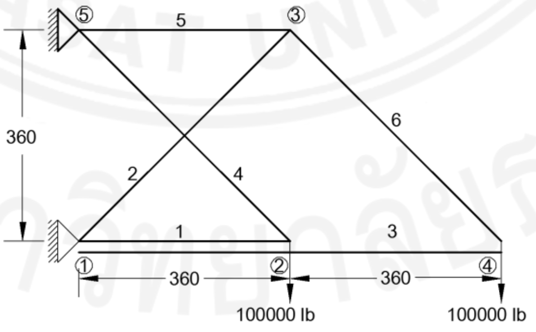


Fig. 7.1 Problem 1--truss with six members and five nodes

The results are shown in Table 7.1. The results from `fmincon` without and with gradients and Hessians are shown as fmincon1 and fmincon2, respectively. The results from `fmincon` without and with gradients and Hessians are found to be the same and comparable to the results from the literature. Note that the results by Deb and Gulati

(2001) and Luh and Lin (2008) are from topology optimization. The comparison with the literature is only to validate the present results. Fig. 7.2 shows the convergence rates from `fmincon` without and with gradients and Hessians. It can be seen that the convergence rate from `fmincon` when gradients and Hessians are used is the fastest.

Table 7.1 Results of Problem 1

Member	Area (in^2)			
	Deb and Gulati (2001)	Luh and Lin (2008)	Present study	
			fmincon1	fmincon2
1	05.219	05.428	05.4000	05.4000
2	20.310	20.549	20.3647	20.3640
3	14.593	14.308	14.4000	14.4000
4	07.772	07.617	07.6368	07.6368
5	28.187	28.876	28.8000	28.8000
6	20.650	20.265	20.3647	20.3647
Weight of truss (lb)	4731.650	4730.824	4730.4000	4730.4000

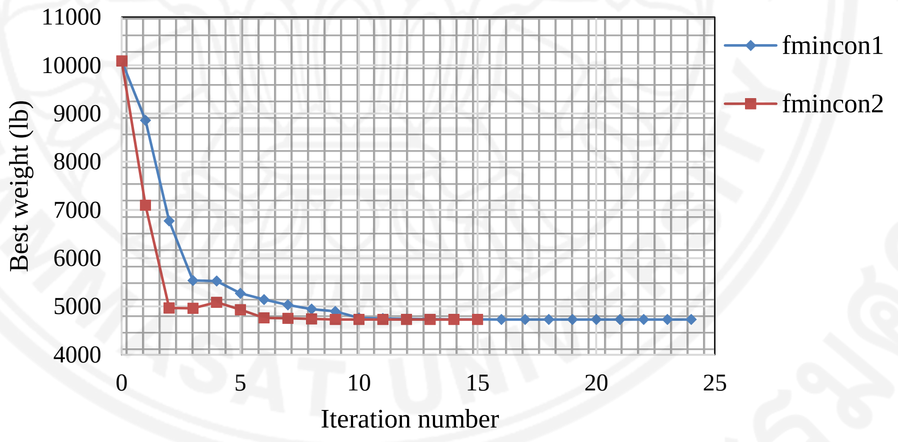


Fig. 7.2 Convergence rates of Problem 1

7.4.2 Truss with ten members and six nodes

Fig. 7.3 shows a truss with ten members and six nodes (Farshi and Alinia-ziazi, 2010, Li et al., 2007). The problem parameters are the same as the previous problem except for A^{min}, which is equal to 0.1 in^2 in this problem. The applied loads are given as $P1 = 150\ kips$ and $P2 = 50\ kips$.

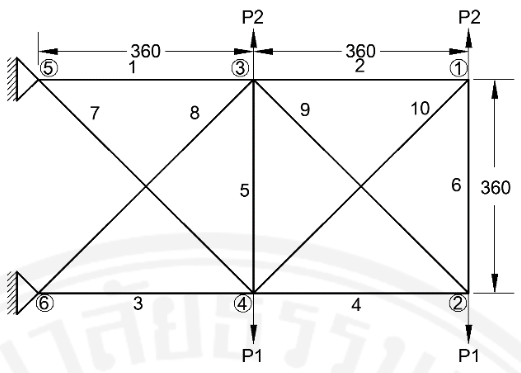


Fig. 7.3 Problem 2--truss with ten members and six nodes

The results are shown in Table 7.2. The results from `fmincon` without and with gradients and Hessians are found to be the same and comparable with those from the literature. Note that the results by Farshi and Alinia-ziazi (2010) and Li et al. (2007) are from sizing optimization. From Fig. 7.4, it can be seen that the convergence rate from `fmincon` with gradients and Hessians is the fastest compared to `fmincon` without gradients and Hessian.

Table 7.2 Results of Problem 2

Member	Area (in^2)			
	Li et al. (2007)	Farshi and Alinia-ziazi (2010)	Present study	
			fmincon1	fmincon2
1	23.353	23.5270	23.5307	23.5307
2	00.100	00.1000	00.1000	00.1000
3	25.502	25.2941	25.2851	25.2851
4	14.250	14.3760	14.3745	14.3745
5	00.100	00.1000	00.1000	00.1000
6	01.972	01.9698	01.9697	01.9697
7	12.363	12.4041	12.3906	12.3906
8	12.894	12.8245	12.8277	12.8277
9	20.356	20.3304	20.3286	20.3286
10	00.101	00.1000	00.1000	00.1000
Weight of truss (lb)	4677.29	4677.80	4676.92	4676.92

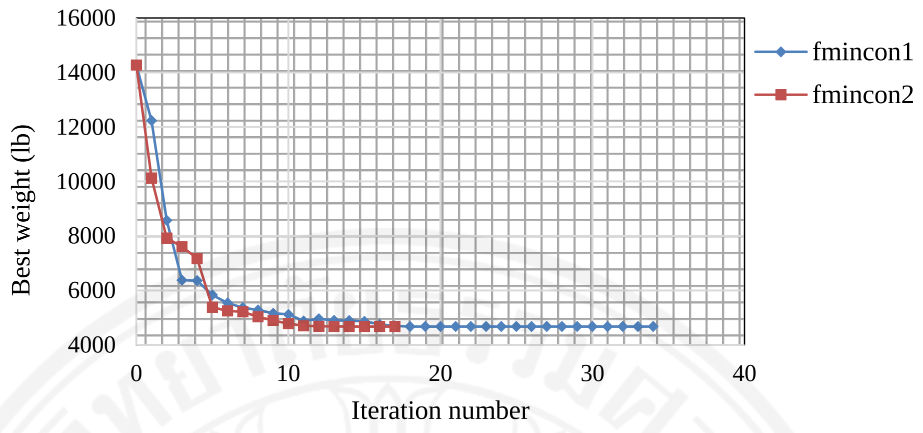


Fig. 7.4 Convergence rates of Problem 2

7.5 Discussions

In this chapter, the advantages of the obtained program for sizing optimization of truss structures are demonstrated. By using the proposed program, the constraint displacement and stress functions can be constructed from the stress and displacement solutions. In this study, two 2D truss optimization problems are solved as case studies using a conventional NLP algorithm in MATLAB. The analytical gradients and Hessians of the objective and constraint functions are used in the optimization. The obtained results show that, although symbolic FE solutions do not help improve the quality of the optimization solutions, they can help improve the convergence rates of the optimization process.

Chapter 8
Effective Constants of Frame-Like Periodic Cellular Solids

8.1 General

The homogenization method based on equivalent strain energy is one of the well-known homogenization methods that have been used to find the effective elastic properties of periodic cellular solids (Zhang et al., 2007, Dai and Zhang, 2009). In this method, the values of strain energy of a unit cell under different strain modes are used in the determination of the effective elastic properties. The unit cell can be modeled by using FEM. These strain modes are created by prescribing periodic kinematic boundary conditions to the unit cell. The periodic boundary conditions are constraint equations that prescribe relationships between different degrees of freedom in the FE model of the unit cell, and are usually called multi-point or multi-freedom constraints. The method of Lagrange multipliers are usually used to enforce multi-freedom constraints.

The work in this chapter aims to demonstrate how symbolic FE programming can be used to analytically determine the closed-form solutions of the effective elastic constants of frame-like periodic cellular solids. Unit-cell structures are modelled by using Euler beam elements (Gibson and Ashby, 1999). Periodic boundary conditions are prescribed by the method of Lagrange multipliers. The proposed symbolic FE program is used to determine the closed-form effective elastic constants of some 2D frame-like periodic cellular solids. First, the symbolic FE program is used to determine the closed-form solutions of strain energy of a unit cell under various strain modes. These closed-form strain energy expressions are then used to determine the closed-form effective elastic constants. The closed-form effective elastic constants obtained in this study are compared numerically with numerical results obtained from a commercial FE program.

