COMPUTER ASSISTED LEARNING FOR STRUCTURAL ANALYSIS USING OBJECT ORIENTED PROGRAMMING

By
As'ad Said Abu Khalaf

Supervisor
Prof. Hassan Saffarini

This Thesis was Submitted in Partial Fulfillment of the Requirements for the Master's Degree of Science in Civil Engineering/Structures

Faculty of Graduate Studies
The University of Jordan

May, 2006

This thesis was successfully defended and approved on: 23th May 2006

Examination Committee	**Signature**
Dr. Hassan S. Saffarini, Chairman Prof. of Civil Engineering	…….
Dr. Raed M. Samra, Member Prof. of Civil Engineering	…….
Dr. Nazzal S. Armouti, Member Assoc. Prof. of Civil Engineering	…….
Dr. Hassoun Hadid, Member Prof. of Civil Engineering (Hashemite University)	…….

DEDICATION

TO MY MOTHER, FATHER, DEAREST LADY AND FRIENDS

WITH SINCERE LOVE

As'ad

ACKNOWLEDGEMENT

I would like to express my deep appreciation and sincere gratitude to my supervisor Prof. Hassan Saffarini for his endless and generous assistance and support in carrying out this study. I would also like to extend my gratitude to Eng. Maher Khoury who has supported me all through the period of my study.

Sincere thanks and appreciation are also extended to the Civil Engineering Department and members of the examining committee for their suggestions.

Finally, I would like to thank my mother, father and all members of my family for their deep love and endless support.

LIST OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

C	Damping Matrix
C*	Generalized Damping
CAL	Computer Analysis Language
DAF	Dynamic Analysis of Multistory Frame
DSM	Direct Stiffness Method
DRS	Design Response Spectra
E	Modulus of elasticity
$\underline{F}_{t+\Delta t}$	Effective Loads at t+ Δt
F*	Effective Load
f(t)	dynamic load variation
F(t)	External Load vector
I	Moment of inertia
IT	Information Technology
GUI	Graphic User Interface
K	Stiffness Matrix
K'	Effective Stiffness Matrix
K*	Generalized Stiffness
L	Length of member
m	Mass of structure
m*	Generalized mass
M_i	Moment at joint i.

M_j	Moment at joint j.
MDF	Multi degree of freedom
OOP	Object Oriented Programming
p	Axial force in member
P(t)*	Generalized effective load
RDS	Rayleigh Deflected Shape
S_a	Spectral acceleration
S_d	Spectral displacement
S_v	Spectral Velocity
S_{pa}	Spectral pseudo acceleration
S_{pv}	Spectral pseudo velocity
SDF	Single Degree of Freedom
SRSS	Square-Root-of-Sum-of-Squares
T	Fundamental Period of Vibration
U	Displacement vector
$\dot{U}$	Velocity vector
$\ddot{U}$	Acceleration vector
$U_{t+\Delta t}$	Displacement at t+ Δt
$\dot{U}_{t+\Delta t}$	Velocity at t+ Δt
$\ddot{U}_{t+\Delta t}$	Acceleration at t+ Δt
V_i	Force at joint i

V_j	Force at joint j
v_i	Displacement at joint i
v_j	Displacement at joint j
VB	Visual Basic
VRML	Virtual Reality Modeling Languages
X(t)	Relative Displacement
$\overset{*}{X}(t)$	Relative Velocity
$\overset{**}{X}(t)$	Relative Acceleration
α , δ	Parameter that can be determined to obtain integration accuracy and stability
Δt	Time Interval or Increment
λ	Eigenvalue
θ_i	Rotation at joint i
θ_j	Rotation at joint j
ξ_n	Damping ratio
Ø	Eigenvector
ψi	shape function
ω_n	Circular frequency

LIST OF APPENDICES

COMPUTER ASSISTED LEARNING FOR STRUCTURAL ANALYSIS USING OBJECT ORIENTED PROGRAMMING

By
As'ad Said Abu Khalaf

Supervisor
Prof. Hassan S. Saffarini

ABSTRACT

In this research, the computer program JU.CAL is developed to assist in the learning of basic structural analysis concepts and to bridge the gap between the teachings of traditional matrix methods of analysis and the use of completely automated structural analysis programs. The program JU.CAL was designed using Visual Basic .Net 2003 programming language to take advantage of Object Oriented Programming (OOP). This approach in programming is currently the methodology of choice for development of most state-of-the-art software. The powerful and flexible characteristic of OOP allows the programmer to produce more intelligible, verifiable, maintainable and expandable codes with a high degree of efficiency.

The graphic user interface of JU.CAL is divided into matrix definitions where any matrix of any size is input, revised and/or saved. Consequently, the previously saved matrices can be added, multiplied, transposed, inverted, etc. Such operations can be used as a stand alone application or as input for other parts of the program. Direct stiffness operations are utilized in the second part of the program to form element stiffness matrices and add them to arrive at the global stiffness matrix for any plane structure consisting of line elements. Such stiffness is then solved in conjunction with user-entered load vector and displacements are obtained. Similarly,

the member forces are obtained using special operations. The whole process is intended to be educational. Moreover, in the third part of this program, dynamic analysis includes the Eigen problem and step by step integration to solve equations of motion. The approach used in JU.CAL is subdivided into a logical sequence of separate operations.