8.2 Strain-energy based homogenization

Consider a domain V of a periodic cellular solid that is composed of a large number of unit cells. A set of kinematic boundary conditions is applied to the domain such that the displacement u_i field becomes

$$u_i = \varepsilon_{ij}^o x_j + u_i^p. \tag{15}$$

Here, x_j is the coordinate vector. In addition, ε_{ij}^o is a constant symmetric tensor and u_i^p is the periodic component of u_i. Let $\langle Q \rangle$ denote the volume average of any quantity Q in V. The effective material constitutive C_{ijkl}^* tensor is defined as

$$\langle \sigma_{ij} \rangle = C_{ijkl}^* \langle \varepsilon_{kl} \rangle. \tag{16}$$

It can be shown that Eq. (15) results in $\langle \epsilon_{kl} \rangle = \epsilon_{kl}^o$. (Suquet, 1987). For 2D orthotropic solids under the plane stress condition, Eq. (16) can be written in matrix form as

$$\boldsymbol{\sigma}^o = \langle \begin{Bmatrix} \sigma_{11} \\ \sigma_{22} \\ \sigma_{12} \end{Bmatrix} \rangle = \begin{bmatrix} c_{11}^* & c_{12}^* & 0 \\ & c_{22}^* & 0 \\ Sym & & c_{33}^* \end{bmatrix} \langle \begin{Bmatrix} \varepsilon_{11} \\ \varepsilon_{22} \\ 2\varepsilon_{12} \end{Bmatrix} \rangle = \mathbf{C}^* \boldsymbol{\varepsilon}^o. \tag{17}$$

The effective elastic constants can be expressed in terms of components of the constitutive matrix as

$$E_1^* = [c_{11}^* c_{22}^* - (c_{12}^*)^2]/c_{22}^*, \quad E_2^* = [c_{11}^* c_{22}^* - (c_{12}^*)^2]/c_{11}^*, \tag{18}$$

$$v_{12}^* = c_{12}^*/c_{22}^*, \quad v_{21}^* = c_{12}^*/c_{11}^*, \quad G_{12}^* = c_{33}^*. \tag{19}$$

It can be shown that (Suquet, 1987)

$$\frac{U_C}{V_C} = \langle \frac{1}{2} \sigma_{ij} \varepsilon_{ij} \rangle = \frac{1}{V} \int_V \frac{1}{2} \sigma_{ij} \varepsilon_{ij} dV = \frac{1}{V_C} \int_{V_C} \frac{1}{2} \sigma_{ij} \varepsilon_{ij} dV = \frac{1}{2} \langle \sigma_{ij} \rangle \langle \varepsilon_{ij} \rangle = \frac{1}{2} C_{ijkl}^* \varepsilon_{kl}^o \varepsilon_{ij}^o, \tag{20}$$

where U_C and V_C denote the strain energy and the volume of the unit cell, respectively. Note that the displacement field u_i in Eq. (15) results in σ_{ij} and ε_{ij} that are periodic, and an average of a periodic quantity over V is the same as an average over V_C. By prescribing different values of ε_{kl}^o to the unit cell via Eq. (15) and computing the corresponding strain energy values by FEM, C_{ijkl}^* can be obtained from Eq.(20).

The constitutive matrix of a 2D periodic cellular solid under the plane stress condition can be obtained from its unit cell by using the following equations, i.e.

$$c_{11}^* = \frac{2U_{C1}}{V_C}, \quad c_{22}^* = \frac{2U_{C2}}{V_C}, \quad c_{33}^* = \frac{2U_{C4}}{V_C}, \quad c_{12}^* = \frac{(U_{C3} - U_{C1} - U_{C2})}{V_C}, \tag{21}$$

where U_{C1}, U_{C2}, U_{C3}, and U_{C4} are the strain energy values of the unit cell under four strain modes in which, respectively,

$$\varepsilon^o = [1 \quad 0 \quad 0]^T, \ [0 \quad 1 \quad 0]^T, \ [1 \quad 1 \quad 0]^T, \text{ and } [0 \quad 0 \quad 1]^T. \tag{22}$$

8.3 Periodic boundary conditions

When periodic boundary conditions are prescribed to an FE domain, two types of boundary prescription are necessary. The first type is the ordinary prescription of exact values of degrees of freedom. The second type is the prescription of relative values between degrees of freedom. The first type of boundary condition is used to prevent rigid body displacements and these boundary conditions must not, by themselves, create any strain. The second type of boundary condition is from the periodic displacement u_i^p in Eq. (15). If any two nodes have the same u_i^p because of the periodicity, the relative displacements between the two nodes can be obtained from Eq. (15) for each prescribed strain mode. In addition, their rotational degrees of freedom must be the same.

Fig. 8.1 shows the 2D frame-like periodic cellular solids considered in this study. Fig. 8.2 shows the frame structures that represent the unit cells of the two cellular solids. Note that cutting a strut of a frame-like periodic cellular solid in half longitudinally to create its unit-cell structure results in a strut of the unit-cell structure that has only half axial and bending rigidities of the original strut (Theerakittayakorn and Nanakorn, 2013). Write Eq. (15) in matrix form for 2D periodic cellular solids as

$$\mathbf{u} = \begin{Bmatrix} u_1 \\ u_2 \end{Bmatrix} = \begin{bmatrix} \varepsilon_{11}^o & \varepsilon_{12}^o \\ Sym & \varepsilon_{22}^o \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \end{Bmatrix} + \begin{Bmatrix} u_1^p \\ u_2^p \end{Bmatrix} = \mathbf{E}^o\mathbf{x} + \boldsymbol{u}^{\boldsymbol{p}}. \tag{23}$$

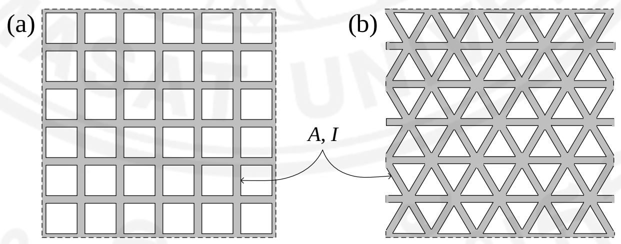


Fig. 8.1 Periodic cellular solids: (a) square unit cells, (b) triangular unit cells

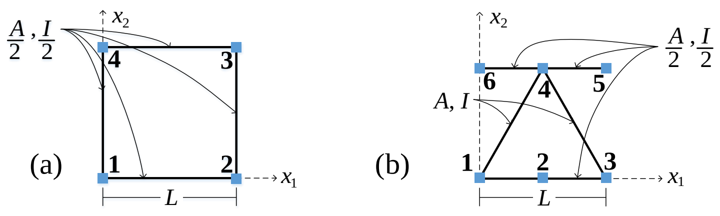


Fig. 8.2 Frame structures representing the square and triangular unit cells

In order to utilize Eq. (23) to create the required periodic boundary conditions, the periodicity of the unit cell must be considered. As an example, consider the triangular unit-cell structure in Fig. 8.2(b). This unit cell has the following periodicity conditions

$$\mathbf{u}^p(0,0) = \mathbf{u}^p(L,0) = \mathbf{u}^p\left(\frac{L}{2},\frac{\sqrt{3}L}{2}\right), \tag{24}$$

$$\boldsymbol{u}^p\left(\frac{L}{2},0\right) = \boldsymbol{u}^p\left(L,\frac{\sqrt{3}L}{2}\right) = \boldsymbol{u}^p\left(0,\frac{\sqrt{3}L}{2}\right), \tag{25}$$

$$\boldsymbol{\theta}^p(0,0) = \boldsymbol{\theta}^p(L,0) = \boldsymbol{\theta}^p\left(\frac{L}{2},\frac{\sqrt{3}L}{2}\right), \tag{26}$$

$$\boldsymbol{\theta}^p\left(\frac{L}{2},0\right) = \boldsymbol{\theta}^p\left(L,\frac{\sqrt{3}L}{2}\right) = \boldsymbol{\theta}^p\left(0,\frac{\sqrt{3}L}{2}\right). \tag{27}$$

In order to prevent the rigid body displacements, $\mathbf{u}(0,0)$ can be selected to be fixed. If, for example, U_{C4} is to be determined, Eq. (23), Eq. (24) and Eq. (25) yield

For node 1:

$$\begin{Bmatrix} u_1(0,0) \\ u_2(0,0) \end{Bmatrix} = \begin{bmatrix} 0 & 1/2 \\ 1/2 & 0 \end{bmatrix} \begin{Bmatrix} 0 \\ 0 \end{Bmatrix} + \begin{Bmatrix} u_1^p(0,0) \\ u_2^p(0,0) \end{Bmatrix} = \begin{Bmatrix} 0 \\ 0 \end{Bmatrix}, \tag{28}$$

For node 2:

$$\begin{Bmatrix} u_1\left(\frac{L}{2},0\right) \\ u_2\left(\frac{L}{2},0\right) \end{Bmatrix} = \begin{bmatrix} 0 & 1/2 \\ 1/2 & 0 \end{bmatrix} \begin{Bmatrix} \frac{L}{2} \\ 0 \end{Bmatrix} + \begin{Bmatrix} u_1^p\left(\frac{L}{2},0\right) \\ u_2^p\left(\frac{L}{2},0\right) \end{Bmatrix} = \begin{Bmatrix} u_1^p\left(\frac{L}{2},0\right) \\ \frac{L}{4} + u_2^p\left(\frac{L}{2},0\right) \end{Bmatrix}, \tag{29}$$

For node 3:

$$\begin{Bmatrix} u_1(L,0) \\ u_2(L,0) \end{Bmatrix} = \begin{bmatrix} 0 & 1/2 \\ 1/2 & 0 \end{bmatrix} \begin{Bmatrix} L \\ 0 \end{Bmatrix} + \begin{Bmatrix} u_1^p(L,0) \\ u_2^p(L,0) \end{Bmatrix} = \begin{Bmatrix} u_1^p(0,0) \\ \frac{L}{2} + u_2^p(0,0) \end{Bmatrix} = \begin{Bmatrix} 0 \\ \frac{L}{2} \end{Bmatrix}, \tag{30}$$

For node 4:

$$\begin{Bmatrix} u_1\left(\frac{L}{2},\frac{\sqrt{3}L}{2}\right) \\ u_2\left(\frac{L}{2},\frac{\sqrt{3}L}{2}\right) \end{Bmatrix} = \begin{bmatrix} 0 & 1/2 \\ 1/2 & 0 \end{bmatrix} \begin{Bmatrix} \frac{L}{2} \\ \frac{\sqrt{3}L}{2} \end{Bmatrix} + \begin{Bmatrix} u_1^p\left(\frac{L}{2},\frac{\sqrt{3}L}{2}\right) \\ u_2^p\left(\frac{L}{2},\frac{\sqrt{3}L}{2}\right) \end{Bmatrix} = \begin{Bmatrix} \frac{\sqrt{3}L}{4} \\ \frac{L}{4} \end{Bmatrix}, \tag{31}$$

For node 5:

$$\begin{Bmatrix} u_1\left(L,\dfrac{\sqrt{3}L}{2}\right) \\ u_2\left(L,\dfrac{\sqrt{3}L}{2}\right) \end{Bmatrix} = \begin{bmatrix} 0 & 1/2 \\ 1/2 & 0 \end{bmatrix} \begin{Bmatrix} L \\ \dfrac{\sqrt{3}L}{2} \end{Bmatrix} + \begin{Bmatrix} u_1^p\left(L,\dfrac{\sqrt{3}L}{2}\right) \\ u_2^p\left(L,\dfrac{\sqrt{3}L}{2}\right) \end{Bmatrix} = \begin{Bmatrix} \dfrac{\sqrt{3}L}{4} + u_1^p\left(\dfrac{L}{2},0\right) \\ \dfrac{L}{2} + u_2^p\left(\dfrac{L}{2},0\right) \end{Bmatrix}, \tag{32}$$

For node 6:

$$\begin{Bmatrix} u_1\left(0,\dfrac{\sqrt{3}L}{2}\right) \\ u_2\left(0,\dfrac{\sqrt{3}L}{2}\right) \end{Bmatrix} = \begin{bmatrix} 0 & 1/2 \\ 1/2 & 0 \end{bmatrix} \begin{Bmatrix} 0 \\ \dfrac{\sqrt{3}L}{2} \end{Bmatrix} + \begin{Bmatrix} u_1^p\left(0,\dfrac{\sqrt{3}L}{2}\right) \\ u_2^p\left(0,\dfrac{\sqrt{3}L}{2}\right) \end{Bmatrix} = \begin{Bmatrix} \dfrac{\sqrt{3}L}{4} + u_1^p\left(\dfrac{L}{2},0\right) \\ u_2^p\left(\dfrac{L}{2},0\right) \end{Bmatrix}. \tag{33}$$

From Eq. (29) and Eq. (32), the multi-freedom constraints between nodes 2 and 5 can be obtained as

$$\begin{Bmatrix} u_1\left(\dfrac{L}{2},0\right) - u_1\left(L,\dfrac{\sqrt{3}L}{2}\right) \\ u_2\left(\dfrac{L}{2},0\right) - u_2\left(L,\dfrac{\sqrt{3}L}{2}\right) \end{Bmatrix} = \begin{Bmatrix} -\dfrac{\sqrt{3}L}{4} \\ -\dfrac{L}{4} \end{Bmatrix}. \tag{34}$$

From Eq. (29) and Eq. (33), the multi-freedom constraints between nodes 2 and 6 can be obtained as

$$\begin{Bmatrix} u_1\left(\dfrac{L}{2},0\right) - u_1\left(0,\dfrac{\sqrt{3}L}{2}\right) \\ u_2\left(\dfrac{L}{2},0\right) - u_2\left(0,\dfrac{\sqrt{3}L}{2}\right) \end{Bmatrix} = \begin{Bmatrix} -\dfrac{\sqrt{3}L}{4} \\ \dfrac{L}{4} \end{Bmatrix}. \tag{35}$$

In addition to the four constraint equations from Eq. (34) and (35), the four constraint equations from Eq. (26) and (27) have to be considered. It can be seen from Eq. (30) and (31) that, for nodes 3 and 4, exact values of u_1 and u_2 are known and can be directly prescribed. The multi-freedom constraints in the determination of U_{C1}, U_{C2}, and U_{C3} can be obtained in the same way.

8.4 FE formulation with Lagrange multipliers

The multi-freedom constraint equations are incorporated into FE analysis by means of Lagrange multipliers. To begin with, consider the total potential energy Π of an FE domain in Eq. (9). All multi-freedom constraint equations can be written together in matrix form as

$$\mathbf{AQ} - \mathbf{B} = \mathbf{0}. \tag{36}$$

Here, the sizes of the matrix **A** and vector **B** are $n \times m$ and $n \times 1$, respectively, where n denotes the number of the multi-freedom constraint equations and m denotes the number of degrees of freedom in **Q**. In order to consider Eq. (36), a Lagrange function can be created with a Lagrange multiplier vector $\boldsymbol{\lambda}$ as

$$L(\mathbf{Q}, \boldsymbol{\lambda}) = \frac{1}{2}\mathbf{Q}^T\mathbf{KQ} - \mathbf{Q}^T\mathbf{F} + \boldsymbol{\lambda}^T(\mathbf{AQ} - \mathbf{B}). \tag{37}$$

By minimizing L with respect to **Q** and $\boldsymbol{\lambda}$, Eq. (37) yields

$$\begin{bmatrix} \mathbf{K} & \mathbf{A}^T \\ \mathbf{A} & 0 \end{bmatrix} \begin{Bmatrix} \mathbf{Q} \\ \boldsymbol{\lambda} \end{Bmatrix} = \begin{Bmatrix} \mathbf{F} \\ \mathbf{B} \end{Bmatrix}. \tag{38}$$

The Eq. (38) is implemented in a function of the `FESys` class. the program will determine the unknown displacement vector **Q** from this equation.

8.5 Symbolic FE computations

By using the obtained program, U_{C1}, U_{C2}, U_{C3}, and U_{C4} can be symbolically determined. Thereafter, the closed-form effective elastic constants can be obtained from the following statements in MATLAB scripts,

```
c11 = 2*Uc1/Vc;
c22 = 2*Uc2/Vc;
c33 = 2*Uc4/Vc;
c12 = (Uc3-Uc1-Uc2)/Vc;
E1 = simplify((c11*c22-c12^2)/c22);
E2 = simplify((c11*c22-c12^2)/c11);
v12 = simplify(c12/c22);
v21 = simplify(c12/c11);
G12 = simplify(c33);
```

8.6 Results

Table 8.1 shows the obtained closed-form solutions of the effective elastic constants of the square and triangular periodic cellular solids. All struts in each solid have the same area A and moment of inertia I. In the table, E denotes Young's modulus of the base material. In order to assure that the obtained closed forms are correct, numerical comparisons with a commercial FE program are done. In the numerical comparisons, all struts are assumed to have the same square cross section of $T \times T$.

Table 8.2 shows the comparisons of the obtained closed forms with those from MSC.Marc Mentat. It can be seen that the results from this study and MSC.Marc Mentat are exactly the same.

Table 8.1 Closed-form solutions of effective elastic properties

Unit cell	**Effective Young's modulus**	**Effective shear modulus**	**Effective Poisson's ratio**
Square	$E_i^* = \frac{E}{V_C}(AL)$	$G_{12}^* = \frac{E}{V_C}\left[6\left(\frac{I}{L}\right)\right]$	$v_{ij}^* = 0$
Triangle	$E_i^* = \frac{E}{V_C}\left[\frac{(AL)^2 + 12(AI)}{AL + 4\left(\frac{I}{L}\right)}\right]$	$G_{12}^* = \frac{E}{V_C}\left[\frac{3}{8}(AL) + \frac{36}{8}\left(\frac{I}{L}\right)\right]$	$v_{ij}^* = \frac{\frac{1}{3}(AL) - 4\left(\frac{I}{L}\right)}{AL + 4\left(\frac{I}{L}\right)}$

Table 8.2 Numerical results of the obtained closed forms and MSC.Marc Mentat

Unit cell	**Parameters**			**Effective Young's modulus**		**Effective shear modulus**		**Effective Poisson's ratio**	
	E	L	T	Closed form	MSC.Marc Mentat	Closed form	MSC.Marc Mentat	Closed form	MSC.Marc Mentat
Square	1.0	1.0	0.1	0.1000	0.1000	0.0005	0.0005	0.0000	0.0000
	1.0	1.0	0.2	0.2000	0.2000	0.0040	0.0040	0.0000	0.0000
Triangle	1.0	1.0	0.1	0.1162	0.1162	0.0437	0.0437	0.3289	0.3289
	1.0	1.0	0.2	0.2370	0.2370	0.0901	0.0901	0.3158	0.3158

8.7 Discussions

In this chapter, the obtained program is used to determine the effective elastic constants of frame-like periodic cellular solids that can be modelled accurately by using Euler beam elements. The program determines the closed-form solutions of strain energy of unit cells of frame-like periodic cellular solids under various strain modes. These closed-form strain energy expressions are then used to determine the closed-form effective elastic constants. The program can successfully yield the closed forms of effective elastic constants. The numerical results obtained from the closed forms compare satisfactorily with those numerical results from commercial FE software.

Chapter 9
Conclusions

This study proposes a symbolic-numerical object-oriented FE program. The program is completely developed in MATLAB programming language to handle both symbolic and numerical computations. The obtained program can only be used to perform FEA of linear elastic problems. The input of the program is in the text format. The main output is also in the text format and the output in the graphical format can be added as required. The program is designed using the OOP concept and employs the OOP capability in MATLAB. The symbolic computations in the program rely on the symbolic computing capability of MATLAB. The OOP concept allows the program to be implemented efficiently, exactly in the same way as ordinary numerical FE programs.