Additionally JU.CAL contains two separate programs, dynamic analysis of multistory frame and design response spectra generation. These programs are all developed as classes that are incorporated in the environment of JU.CAL. Other classes can be added in later versions of the program by student wishing to take advantage of the environment of JU.CAL.

In conclusion, it is encouraged that this program be used in educational and academic environments to enhance the understanding of matrix methods for static and dynamic analysis of structures. This also allows teaching more complex problems than those that can be solved using hand calculations. It is further encouraged that other modules and classes be added to this program in order to further expand its application.

Finally, the results of JU.CAL are verified against well known commercial software and good comparison between results is found.

Introduction

1.1 Introduction

The proliferation and overwhelming use of personal computers and their associated internal and external networks at this Information Technology (IT) age we are living in has led to the development and use of computer software and applications in almost all activities of our daily lives.

Naturally, the profession of structural engineers has benefited enormously from such IT developments to the extent that it practically re-invented itself in the past fifteen years or so due to the availability on the market of powerful, yet relatively cheap and versatile structural analysis and design software packages.

Structural engineering software as utilized by the majority of structural engineers these days all over the world whether in small or large engineering practices, has managed to relieve the structural engineers of the drudgery of repetitive hand calculation thereby allowing them to concentrate upon exploring and arriving at more efficient and economic structural solutions for their project at much faster turnover time schedules.

The roots of the majority of structural analysis software packages lie deep in the history of the development of matrix methods for the solution of structural analysis problems. Such methods which are prohibitively difficult to apply by hand calculations for all practical purposes, lend themselves most elegantly and efficiently to the internal processing of computers. However, the use of computers and their applications in structural engineering cannot by any means whatsoever replace the education, understanding and experience of the structural engineer who is producing the design of a project. In this context, all structural engineering software must be perceived as a mere tool in the hand of structural engineers, a means to an end, but never the end itself.

As such, it is of paramount importance that structural engineers using any kind of structural analysis packages should have at least an understanding of the inner mathematical processes that go on inside the computer software before it starts displaying results. Most, if not all of the available software packages on the market never show intermediate results in a step by step manner. After the engineers input the structural geometry, material properties, loading and supports for a particular structure, the analysis process is completely automated and the analysis results are evaluated and shown as displacements and forces for the structure in question.

To most practicing structural engineers, a structural analysis package becomes like a black box where data is input, processed and then results are eventually output and displayed. It is precisely the lack of knowledge of how this data is mathematically understood and processed within the software that can lead to tremendous problems especially on complex structures. Such problems can range from misconfiguration of geometry and loading to misinterpretation of results and their questionable verification. All or even part of the aforementioned can lead to taking decisions by the structural engineers which can have adverse effects on their project at the practical level.

It follows from the above that it is vital for structural engineers to learn at least the fundamentals and basic assumptions, theories and interpretations of computers based structural analysis. Unfortunately, most civil/structural engineering educational curricula at institutions of higher learning, these days are not shifting their teaching of structural analysis courses towards computer methods in a comprehensive and efficient manner, thus creating a gap between the education that an engineer receives and real life practice. This is not to diminish the importance of classical methods in structural analysis as such methods still represent the basis for understanding fundamental structural behavior. However, an evident need presents itself these days in the form of a challenge to all who

are responsible for designing courses in structural analysis. This challenge is how best to introduce students even at the first degree level to computer based structural analysis while at the same time maintains the core curriculum of classical structural engineering education. One of the major difficulties of introducing computers based structural analysis techniques is the time consumed in teaching them which would be at the expense of other important topics in structural engineering.

The main computer based analysis technique utilized by the majority of software packages on the market is the Direct Stiffness Method (DSM). Although simple in concept, it does not lend itself easily to hand calculation solutions and thus practical implementation and practice exercises etc. However, the DSM can easily be implemented within a computer program. One possible solution to this problem is to develop tools for teaching computer based structural analysis employing the DSM using basic principles while assisted by purpose made software.

The computer program JU.CAL, which is presented in this study, is a significant extension of a Computer Analysis Language (CAL), which was developed in FORTRAN language by Edward Wilson in 1978 for teaching structural analysis. JU.CAL employs a Graphic User Interface (GUI) to simplify communication between the user and the software. Screen forms are used to input values, select the analysis methods and to view the results. The output results are summarized in tables with numerical values and charts, diagrams and animation with the ability of creating hard copies of these tables and diagrams as reports. The program was designed using Visual Basic .Net-2003 programming language to take advantage of Object Oriented Programming (OOP).

Object Oriented Programming gives the program the ability to be declared publicly as separate packages which can be used in other stand-alone programs.

1.2 Purpose

The overall objective of this study is to develop tools for teaching structural analysis using basic principles while assisted by purpose made software in order to avoid long manual calculations and thus to allow the students to concentrate on the core objectives of structural analysis learning and understanding. Object Oriented Programming is utilized to develop classes that reduce the time and effort of developing applications and increasing program reliability. The developed software shall be readily useable by other programmers seeking to add applications and Classes to it. This software is intended to be used by university students to develop their skills in structural analysis. It shall have an edge over the simplistic applications typically used in many textbooks to demonstrate structural methods and also provide a necessary alternative to ready-made black-box structural analysis software.

1.3 Literature Review

Object Oriented Programming is now the methodology of choice for the development of most state-of-the-art software. The powerful and flexible characteristic of OOP allows the programmer to produce more intelligible, maintainable and expandable codes with less effort. Alok Madan (2004) described the theoretical background, concept and the advantages of OOP for the development of engineering software. He found that the designer can manage the complexity more conveniently and effectively with OOP in comparison to other programming techniques. Marco L. Bittencourt and Raul A. Feigoo (2001) described the development of a two-dimensional interactive software environment for structural analysis and optimization based on OOP using the C++ languages. Adel Al-Assaf and Hassan S. Saffarini (2004) discussed the application of OOP to optimization of concrete slabs. Jianing Ju and M. U. Hosain (1996) addressed the design and application

of a group of graphical finite element object classes that were developed using OOP and were implemented in C++. They found that the object classes can be readily reused in other applications and the control program requires little or no modifications.

At the computer assisted structural analysis level, there have been many advances in information technology and educational institutions have worked to utilize these advances. Educational institutions have realized the importance of creating new methods for teaching engineering concepts and have turned to technology to aid in their development. When teaching complex engineering concepts and theories in standard lecture environments, students do not always easily grasp the information being presented. However, when these concepts and theories are presented in a virtual environment within a process of interaction with instructive programs, Haque (2001) found that a student's understanding of the material was improved. Hence, demonstrating the effectiveness and the need for interactive programs; he developed a web-based interactive virtual environment for the illustration of behavior and the design for flexural and shear in reinforced concrete beams using Java and Virtual Reality Modeling Languanges (VRML). Jiang (2002) created three virtual laboratory modules which educate students on reinforced concrete structures. These modules are based on applets, which perform the calculations involved in the analysis of reinforced concrete sections. One module allows the user to explore the flexural design of rectangular singly reinforced concrete beams. Another module shows the axial force, moment and curvature relationships for rectangular beam and column sections. The last module explores the relationship of uniaxial stress-strain for confined and unconfined reinforced concrete. Hibbeler (2002) included in his textbook software for teaching structural analysis STRAN v.4.0, which is applicable to plane trusses, space trusses, beams and plane frame structures. Unfortunately, this software lacks any graphical output. Edward L. Wilson (1978) has lead in developing a computer analysis

language for teaching structural analysis to avoid long hand calculations and matrix manipulation in order to let the student concentrate on the core objectives of structural analysis learning. Based on FORTRAN language, this software also lacked a graphical user interface. The program was implemented during the eighties and nineties on various operation systems at the University of Jordan from VMS to UNIX and DOS and was later implemented partially on Windows using VB.

1.4 Research Organization

This study consists of six chapters. The first chapter is an introduction of this study. In chapter two, an overview of object oriented programming methodology is presented. Chapter three describes the JU.CAL operations which consist of three parts. The first part is the matrix operations which are used to add to the working space new matrices by defining the matrix name and the matrix entries values. The new matrices are then used in calculations like matrix addition, multiplication, transpose, inverse, determinant, transformation and other operations. These operations may be used alone or in conjunction within the second and/or third part of the program. The second part utilizes direct stiffness operations to analyze beams and frames. The third part describes dynamic operations. Chapter four describes the dynamic analysis of multistory frames and illustrates its incorporation - JU.CAL. In chapter five, response spectrum design is described and presented with reference to implementation in the current software. A summary of the results, conclusions and recommendations is provided in chapter six.

Overview of Object Oriented Programming

2.1. Introduction

In this chapter, an overview of object oriented programming is presented. A brief description of the traditionally used procedural programming technique is given. Then the concept of object oriented programming, encapsulating, abstraction, inheritance and polymorphism is presented. Advantages and disadvantages of OOP is presented. Finally, a description of the object oriented Visual Basic .Net language is presented.

2.2 Procedural Programming

Procedural languages organize the program in a linear fashion, they run from top to bottom. In other words, the program is a series of steps that run one after another. The development of higher level languages such as FORTRAN, Pascal and C ushered the era of structured programming that came to be known as procedural programming. In procedural programming the data and functions are separate entities within the program. Data variables have to be declared as global in order for functions in the program to have access to these variables. The use of global variables increases the possibility of functions accidentally changing the data. The concept of repetitive functions or algorithms within procedural language programming is handled by subroutines which introduces jumps in the consecutive steps of the program these by adding more difficulty in controlling the program flow. The data and functions of procedural programs do not model real life objects and program development is significantly more difficult. Furthermore, the maintaining of data and functions of a large complex program becomes a difficult task. For smaller programs these disadvantages are not as noticeable. Procedural programming is still widely used.

2.3 Object Oriented Programming

Object oriented programming is a computer programming paradigm in which a software system is modeled as a set of objects that interact with each other. This interaction takes the form of messages passing back and forth between the objects. In response to a message, an object can perform an action, or a procedure. These objects contain both data members and functions. There are two main concepts in object oriented programming, objects and classes.