The obtained program can be used to determine the closed-form solutions of simple structures such as trusses and frames. Examples of its advantages include those shown below:

- The program can be used to determine the exact mathematical expressions or formulas of some responses of simple structures when closed-form solutions are required or preferred. The effects and relationships of parameters found in closed-form solutions can facilitate better understanding of the behavior of structures under loads. In addition, the obtained closed-form solutions can also be used in other engineering tasks, such as structural design and optimization. Consequently, the program can be a useful tool for researches in the fields of computational mechanics and structural engineering.
- The program can help students and teachers better understand FEM by investigating the FE derivation processes in the program and symbolic analysis results from the symbolic computations.

In this study, the advantages of the obtained program are shown by using it to solve several engineering problems analytically. The obtained results are found to be satisfactory.

References

Yew, C. K., Boyle, J. T., and MacKenzie, D. (1995). "Closed form integration of element stiffness matrices using a computer algebra system." Computers & Structures, 56(4), 529-539.

Jiang, Y., and Wang, C. (2008). "On teaching finite element method in plasticity with Mathematica." Computer Applications in Engineering Education, 16(3), 233-242.

Shiakolas, P. S., Lawrence, K. L., and Nambiar, R. V. (1994). "Closed-form expressions for the linear and quadratic strain tetrahedral finite elements." Computers & Structures, 50(6), 743-747.

Zhang, W., Dai, G., Wang, F., Sun, S., and Bassir, H. (2007). "Using strain energy-based prediction of effective elastic properties in topology optimization of material microstructures." Acta Mechanica Sinica, 23(1), 77-89.

Dai, G., and Zhang, W. (2009). "Cell size effect analysis of the effective Young's modulus of sandwich core." Computational Materials Science, 46(3), 744-748.

Wang, H., Zhang, W., Xu, Y., and Zeng, Q. (2008). "Numerical computing and experimental validation of effective elastic properties of 2D multilayered C/SiC composites." Materials Science and Technology, 24(11), 1385-1398.

Forde, B. W. R., Foschi, R. O., and Stiemer, S. F. (1990). "Object-oriented finite element analysis." Comput Struct, 34(3), 355-374.

Abdalla, J. A., and Yoon, C. J. (1992). "Object-Oriented Finite Element and Graphics Data-Translation Facility." Journal of Computing in Civil Engineering, 6(3), 302-322.

Mackie, R. I. (1998). "An object-oriented approach to fully interactive finite element software." Advances in Engineering Software, 29(2), 139-149.

Archer, G. C., Fenves, G., and Thewalt, C. (1999). "A new object-oriented finite element analysis program architecture." Computers & Structures, 70(1), 63-75.

Mackie, R. I. (2002). "Using objects to handle calculation control in finite element modelling." Computers & Structures, 80(27-30), 2001-2009.

Mackie, R. I. (2007). "Object oriented implementation of distributed finite element analysis in .NET." Advances in Engineering Software, 38(11-12), 726-737.

Pavlovic, M. N. (2003). "Symbolic computation in structural engineering." Computers & Structures, 81(22-23), 2121-2136.

Shiakolas, P. S., Lawrence, K. L., and Nambiar, R. V. (1993). "Closed-form error estimators for the linear strain and quadratic strain tetrahedron finite elements." Computers & Structures, 47(6), 907-915.

Mackerle, J. (2000). "Object-oriented techniques in FEM and BEM A bibliography (1996–1999)." Finite Elements in Analysis and Design, 36(2), 189-196.

Kwon, Y. W., and Bang, H. (2000). The finite element method using MATLAB, CRC press.

Patzák, B., and Bittnar, Z. (2001). "Design of object oriented finite element code." Advances in Engineering Software, 32(10–11), 759-767.

Akin, J. E., and Singh, M. (2002). "Object-oriented Fortran 90 P-adaptive finite element method." Advances in Engineering Software, 33(7–10), 461-468.

Martha, L. F., and Parente Jr, E. "An object-oriented framework for finite element programming." Proc., The Fifth World Congress on Computational Mechanics, 1-10.

Heng, B. C. P., and Mackie, R. I. (2009). "Using design patterns in object-oriented finite element programming." Computers & Structures, 87(15–16), 952-961.

Piedade Neto, D., Ferreira, M. D. C., and Proença, S. P. B. (2013). "An object-oriented class design for the generalized finite element method programming." Latin American Journal of Solids and Structures, 10(6), 1267-1291.

Alves, P. D., Barros, F. c. B., and Pitangueira, R. L. S. (2013). "An object-oriented approach to the Generalized Finite Element Method." Advances in Engineering Software, 59(0), 1-18.

Rahman, T., and Valdman, J. (2013). "Fast MATLAB assembly of FEM matrices in 2D and 3D: Nodal elements." Applied Mathematics and Computation, 219(13), 7151-7158.

Zander, N., Bog, T., Elhaddad, M., Espinoza, R., Hu, H., Joly, A., Wu, C., Zerbe, P., Düster, A., Kollmannsberger, S., Parvizian, J., Ruess, M., Schillinger, D., and Rank, E. (2014). "FCMLab: A finite cell research toolbox for MATLAB." Advances in Engineering Software, 74(0), 49-63.

Zimmermann, T., and Eyheramendy, D. (1996). "Object-oriented finite elements I. Principles of symbolic derivations and automatic programming." Computer Methods in Applied Mechanics and Engineering, 132(3-4), 259-276.

Eyheramendy, D., and Zimmermann, T. (1996). "Object-oriented finite elements II. A symbolic environment for automatic programming." Computer Methods in Applied Mechanics and Engineering, 132(3-4), 277-304.

Eyheramendy, D., and Zimmermann, T. (1996). "Object-oriented finite element programming: an interactive environment for symbolic derivations, application to an initial boundary value problem." Advances in Engineering Software, 27(1–2), 3-10.

Eyheramendy, D., and Zimmermann, T. (1998). "Object-oriented finite elements III. Theory and application of automatic programming." Computer Methods in Applied Mechanics and Engineering, 154(1-2), 41-68.

Eyheramendy, D., and Zimmermann, T. (2001). "Object-oriented finite elements IV. Symbolic derivations and automatic programming of nonlinear formulations." Computer Methods in Applied Mechanics and Engineering, 190(22-23), 2729-2751.

Cheng, K. J. (1991). "Symbolic finite element analysis using computer algebra: Heat transfer in rectangular duct flow." Computers & Mathematics with Applications, 22(12), 15-22.

Tummarakota, S., and Lieh, J. (1996). "Symbolic Finite Element Modeling of Structural Systems." Journal of Symbolic Computation, 22(1), 105-119.

Cameron, F. (1997). "Automatic generation of efficient routines for evaluating multivariate polynomials arising in finite element computations." Advances in Engineering Software, 28(4), 239-245.

Korelc, J. (1997). "Automatic generation of finite-element code by simultaneous optimization of expressions." Theoretical Computer Science, 187(1-2), 231-248.

Lee, C. K., and Hobbs, R. E. (1998). "Closed form stiffness matrix solutions for some commonly used hybrid finite elements." Computers & Structures, 67(6), 463-482.

Zimmermann, T., Bomme, P., Eyheramendy, D., Vernier, L., and Commend, S. (1998). "Aspects of an object-oriented finite element environment." Computers & Structures, 68(1–3), 1-16.

Eriksson, A., and Pacoste, C. (1999). "Symbolic software tools in the development of finite elements." Computers & Structures, 72(4–5), 579-593.

Eyheramendy, D. (2000). "An object-oriented hybrid symbolic/numerical approach for the development of finite element codes." Finite Elements in Analysis and Design, 36(3–4), 315-334.

McCaslin, S. E., Shiakolas, P. S., Dennis, B. H., and Lawrence, K. L. (2012). "Closed-form stiffness matrices for higher order tetrahedral finite elements." Advances in Engineering Software, 44(1), 75-79.

Glasser, M. (2009). "Fundamentals of Object-Oriented Programming." Open Verification Methodology Cookbook, Springer New York, 27-48.

Rumbaugh, J., Jacobson, I., and Booch, G. (2004). Unified Modeling Language Reference Manual, The, Pearson Higher Education.

Gibson, L. J., and Ashby, M. F. (1999). Cellular solids: structure and properties, Cambridge university press.

Christensen, P. W., and Klarbring, A. (2008). An introduction to structural optimization, Springer Science & Business Media.

Deb, K., and Gulati, S. (2001). "Design of truss-structures for minimum weight using genetic algorithms." Finite Elements in Analysis and Design, 37(5), 447-465.

Luh, G.-C., and Lin, C.-Y. (2008). "Optimal design of truss structures using ant algorithm." Struct Multidisc Optim, 36(4), 365-379.

Farshi, B., and Alinia-ziazi, A. (2010). "Sizing optimization of truss structures by method of centers and force formulation." International Journal of Solids and Structures, 47(18–19), 2508-2524.

Li, L. J., Huang, Z. B., Liu, F., and Wu, Q. H. (2007). "A heuristic particle swarm optimizer for optimization of pin connected structures." Computers & Structures, 85(7–8), 340-349.

Suquet, P. M. (1987). "Elements of Homogenization for Inelastic Solid Mechanics." Homogenization Techniques for Composite Media, Springer-Verlag, 193-278.

Theerakittayakorn, K., and Nanakorn, P. "Periodic boundary conditions for unit cells of periodic cellular solids in the determination of effective properties using beam elements." Proc., The 2013 World Congress on Advances in Structural Engineering and Mechanics, 3738-3748.

Appendices

Appendix A

Inputs of the Program

2DTruss_Input

```
FEMLAB
Comments:

   5-----4
     \ /  \
     / \    \
   1----2|--|3
         |P1|P2

Node
1    0      0     0
2    L      0     0
3    2*L    0     0
4    L      H     0
5    0      H     0

ElementTable
1 ETrussL2_2D

Element
1   1 1 1    1 2
2   1 1 2    2 3
3   1 1 3    3 4
4   1 1 4    4 5
5   1 1 5    1 4
6   1 1 6    2 5

Material
1   MLElasIso  Ee  0

Geometry
1   GLine  A1   0   0   0
2   GLine  A2   0   0   0
3   GLine  A3   0   0   0
4   GLine  A4   0   0   0
5   GLine  A5   0   0   0
6   GLine  A6   0   0   0
```

```
BounC
1    1    1
2    0    0
3    0    0
4    0    0
5    1    1

Forces
2    0   P
3    0   P

MultiFreedomConst
%coef1*(u_dof1_of_node1)+coef2(u_dof2_of_node2)  = B
%node1    node2    dof1    dof2    coef1    coef2     B

End
```

3DTruss_Input

```
FEMLAB
Comments:
Figure is not here

Node
1    0         0         0
2    L         0         0
3    L         0        -L
4    0         0        -L
5    0         H         0
6    L         H         0
7    L         H        -L
8    0         H        -L
9    L/2       H/2      -L/2

ElementTable
1 ETrussL2

Element
1    1  1  1    1   9
2    1  1  1    2   9
3    1  1  1    3   9
4    1  1  1    4   9
5    1  1  1    5   9
6    1  1  1    6   9
7    1  1  1    7   9
8    1  1  1    8   9
```

```
Material
1   MLElasIso   Ee  0

Geometry
1   GLine  A1   0   0   0
2   GLine  A2   0   0   0
3   GLine  A3   0   0   0
4   GLine  A4   0   0   0
5   GLine  A5   0   0   0
6   GLine  A6   0   0   0
7   GLine  A7   0   0   0
8   GLine  A8   0   0   0

BounC
1     1    1    1
2     1    1    1
3     1    1    1
4     1    1    1
5     1    1    1
6     1    1    1
7     1    1    1
8     1    1    1
9     0    0    0

Forces
9    0    P    0

MultiFreedomConst
%coef1*(u_dof1_of_node1)+coef2(u_dof2_of_node2)  = B
%node1    node2    dof1    dof2    coef1    coef2     B

End
```

2DFrame_Input

```
FEMLAB
Comments:

  4 -------- 3 ---> Dx
   |         |
   |         |
  1 -------- 2 ---> Dx

Node
1     0    0    0
2     L    0    0
3     L    H    0
4     0    H    0
```

```
ElementTable
1 EEulerBeamL2_2D

Element
1   1 1 1    1 2
2   1 1 2    2 3
3   1 1 3    3 4
4   1 1 4    4 1

Material
1   MLElasIso   Ee  0

Geometry
1   GLine  A1   Iz1   0   0
2   GLine  A2   Iz2   0   0
3   GLine  A3   Iz3   0   0
4   GLine  A4   Iz4   0   0

BounC
1     1  1  1
2    -1  1  1
3    -1  1  1
4     1  1  1

Forces
2    Dx  0  0
3    Dx  0  0

MultiFreedomConst
%coef1*(u_dof1_of_node1)+coef2(u_dof2_of_node2)  = B
%node1    node2    dof1    dof2    coef1    coef2     B

End
```

3DFrame_Input

```
FEMLAB
Comments:

    8_______7-----> Dx
   5/|_____6/|--> Dx
   |4/     |  |3----> Dx
   |/_____|/---> Dx
   1       2
```

```
Node
1    0    0    0
2    L    0    0
3    L    0   -L
4    0    0   -L
5    0    H    0
6    L    H    0
7    L    H   -L
8    0    H   -L

ElementTable
1 EEulerBeamL2

Element
1    1  1  1     1  2
2    1  1  2     2  3
3    1  1  3     3  4
4    1  1  4     4  1
5    1  1  5     5  6
6    1  1  6     6  7
7    1  1  7     7  8
8    1  1  8     8  5
9    1  1  9     1  5
10   1  1  10    2  6
11   1  1  11    3  7
12   1  1  12    4  8


Material
1    MLElasIso    Ee   0

Geometry
1     GLine   A1    Iz1    Iy1    0
2     GLine   A2    Iz2    Iy2    0
3     GLine   A3    Iz3    Iy3    0
4     GLine   A4    Iz4    Iy4    0
5     GLine   A5    Iz5    Iy5    0
6     GLine   A6    Iz6    Iy6    0
7     GLine   A7    Iz7    Iy7    0
8     GLine   A8    Iz8    Iy8    0
9     GLine   A9    Iz9    Iy9    0
10    GLine   A10   Iz10   Iy10   0
11    GLine   A11   Iz11   Iy11   0
12    GLine   A12   Iz12   Iy12   0
```

```
BounC
1     1    1    1    1    1    1
2    -1    1    1    1    1    1
3    -1    1    1    1    1    1
4     1    1    1    1    1    1
5     1    1    1    1    1    1
6    -1    1    1    1    1    1
7    -1    1    1    1    1    1
8     1    1    1    1    1    1

Forces
2    Dx    0    0    0    0    0
3    Dx    0    0    0    0    0
6    Dx    0    0    0    0    0
7    Dx    0    0    0    0    0

MultiFreedomConst
%coef1*(u_dof1_of_node1)+coef2(u_dof2_of_node2) = B
%node1   node2   dof1   dof2   coef1   coef2    B

End
```

Appendix B

Outputs of the Program

```
2DTruss_Output

I.  Nodal Displacements:
    Node number: 1
    Q = 0
    Q = 0
    Node number: 2
    Q = (2*L^2*P)/(A1*Ee*H)
    Q = (P*(2*A6*L^3 + A1*(H^2 + L^2)^(3/2)))/(A1*A6*Ee*H^2)
    Node number: 3
    Q = (L^2*P*(A1 + 2*A2))/(A1*A2*Ee*H)
    Q = (P*(4*A1*A2*A3*A5*L^3 + A1*A3*A4*A5*L^3 + 2*A2*A3*A4*A5*L^3 +...
        A1*A2*A3*A4*(H^2 + L^2)^(3/2) + A1*A2*A4*A5*(H^2 + L^2)^(3/2)))/...
        (A1*A2*A3*A4*A5*Ee*H^2)
    Node number: 4
    Q = -(2*L^2*P)/(A4*Ee*H)
    Q = (P*(2*A5*L^3 + A4*(H^2 + L^2)^(3/2)))/(A4*A5*Ee*H^2)
    Node number: 5
    Q = 0
    Q = 0

II. Supported Reactions:
    Node number: 1
    R = -(3*L*P)/H
    R = -P
    Node number: 2
    R = 0
    R = 0
    Node number: 3
    R = 0
    R = 0
    Node number: 4
    R = 0
    R = 0
    Node number: 5
    R = (3*L*P)/H
    R = -P

III.Total Strain Energy:
    U = (P^2*(4*A1*A2*A3*A5*A6*L^3 + A1*A3*A4*A5*A6*L^3 +...
        4*A2*A3*A4*A5*A6*L^3 + A1*A2*A3*A4*A5*(H^2 + L^2)^(3/2) +...
        A1*A2*A3*A4*A6*(H^2 + L^2)^(3/2) +...
        A1*A2*A4*A5*A6*(H^2 + L^2)^(3/2)))/(2*A1*A2*A3*A4*A5*A6*Ee*H^2)
```

```
IV. Strains in local coordinates:
    Element number: 1
    e = (2*L*P)/(A1*Ee*H)
    Element number: 2
    e = (L*P)/(A2*Ee*H)
    Element number: 3
    e = -(P*(H^2 + L^2)^(1/2))/(A3*Ee*H)
    Element number: 4
    e = -(2*L*P)/(A4*Ee*H)
    Element number: 5
    e = (P*(H^2 + L^2)^(1/2))/(A5*Ee*H)
    Element number: 6
    e = -(P*(H^2 + L^2)^(1/2))/(A6*Ee*H)
V.  Stresses in local coordinates:
    Element number: 1
    S = (2*L*P)/(A1*H)
    Element number: 2
    S = (L*P)/(A2*H)
    Element number: 3
    S = -(P*(H^2 + L^2)^(1/2))/(A3*H)
    Element number: 4
    S = -(2*L*P)/(A4*H)
    Element number: 5
    S = (P*(H^2 + L^2)^(1/2))/(A5*H)
    Element number: 6
    S = -(P*(H^2 + L^2)^(1/2))/(A6*H)
```