2.3.1 Objects and Classes

An object is a collection of data and code in the form of subroutines and functions to manipulate and access the data. An object can represent a person, place, thing, concept, or event applicable to the system. Objects are created by using classes. In other words, an object is an instance of a class. A class consists of data members and functions. A class defines the data and the functions used by each object of the class. In other words, the class serves as a template for creating the object (Simon, 2002). An example of a class from a structural engineering program that analyzes a frame would be the Direct Stiffness Operation (DSM) class. The DSM class might consist of such data members as the start and end joints of the member, startJoint and endJoint; the cross sectional area of the member, A; the modulus of elasticity of the member, E; the directional cosines of the member, CosineX and CosineY; and the length of the member, L. An example of a function of the class DSM is FRAME() function which calculates the length of the Frame member and evaluate the stiffness matrix [K1] for the two-dimensional frame member. Using the class DSM, objects representing frame members could be created. Each object would have its own data members; startJoint, endJoint, A, E, cosineX, cosineY, and L. Data members are usually declared as being private. Private data members can only be used by functions within the class in which they are defined. Data members, however, can

also be declared as protected or public. When data members are declared as protected, functions in a derived class can also have access to the data members. Data members declared as public in a class are accessible to any class.

2.3.2 Concepts of Object Oriented Programming

2.3.2.1 Encapsulation

One of the most important principles of objects is that they enclose functionality and data while providing an interface with which to interact with other objects. An object's users don't care how an object works or what it does; they just want it to function well and to have a simple interface. This "black box" characteristic of objects is known as encapsulation. An object encapsulates data and functionality, keeping the implementation details out of users' hands. The object provides a simple interface as a combination of methods, and in some cases member data, for users to control the object's actions at high level without dealing with the implementation details. In other words, encapsulation viewed as a methodology of hiding details of the implementation of an object from the user. Encapsulation can speed development and limit the number of errors within an application (Simon, 2002).

2.3.2.2 Abstraction

Abstraction is a process that involves identifying the crucial behavior of an object and eliminating irrelevant and tedious details. In other words, Abstraction determines what a class needs to know about an object (Hamilton, 2002). For example consider a customer of a certain bank. The bank requires certain record keeping information including the customer's name, social security number, address and phone number. However, each

customer has additional characteristics not required by the bank including the customer's weight, hair color and hobbies.

2.3.2.3 Inheritance

Inheritance is the OOP feature that allows new object classes to be created from existing ones. If a class inherits another class, it inherits all the features, behavior, code and characteristics of the class from which it inherits. The new class is called a child or derived class, while the original class is often referred to as the parent or base class. Derived class can be created that uses the same features of the base class and extends the base class by adding new features (Simon, 2002). For example, a car is a vehicle. A boat is a vehicle. A submarine is a vehicle. In OOP, the Vehicle base class would provide the common behaviors of all types of vehicles and perhaps delineate behaviors all vehicles must support. The particular subclasses (derived classes) of vehicles would implement behaviors specific to that type of vehicle.

2.3.2.4 Polymorphism

Polymorphism is the capability of a group of objects to have the same interface with different underlying implementations (Simon, 2002). The benefit comes from being able to interchange the object types without any change in the user code. Each object implements the polymorphic interface appropriately, and the user code stays the same. For example, a group of shape objects all implement a Draw () method. The Draw () method for a circle shape object draws a circle, but for a rectangle object it draws a rectangle. Each shape object is interchangeable, and the user of the objects simply uses the Draw () method, letting each object deal with what is actually drawn. In other words, polymorphic objects have the same interface with different behavior. The advantage of polymorphism is

that the sender of a message doesn't need to know which class the receiver is a member of. It can be any arbitrary class. The sending object only needs to be aware that receiving object can perform a particular behavior.

2.3.3 Advantages and Disadvantages of Object Oriented Programming

Having objects that relate to real world objects makes programming complex problems more manageable. The characteristic of object oriented programming provides several advantages. Abstraction reduces the chance of objects accidentally altering data. Inheritance allows for ease in development of programs by using existing classes to develop new classes. The inheritance and polymorphism characteristics of object oriented programming allow for code reusability. Existing classes can be used as they are or can be modified to solve different problems. Libraries of classes are included in many of the object oriented programming languages. Reusing tested existing classes saves time, effort and increases program reliability. A disadvantage of object oriented programming is the requirement for more memory and slower execution time. This disadvantage is due to message passing and dynamic linking. However, with the development of faster computers with increased memory at relatively lower costs, this disadvantage is becoming less critical. Instead, more emphasis is being placed on the ease of development i.e. lower labor costs of the programmers, than the cost of the hardware.

2.4 Visual Basic.NET Programming Language

The Visual Basic.NET programming language is an object oriented language developed by Microsoft (Deitel, 2002). It is modeled after the Visual Basic 6.0 programming language. Visual Basic.NET has all of the advantages of other object oriented programming language such as reusability of code, extensibility, encapsulation,

abstraction, polymorphism and inheritance. Visual Basic 6.0 did not incorporate basic OOP features such as inheritance and method overloading. Without these features, developers were severely limited in their ability to construct complex distribution software systems. Microsoft has recognized these shortcomings and has changed Visual Basic 6.0 into a true OOP language with the release of Visual Basic .NET 2003.

JU.CAL Operations

3.1. Introduction

JU.CAL is an interpretive language which is designed to manipulate matrices and to perform several standard structural analysis operations. JU.CAL operations consist of three parts. The first part is the matrix operations which are used to add to the working space new matrices by defining the matrix name and the matrix entries (row or column) values. The new matrices are then used in calculations like matrix addition, multiplication, transpose, inverse, determinant, transformation and other operations. These operations may be used alone or in conjunction with the second and/or third part of the program. The second part utilizes direct stiffness operations to analyze beams and frames. The third part describes dynamic operations. For each part of above described operations the analysis process, the program interface and classes are presented. Finally, verification of results is presented for each part showing the comparison between known reference solutions from other sources and those from JU.CAL operations for the same input data.

3.2 Matrix Operations

3.2.1 Analysis Process

JU.CAL has most of the standard matrix operations plus some special array operations which are useful in structural analysis. Kassimali (1999) describes some of the common types of matrices and matrix operations as follows:

- Column Matrix (Vector): All elements of a matrix are arranged in a single column (n=1).
- Row Matrix: A matrix with all of its elements arranged in a single row (m=1).
- Square Matrix: If a matrix has the same number of rows and columns (m=n).

- Symmetric Matrix: when the elements of a square matrix are symmetric about its main diagonal ($A_{ij}=A_{ji}$).
- Lower Triangular Matrix: If all the elements of a square matrix above its main diagonal are zero ($A_{ij} = 0$ for $j > i$).
- Upper Triangular Matrix: When all the elements of a square matrix below its main diagonal are zero ($A_{ij} = 0$ for $j < i$).
- Diagonal Matrix: A square matrix with all of its off-diagonal elements equal to zero ($A_{ij} = 0$ for $i \neq j$).
- Unit or Identity Matrix: If all the diagonal elements of a diagonal matrix are equal to one ($I_{ij} = 1$ and $I_{ij} = 0$ for $i \neq j$).
- Null Matrix: If all the elements of a matrix are zero ($A_{ij} = 0$).
- Addition and Subtraction: Matrices can be added or subtracted only if they are in the same order (same number of rows and columns in each).
- Multiplication by a scalar: The product of a scalar C and a matrix [A] is obtained by multiplying each element of the matrix [A] by the scalar C.
- Multiplication of Matrices: Two matrices can be multiplied only if the number of columns of the first matrix equals the number of rows of the second matrix.
- Transpose of a Matrix: The transpose of a matrix is obtained by interchanging its corresponding rows and columns.
- Inverse of a square Matrix: The inverse of a square matrix [A] is defined as matrix $[A]^{-1}$ with elements of such magnitudes that the product of the original matrix [A] and its inverse $[A]^{-1}$ equals a unit matrix [I]. $[A]\,[A]^{-1} = [A]^{-1}\,[A] = [I]$.

Table 3.1 presents the matrix operations in JU.CAL.

Table 3.1 Matrix Operations in JU.CAL.

Operations	Description
ADD	This operation generates a new matrix, which is the addition of two entered.
DUP	This operation will form a new matrix which is identical to the entered matrix.
DUPDG	This operation forms a new row matrix from the diagonal terms of the entered matrix.
DUPSM	This operation forms a new sub matrix from the main matrix according to the columns and rows ranges.
INVERSE	This operation generates a new matrix which is the inverse of the entered matrix.
MAX	This operation forms a column matrix in which each row contains the maximum absolute value of the corresponding row in the entered matrix.
MULTI	This operation generates a new matrix which is the multiplication of the two entered matrices.
NORM	This operation forms a row matrix in which each entry contains the sum of the absolute values or the square roots of sum of squares (SRSS), for the corresponding columns of the entered matrix.
LOG	This operation replaces each element in the entered matrix with the logarithm value of each term within the matrix.
PRINT	This operation is used to print any matrix as a report.
SCALE	This operation replaces each element in the entered matrix with the scale of each term in accordance with a scale value input by the user

Operations	Description
SOLVE	This operation solves the equation [F] = [K].[u], where: $[u]_{mx1}$: The unknown matrix and it is a column matrix. $[F]_{mx1}$: A column matrix contains all external loads at the joints. $[K]_{mxn}$: The global stiffness matrix of the structure.
STODG	This operation stores a row or column matrix at the diagonal location of the main matrix in a new matrix.
STOSM	This operation stores the sub matrix in the main matrix and saves the output in a new matrix.
SUBTRACTION	This operation generates a new matrix which is the subtraction of two entered matrices of the same order.
TRANSPOSE	This operation generates a new matrix which is the transpose of the entered matrices.
INITIATE	This operation can be used to form null or unit matrices.
DETERMINANT	This operation evaluates the determinant of a matrix.

3.2.2 Matrix Operations Interface

JU.CAL is intuitive and easy to use. The main frame of the matrix operations interface contains a buttons bar, combo box, text box, flex grid, key window, animation, a panel of buttons and a menu bar as shown in Figure 3.1. As seen in Figure 3.2, the menu bar contains a File selection, Operations selection, Analysis selection, Information, About and Help selection. The File Menu selection has five menu items, New, Open, Save, Close and Exit as shown in Figure 3.3. Another way to select these items is the Button bar as shown in Figure 3.4.