3DTruss_Output

```
I.  Nodal Displacements:
    Node number: 1
    Q = 0
    Q = 0
    Q = 0
    Node number: 2
    Q = 0
    Q = 0
    Q = 0
    Node number: 3
    Q = 0
    Q = 0
    Q = 0
    Node number: 4
    Q = 0
    Q = 0
    Q = 0
    Node number: 5
    Q = 0
    Q = 0
    Q = 0
    Node number: 6
    Q = 0
    Q = 0
    Q = 0
    Node number: 7
    Q = 0
    Q = 0
    Q = 0
    Node number: 8
    Q = 0
    Q = 0
    Q = 0
    Node number: 9
    Q = 0
    Q = (P*(H^2 + 2*L^2)^(3/2))/(16*A1*Ee*H^2)
    Q = 0

II. Supported Reactions:
    Node number: 1
    R = -(L*P)/(8*H)
    R = -P/8
    R = (L*P)/(8*H)
    Node number: 2
    R = (L*P)/(8*H)
    R = -P/8
    R = (L*P)/(8*H)
    Node number: 3
    R = (L*P)/(8*H)
    R = -P/8
    R = -(L*P)/(8*H)
    Node number: 4
    R = -(L*P)/(8*H)
    R = -P/8
    R = -(L*P)/(8*H)
    Node number: 5
    R = (L*P)/(8*H)
    R = -P/8
    R = -(L*P)/(8*H)
    Node number: 6
    R = -(L*P)/(8*H)
    R = -P/8
    R = -(L*P)/(8*H)
```

```
     Node number: 7
     R = -(L*P)/(8*H)
     R = -P/8
     R = (L*P)/(8*H)
     Node number: 8
     R = (L*P)/(8*H)
     R = -P/8
     R = (L*P)/(8*H)
     Node number: 9
     R = 0
     R = 0
     R = 0
III.Total Strain Energy:
     U = (P^2*(H^2 + 2*L^2)^(3/2))/(32*A1*Ee*H^2)
IV.  Strains in local coordinates:
     Element number: 1
     e = (P*(H^2 + 2*L^2)^(1/2))/(8*A1*Ee*H)
     Element number: 2
     e = (P*(H^2 + 2*L^2)^(1/2))/(8*A1*Ee*H)
     Element number: 3
     e = (P*(H^2 + 2*L^2)^(1/2))/(8*A1*Ee*H)
     Element number: 4
     e = (P*(H^2 + 2*L^2)^(1/2))/(8*A1*Ee*H)
     Element number: 5
     e = -(P*(H^2 + 2*L^2)^(1/2))/(8*A1*Ee*H)
     Element number: 6
     e = -(P*(H^2 + 2*L^2)^(1/2))/(8*A1*Ee*H)
     Element number: 7
     e = -(P*(H^2 + 2*L^2)^(1/2))/(8*A1*Ee*H)
     Element number: 8
     e = -(P*(H^2 + 2*L^2)^(1/2))/(8*A1*Ee*H)
V.   Stresses in local coordinates:
     Element number: 1
     S = (P*(H^2 + 2*L^2)^(1/2))/(8*A1*H)
     Element number: 2
     S = (P*(H^2 + 2*L^2)^(1/2))/(8*A1*H)
     Element number: 3
     S = (P*(H^2 + 2*L^2)^(1/2))/(8*A1*H)
     Element number: 4
     S = (P*(H^2 + 2*L^2)^(1/2))/(8*A1*H)
     Element number: 5
     S = -(P*(H^2 + 2*L^2)^(1/2))/(8*A1*H)
     Element number: 6
     S = -(P*(H^2 + 2*L^2)^(1/2))/(8*A1*H)
     Element number: 7
     S = -(P*(H^2 + 2*L^2)^(1/2))/(8*A1*H)
     Element number: 8
     S = -(P*(H^2 + 2*L^2)^(1/2))/(8*A1*H)
```

2DFrame_Output

```
I.  Nodal Displacements:
    Node number: 1
    Q = 0
    Q = 0
    Q = 0
    Node number: 2
    Q = Dx
    Q = 0
    Q = 0
    Node number: 3
    Q = Dx
    Q = 0
    Q = 0
    Node number: 4
    Q = 0
    Q = 0
    Q = 0
II. Supported Reactions:
    Node number: 1
    R = -(A1*Dx*Ee)/L
    R = 0
    R = 0
    Node number: 2
    R = (A1*Dx*Ee)/L
    R = 0
    R = 0
    Node number: 3
    R = (A3*Dx*Ee)/L
    R = 0
    R = 0
    Node number: 4
    R = -(A3*Dx*Ee)/L
    R = 0
    R = 0
III.Total Strain Energy:
    U = (Dx^2*Ee*(A1 + A3))/(2*L)
IV. Internal forces in local coordinates:
    Element number: 1
    F = -(A1*Dx*Ee)/L
    F = 0
    F = 0
    F = (A1*Dx*Ee)/L
    F = 0
    F = 0
```

```
     Element number: 2
     F = 0
     F = 0
     F = 0
     F = 0
     F = 0
     F = 0
     Element number: 3
     F = -(A3*Dx*Ee)/L
     F = 0
     F = 0
     F = (A3*Dx*Ee)/L
     F = 0
     F = 0
     Element number: 4
     F = 0
     F = 0
     F = 0
     F = 0
     F = 0
     F = 0
```

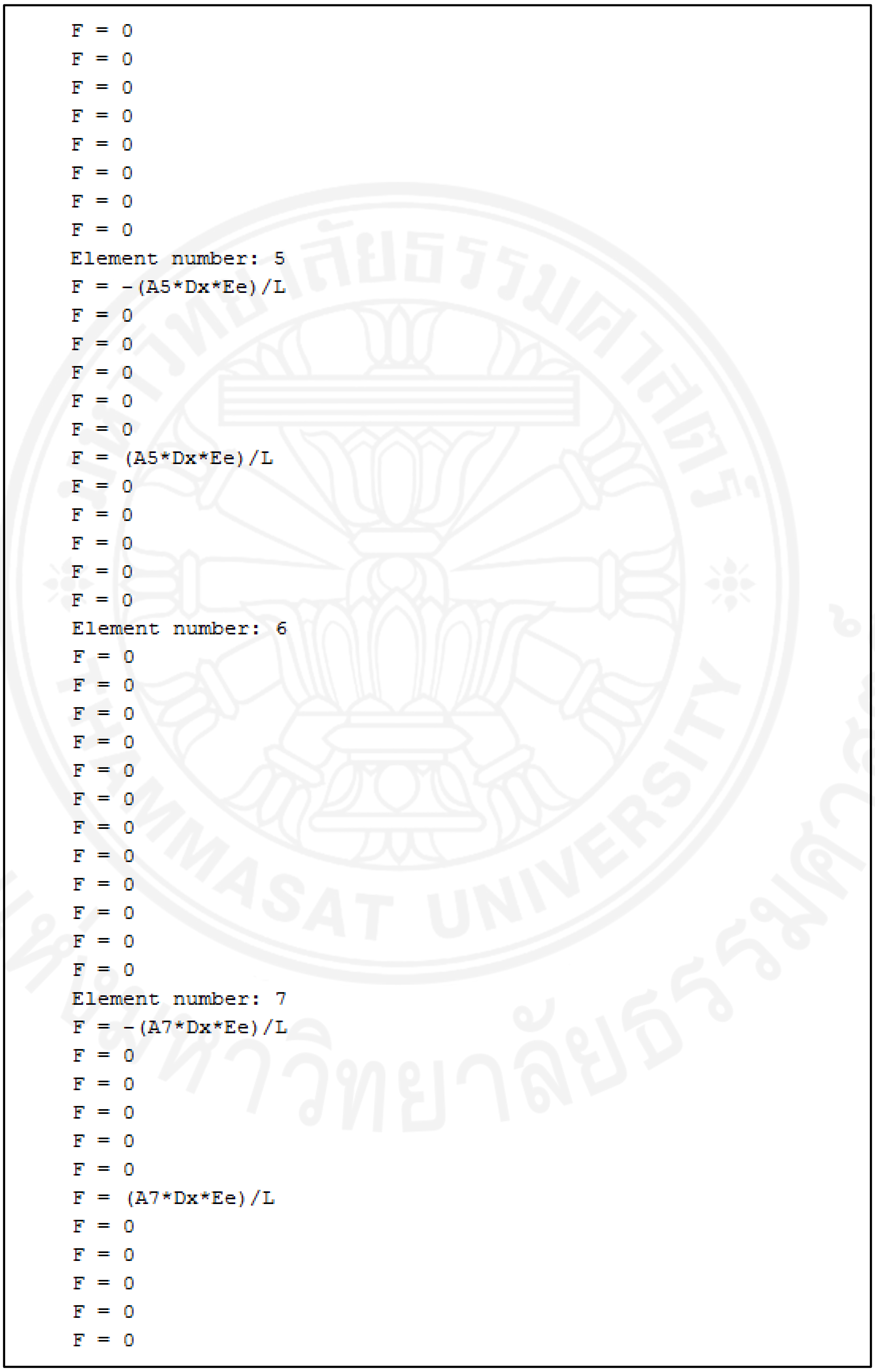
3DFrame_Output

```
I.   Nodal Displacements:
     Node number: 1
     Q = 0
     Q = 0
     Q = 0
     Q = 0
     Q = 0
     Q = 0
     Node number: 2
     Q = Dx
     Q = 0
     Q = 0
     Q = 0
     Q = 0
     Q = 0
     Node number: 3
     Q = Dx
     Q = 0
     Q = 0
     Q = 0
     Q = 0
     Q = 0
```

```
    Node number: 4
    Q = 0
    Q = 0
    Q = 0
    Q = 0
    Q = 0
    Q = 0
    Node number: 5
    Q = 0
    Q = 0
    Q = 0
    Q = 0
    Q = 0
    Q = 0
    Node number: 6
    Q = Dx
    Q = 0
    Q = 0
    Q = 0
    Q = 0
    Q = 0
    Node number: 7
    Q = Dx
    Q = 0
    Q = 0
    Q = 0
    Q = 0
    Q = 0
    Node number: 8
    Q = 0
    Q = 0
    Q = 0
    Q = 0
    Q = 0
    Q = 0
II. Supported Reactions:
    Node number: 1
    R = -(A1*Dx*Ee)/L
    R = 0
    R = 0
    R = 0
    R = 0
    R = 0
    Node number: 2
    R = (A1*Dx*Ee)/L
    R = 0
```

```
R = 0
R = 0
R = 0
R = 0
Node number: 3
R = (A3*Dx*Ee)/L
R = 0
R = 0
R = 0
R = 0
R = 0
Node number: 4
R = -(A3*Dx*Ee)/L
R = 0
R = 0
R = 0
R = 0
R = 0
Node number: 5
R = -(A5*Dx*Ee)/L
R = 0
R = 0
R = 0
R = 0
R = 0
Node number: 6
R = (A5*Dx*Ee)/L
R = 0
R = 0
R = 0
R = 0
R = 0
Node number: 7
R = (A7*Dx*Ee)/L
R = 0
R = 0
R = 0
R = 0
R = 0
Node number: 8
R = -(A7*Dx*Ee)/L
R = 0
R = 0
R = 0
R = 0
R = 0
```

```
III.Total Strain Energy:
    U = (Dx^2*Ee*(A1 + A3 + A5 + A7))/(2*L)
IV. Internal forces in local coordinates:
    Element number: 1
    F = -(A1*Dx*Ee)/L
    F = 0
    F = 0
    F = 0
    F = 0
    F = 0
    F = (A1*Dx*Ee)/L
    F = 0
    F = 0
    F = 0
    F = 0
    F = 0
    Element number: 2
    F = 0
    F = 0
    F = 0
    F = 0
    F = 0
    F = 0
    F = 0
    F = 0
    F = 0
    F = 0
    F = 0
    F = 0
    Element number: 3
    F = -(A3*Dx*Ee)/L
    F = 0
    F = 0
    F = 0
    F = 0
    F = 0
    F = (A3*Dx*Ee)/L
    F = 0
    F = 0
    F = 0
    F = 0
    F = 0
    Element number: 4
    F = 0
    F = 0
    F = 0
    F = 0
```



```
F = 0
F = 0
F = 0
F = 0
F = 0
F = 0
F = 0
F = 0
Element number: 5
F = -(A5*Dx*Ee)/L
F = 0
F = 0
F = 0
F = 0
F = 0
F = (A5*Dx*Ee)/L
F = 0
F = 0
F = 0
F = 0
F = 0
Element number: 6
F = 0
F = 0
F = 0
F = 0
F = 0
F = 0
F = 0
F = 0
F = 0
F = 0
F = 0
F = 0
Element number: 7
F = -(A7*Dx*Ee)/L
F = 0
F = 0
F = 0
F = 0
F = 0
F = (A7*Dx*Ee)/L
F = 0
F = 0
F = 0
F = 0
F = 0
```

```
Element number: 8
F = 0
F = 0
F = 0
F = 0
F = 0
F = 0
F = 0
F = 0
F = 0
F = 0
F = 0
F = 0
Element number: 9
F = 0
F = 0
F = 0
F = 0
F = 0
F = 0
F = 0
F = 0
F = 0
F = 0
F = 0
F = 0
Element number: 10
F = 0
F = 0
F = 0
F = 0
F = 0
F = 0
F = 0
F = 0
F = 0
F = 0
F = 0
F = 0
Element number: 11
F = 0
F = 0
F = 0
F = 0
F = 0
F = 0
F = 0
```

```
F = 0
F = 0
F = 0
F = 0
F = 0
Element number: 12
F = 0
F = 0
F = 0
F = 0
F = 0
F = 0
F = 0
F = 0
F = 0
F = 0
F = 0
F = 0
```