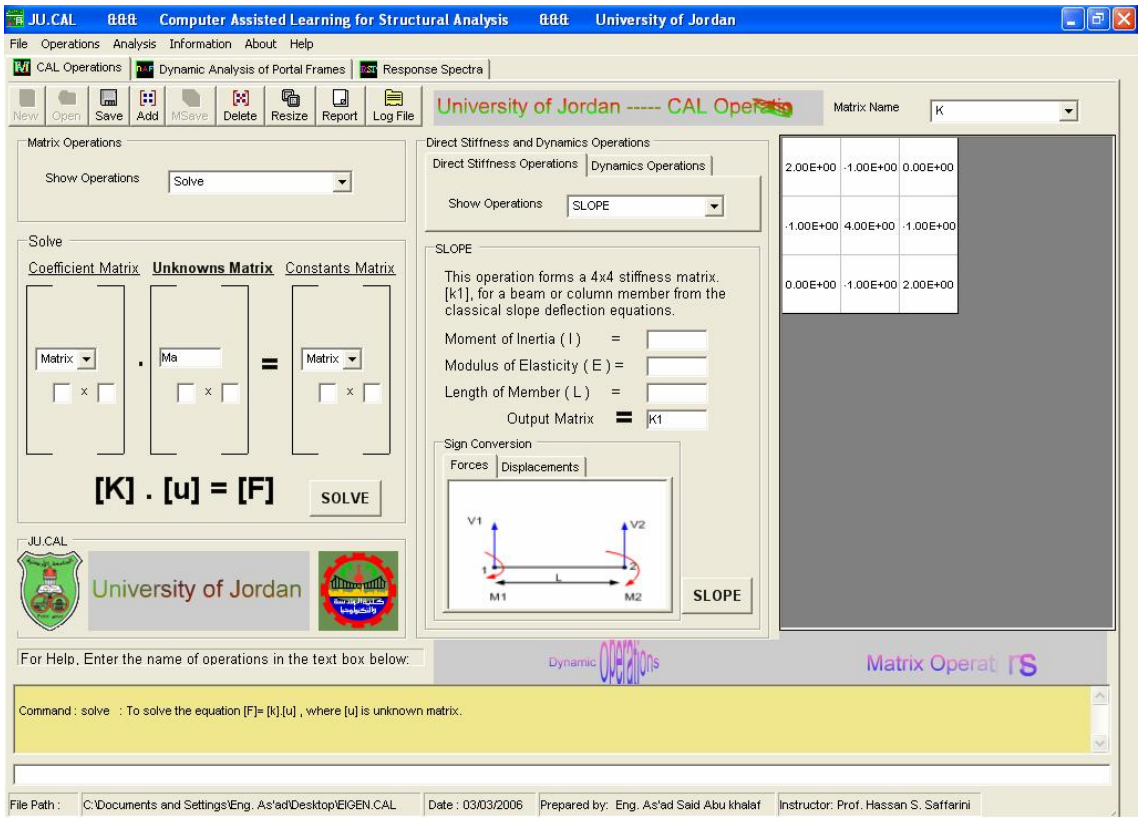


Figure 3.1 Graphical User Interface of JU.CAL.

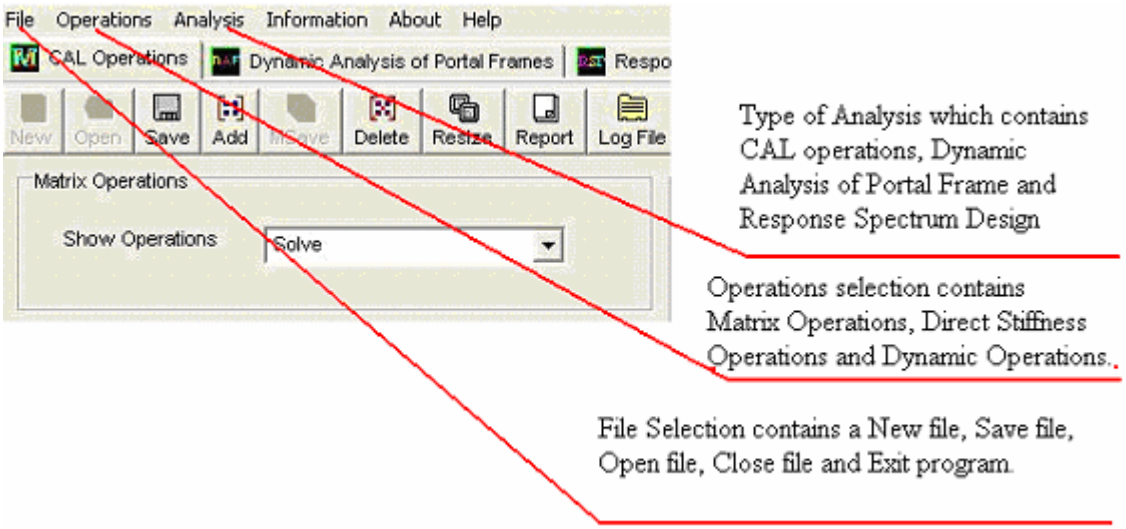


Figure 3.2 Menu bar of JU.CAL.

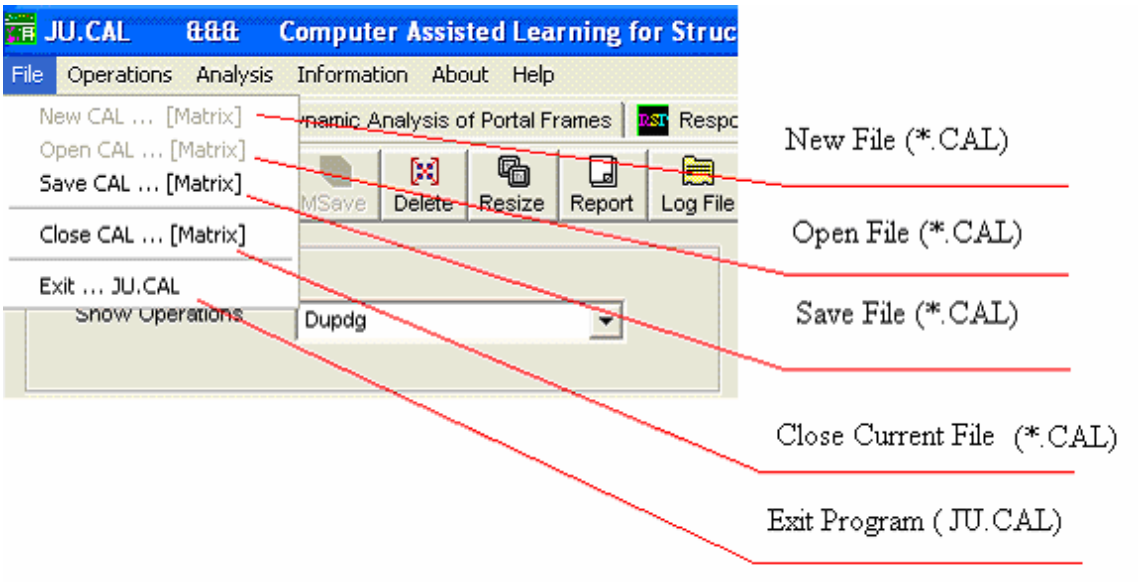


Figure 3.3 File Menu of JU.CAL.

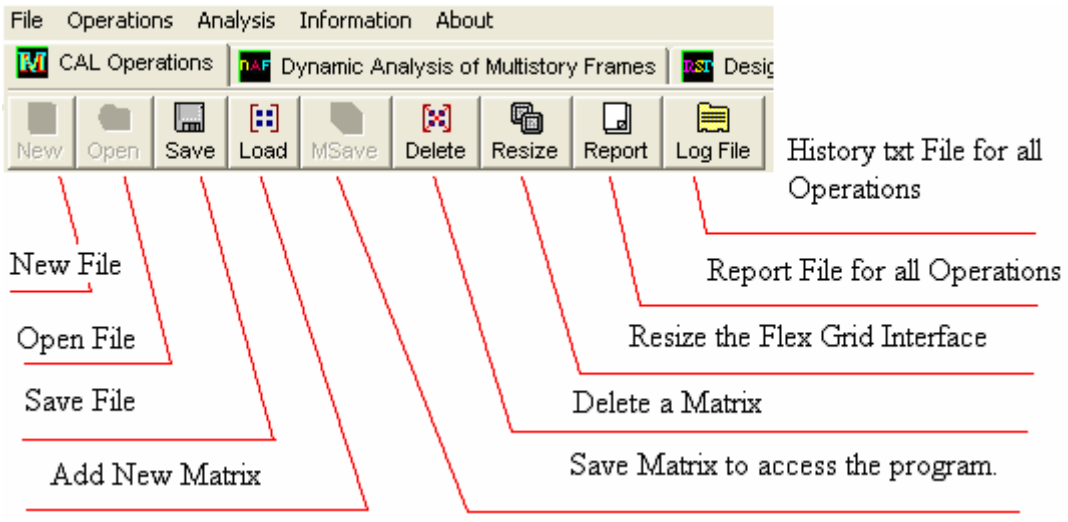


Figure 3.4 Button Bar of CAL Operations.

As seen in Figure 3.4, the Button bar has nine button items, New, Open, Save, Report and Log File for CAL operations. Other items are used for adding a new matrix, deleting and saving it, like Add, MSave and Delete as shown in the Figure 3.4. When we add a new Matrix, all buttons on the bar are disabled, except the MSave button in order to save the

matrix and to use this matrix in all subsequent operations. Resize button is used to control the size of the flex grid for the interface frame as shown in Figure 3.5.

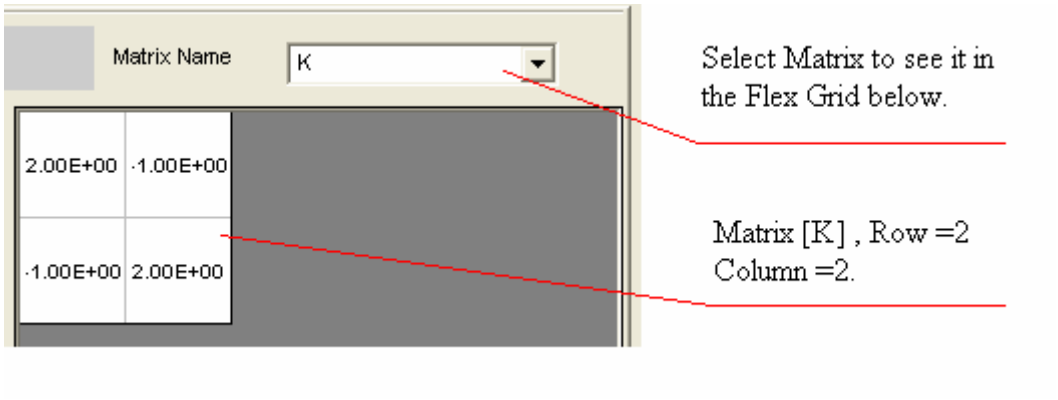


Figure 3.5 Flex Grid Interface.