Appendix C
Numerical Comparisons

For trusses, the numerical results are obtained by substituting the numerical values of the variables A_i = 0.006 m^2, P_i = 2 kN, L = 3 m, H = 3 m, and E_e = 2 x 10^8 kPa. For frames, the numerical results are obtained from these numerical variables A_i = 0.04 m^2, I_i = 10^{-4}/7500 m^2, D_x = 0.01 m, L = 3 m, H = 3 m, and E_e = 2 x 10^8 kPa.

Structures		**Closed Forms**	**MSC.Marc Mentat**
Types	Node	Nodal Displacements [m]	
2D Truss	1	Q_x = 0	Q_x = 0
		Q_y = 0	Q_y = 0
	2	Q_x = -0.00001	Q_x = -0.00001
		Q_y = -0.0000241421	Q_y = -0.0000241421
	3	Q_x = -0.000015	Q_x = -0.000015
		Q_y = -0.0000632843	Q_y = -0.0000632843
	4	Q_x = 0.00001	Q_x = 0.00001
		Q_y = -0.0000241421	Q_y = -0.0000241421
	5	Q_x = 0	Q_x = 0
		Q_y = 0	Q_y = 0
	Node	Support Reactions [kN]	
	1	R_x = 6	R_x = 6
		R_y = 2	R_y = 2
	5	R_x = -6	R_x = -6
		R_y = 2	R_y = 2
	Element	Total Strain Energy [kN.m]	
	All	U = 0.0000874264	U = 0.0000874264
	Element	Strains [m/m]	
	1	e = -0.00000333333	e = -0.00000333333
	2	e = -0.00000166667	e = -0.00000166667

	3	e = 0.00000235702	e = 0.00000235702
	4	e = 0.00000333333	e = 0.00000333333
	5	e = -0.00000235702	e = -0.00000235702
	6	e = 0.00000235702	e = 0.00000235702
	Element	Stresses [kN/m^2]	
	1	s = -666.667	s = -666.667
	2	s = -333.333	s = -333.333
	3	s = 471.405	s = 471.405
	4	s = 666.667	s = 666.667
	5	s = -471.405	s = -471.405
	6	s = 471.405	s = 471.405
Types	Node	Nodal Displacements [m]	
3D Truss	1	Q_x = 0	Q_x = 0
		Q_y = 0	Q_y = 0
		Q_z = 0	Q_z = 0
	2	Q_x = 0	Q_x = 0
		Q_y = 0	Q_y = 0
		Q_z = 0	Q_z = 0
	3	Q_x = 0	Q_x = 0
		Q_y = 0	Q_y = 0
		Q_z = 0	Q_z = 0
	4	Q_x = 0	Q_x = 0
		Q_y = 0	Q_y = 0
		Q_z = 0	Q_z = 0
	5	Q_x = 0	Q_x = 0
		Q_y = 0	Q_y = 0
		Q_z = 0	Q_z = 0
	6	Q_x = 0	Q_x = 0
		Q_y = 0	Q_y = 0
		Q_z = 0	Q_z = 0

	7	Q_x = 0	Q_x = 0
		Q_y = 0	Q_y = 0
		Q_z = 0	Q_z = 0
	8	Q_x = 0	Q_x = 0
		Q_y = 0	Q_y = 0
		Q_z = 0	Q_z = 0
	9	Q_x = 0	Q_x = 0
		Q_y = -0.000001628	Q_y = -0.000001628
		Q_z = 0	Q_z = 0
	Node	Support Reactions [kN]	
	1	R_x = 0.25	R_x = 0.25
		R_y = 0.25	R_y = 0.25
		R_z = -0.25	R_z = -0.25
	2	R_x = -0.25	R_x = -0.25
		R_y = 0.25	R_y = 0.25
		R_z = -0.25	R_z = -0.25
	3	R_x = -0.25	R_x = -0.25
		R_y = 0.25	R_y = 0.25
		R_z = 0.25	R_z = 0.25
	4	R_x = 0.25	R_x = 0.25
		R_y = 0.25	R_y = 0.25
		R_z = 0.25	R_z = 0.25
	5	R_x = -0.25	R_x = -0.25
		R_y = 0.25	R_y = 0.25
		R_z = 0.25	R_z = 0.25
	6	R_x = 0.25	R_x = 0.25
		R_y = 0.25	R_y = 0.25
		R_z = 0.25	R_z = 0.25
	7	R_x = 0.25	R_x = 0.25
		R_y = 0.25	R_y = 0.25

		R_z = -0.25	R_z = -0.25
	8	R_x = -0.25	R_x = -0.25
		R_y = 0.25	R_y = 0.25
		R_z = -0.25	R_z = -0.25
	Element	Total Strain Energy [kN.m]	
	All	U = 0.0000016238	U = 0.0000016238
	Element	Strains [m/m]	
	1	e = -0.000000360844	e = -0.000000360844
	2	e = -0.000000360844	e = -0.000000360844
	3	e = -0.000000360844	e = -0.000000360844
	4	e = -0.000000360844	e = -0.000000360844
	5	e = 0.000000360844	e = 0.000000360844
	6	e = 0.000000360844	e = 0.000000360844
	7	e = 0.000000360844	e = 0.000000360844
	8	e = 0.000000360844	e = 0.000000360844
	Element	Stresses [kN/m^2]	
	1	s = -72.1688	s = -72.1688
	2	s = -72.1688	s = -72.1688
	3	s = -72.1688	s = -72.1688
	4	s = -72.1688	s = -72.1688
	5	s = 72.1688	s = 72.1688
	6	s = 72.1688	s = 72.1688
	7	s = 72.1688	s = 72.1688
	8	s = 72.1688	s = 72.1688
Types	Node	Nodal Displacements [m]	
2D Frame	1	Q_x = 0	Q_x = 0
		Q_y = 0	Q_y = 0
		Q_r = 0	Q_r = 0
	2	Q_x = 0.01	Q_x = 0.01
		Q_y = 0	Q_y = 0

		Q_r = 0	Q_r = 0
	3	Q_x = 0.01	Q_x = 0.01
		Q_y = 0	Q_y = 0
		Q_r = 0	Q_r = 0
	4	Q_x = 0	Q_x = 0
		Q_y = 0	Q_y = 0
		Q_r = 0	Q_r = 0
	Node	Support Reactions [kN]	
	1	R_x = -26666.3	R_x = -26666.3
		R_y = 0	R_y = 0
		R_r = 0	R_r = 0
	2	R_x = 26666.3	R_x = 26666.3
		R_y = 0	R_y = 0
		R_r = 0	R_r = 0
	3	R_x = 26666.3	R_x = 26666.3
		R_y = 0	R_y = 0
		R_r = 0	R_r = 0
	4	R_x = -26666.3	R_x = -26666.3
		R_y = 0	R_y = 0
		R_r = 0	R_r = 0
	Element	Total Strain Energy [kN.m]	
	All	U = 266.667	U = 266.667
	Element	Internal Forces [kN]	
	1	N = 26666.7	N = 26666.7
		V = 0	V =
		M = 0	M =
	2	N = 0	N = 0
		V = 0	V = 0
		M = 0	M = 0
	3	N = 26666.7	N = 26666.7

		$V = 0$	$V = 0$
		$M = 0$	$M = 0$
	4	$N = 0$	$N = 0$
		$V = 0$	$V = 0$
		$M = 0$	$M = 0$
Types	Node	Nodal Displacements [m]	
3D Frame	1	$Q_x = 0$	$Q_x = 0$
		$Q_{rx} = 0$	$Q_{rx} = 0$
		$Q_y = 0$	$Q_y = 0$
		$Q_{ry} = 0$	$Q_{ry} = 0$
		$Q_z = 0$	$Q_z = 0$
		$Q_{rz} = 0$	$Q_{rz} = 0$
	2	$Q_x = 0.01$	$Q_x = 0.01$
		$Q_{rx} = 0$	$Q_{rx} = 0$
		$Q_y = 0$	$Q_y = 0$
		$Q_{ry} = 0$	$Q_{ry} = 0$
		$Q_z = 0$	$Q_z = 0$
		$Q_{rz} = 0$	$Q_{rz} = 0$
	3	$Q_x = 0.01$	$Q_x = 0.01$
		$Q_{rx} = 0$	$Q_{rx} = 0$
		$Q_y = 0$	$Q_y = 0$
		$Q_{ry} = 0$	$Q_{ry} = 0$
		$Q_z = 0$	$Q_z = 0$
		$Q_{rz} = 0$	$Q_{rz} = 0$
	4	$Q_x = 0$	$Q_x = 0$
		$Q_{rx} = 0$	$Q_{rx} = 0$
		$Q_y = 0$	$Q_y = 0$
		$Q_{ry} = 0$	$Q_{ry} = 0$
		$Q_z = 0$	$Q_z = 0$
		$Q_{rz} = 0$	$Q_{rz} = 0$

	5	$Q_x =$ 0	$Q_x =$ 0
		Q_{rx} = 0	Q_{rx} = 0
		$Q_y =$ 0	$Q_y =$ 0
		Q_{ry} = 0	Q_{ry} = 0
		$Q_z =$ 0	$Q_z =$ 0
		Q_{rz} = 0	Q_{rz} = 0
	6	$Q_x =$ 0.01	$Q_x =$ 0.01
		Q_{rx} = 0	Q_{rx} = 0
		$Q_y =$ 0	$Q_y =$ 0
		Q_{ry} = 0	Q_{ry} = 0
		$Q_z =$ 0	$Q_z =$ 0
		Q_{rz} = 0	Q_{rz} = 0
	7	$Q_x =$ 0.01	$Q_x =$ 0.01
		Q_{rx} = 0	Q_{rx} = 0
		$Q_y =$ 0	$Q_y =$ 0
		Q_{ry} = 0	Q_{ry} = 0
		$Q_z =$ 0	$Q_z =$ 0
		Q_{rz} = 0	Q_{rz} = 0
	8	$Q_x =$ 0	$Q_x =$ 0
		Q_{rx} = 0	Q_{rx} = 0
		$Q_y =$ 0	$Q_y =$ 0
		Q_{ry} = 0	Q_{ry} = 0
		$Q_z =$ 0	$Q_z =$ 0
		Q_{rz} = 0	Q_{rz} = 0
	Node	Support Reactions [kN]	
	1	R_x = -26666.7	R_x = -26666.7
		R_{rx} = 0	R_{rx} = 0
		$R_y =$ 0	$R_y =$ 0
		R_{ry} = 0	R_{ry} = 0
		$R_z =$ 0	$R_z =$ 0

		$R_{rz} = 0$	$R_{rz} = 0$
	2	$R_x = 26666.7$	$R_x = 26666.7$
		$R_{rx} = 0$	$R_{rx} = 0$
		$R_y = 0$	$R_y = 0$
		$R_{ry} = 0$	$R_{ry} = 0$
		$R_z = 0$	$R_z = 0$
		$R_{rz} = 0$	$R_{rz} = 0$
	3	$R_x = 26666.7$	$R_x = 26666.7$
		$R_{rx} = 0$	$R_{rx} = 0$
		$R_y = 0$	$R_y = 0$
		$R_{ry} = 0$	$R_{ry} = 0$
		$R_z = 0$	$R_z = 0$
		$R_{rz} = 0$	$R_{rz} = 0$
	4	$R_x = -26666.7$	$R_x = -26666.7$
		$R_{rx} = 0$	$R_{rx} = 0$
		$R_y = 0$	$R_y = 0$
		$R_{ry} = 0$	$R_{ry} = 0$
		$R_z = 0$	$R_z = 0$
		$R_{rz} = 0$	$R_{rz} = 0$
	5	$R_x = -26666.7$	$R_x = -26666.7$
		$R_{rx} = 0$	$R_{rx} = 0$
		$R_y = 0$	$R_y = 0$
		$R_{ry} = 0$	$R_{ry} = 0$
		$R_z = 0$	$R_z = 0$
		$R_{rz} = 0$	$R_{rz} = 0$
	6	$R_x = 26666.7$	$R_x = 26666.7$
		$R_{rx} = 0$	$R_{rx} = 0$
		$R_y = 0$	$R_y = 0$
		$R_{ry} = 0$	$R_{ry} = 0$
		$R_z = 0$	$R_z = 0$

		R_{rz} = 0	R_{rz} = 0
	7	R_x = 26666.7	R_x = 26666.7
		R_{rx} = 0	R_{rx} = 0
		R_y = 0	R_y = 0
		R_{ry} = 0	R_{ry} = 0
		R_z = 0	R_z = 0
		R_{rz} = 0	R_{rz} = 0
	8	R_x = -26666.7	R_x = -26666.7
		R_{rx} = 0	R_{rx} = 0
		R_y = 0	R_y = 0
		R_{ry} = 0	R_{ry} = 0
		R_z = 0	R_z = 0
		R_{rz} = 0	R_{rz} = 0
	Element	Total Strain Energy [kN.m]	
	All	U = 533.3333	U = 533.3333
	Element	Internal Forces [kN]	
	1	N_x = 26666.6667	N_x = 26666.6667
		M_x = 0	M_x = 0
		N_y = 0	N_y = 0
		M_y = 0	M_y = 0
		N_z = 0	N_z = 0
		M_z = 0	M_z = 0
	2	N_x = 0	N_x = 0
		M_x = 0	M_x = 0
		N_y = 0	N_y = 0
		M_y = 0	M_y = 0
		N_z = 0	N_z = 0
		M_z = 0	M_z = 0
	3	N_x = 26666.6667	N_x = 26666.6667
		M_x = 0	M_x = 0

		$N_y = 0$	$N_y = 0$
		$M_y = 0$	$M_y = 0$
		$N_z = 0$	$N_z = 0$
		$M_z = 0$	$M_z = 0$
	4	$N_x = 0$	$N_x = 0$
		$M_x = 0$	$M_x = 0$
		$N_y = 0$	$N_y = 0$
		$M_y = 0$	$M_y = 0$
		$N_z = 0$	$N_z = 0$
		$M_z = 0$	$M_z = 0$
	5	$N_x = 26666.6667$	$N_x = 26666.6667$
		$M_x = 0$	$M_x = 0$
		$N_y = 0$	$N_y = 0$
		$M_y = 0$	$M_y = 0$
		$N_z = 0$	$N_z = 0$
		$M_z = 0$	$M_z = 0$
	6	$N_x = 0$	$N_x = 0$
		$M_x = 0$	$M_x = 0$
		$N_y = 0$	$N_y = 0$
		$M_y = 0$	$M_y = 0$
		$N_z = 0$	$N_z = 0$
		$M_z = 0$	$M_z = 0$
	7	$N_x = 26666.6667$	$N_x = 26666.6667$
		$M_x = 0$	$M_x = 0$
		$N_y = 0$	$N_y = 0$
		$M_y = 0$	$M_y = 0$
		$N_z = 0$	$N_z = 0$
		$M_z = 0$	$M_z = 0$
	8	$N_x = 0$	$N_x = 0$
		$M_x = 0$	$M_x = 0$

		$N_y = 0$	$N_y = 0$
		$M_y = 0$	$M_y = 0$
		$N_z = 0$	$N_z = 0$
		$M_z = 0$	$M_z = 0$
	9	$N_x = 0$	$N_x = 0$
		$M_x = 0$	$M_x = 0$
		$N_y = 0$	$N_y = 0$
		$M_y = 0$	$M_y = 0$
		$N_z = 0$	$N_z = 0$
		$M_z = 0$	$M_z = 0$
	10	$N_x = 0$	$N_x = 0$
		$M_x = 0$	$M_x = 0$
		$N_y = 0$	$N_y = 0$
		$M_y = 0$	$M_y = 0$
		$N_z = 0$	$N_z = 0$
		$M_z = 0$	$M_z = 0$
	11	$N_x = 0$	$N_x = 0$
		$M_x = 0$	$M_x = 0$
		$N_y = 0$	$N_y = 0$
		$M_y = 0$	$M_y = 0$
		$N_z = 0$	$N_z = 0$
		$M_z = 0$	$M_z = 0$
	12	$N_x = 0$	$N_x = 0$
		$M_x = 0$	$M_x = 0$
		$N_y = 0$	$N_y = 0$
		$M_y = 0$	$M_y = 0$
		$N_z = 0$	$N_z = 0$
		$M_z = 0$	$M_z = 0$