The Operations Menu selection has three menu items, Matrix, Direct Stiffness and Dynamics as shown in Figure 3.6. With the selection of the Matrix item, we can use the type of operations to be selected in the sub-menu to the right. Another way to show and select matrix operations is the combo box as shown in Figure 3.7. The key window and text box as shown in Figure 3.8 are also used for the same purpose.

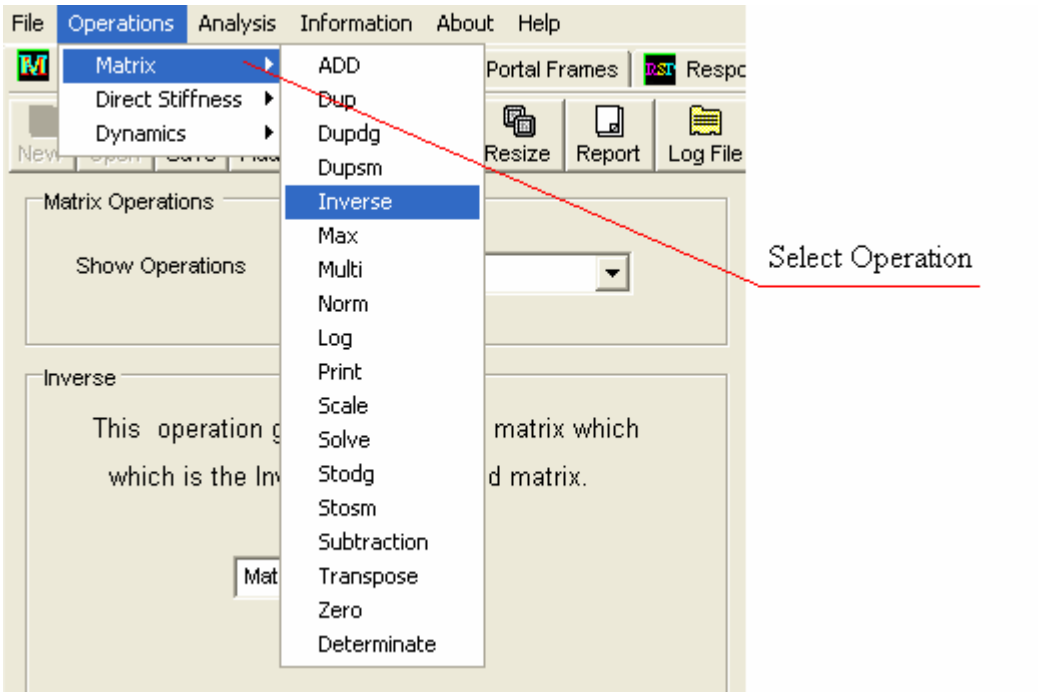


Figure 3.6 Operations Menu of JU.CAL.

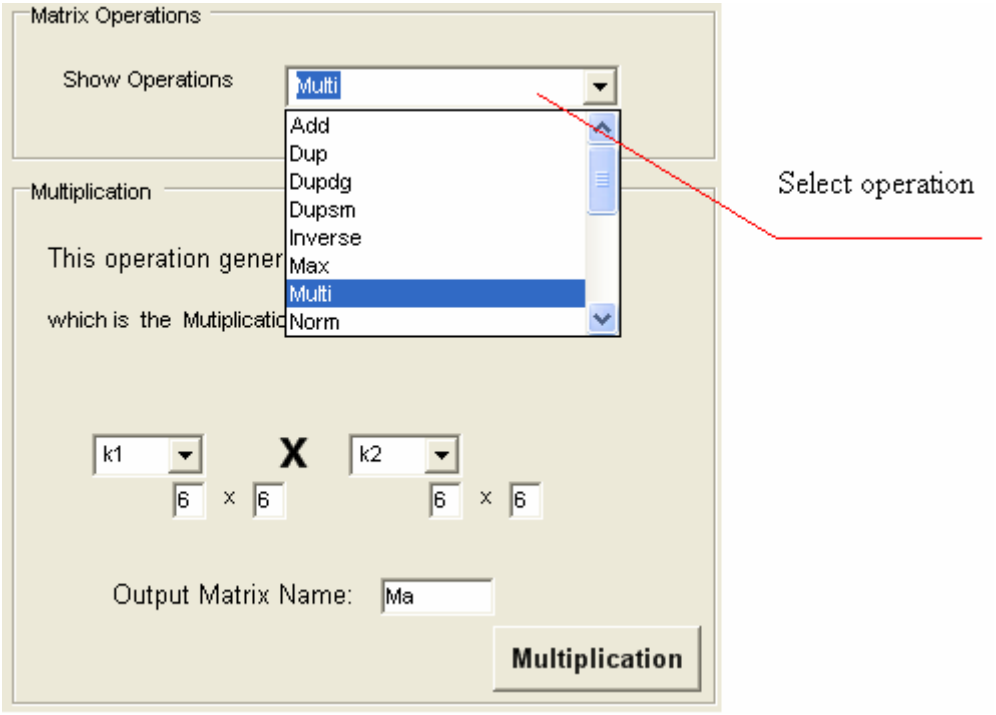


Figure 3.7 Matrix Operations.

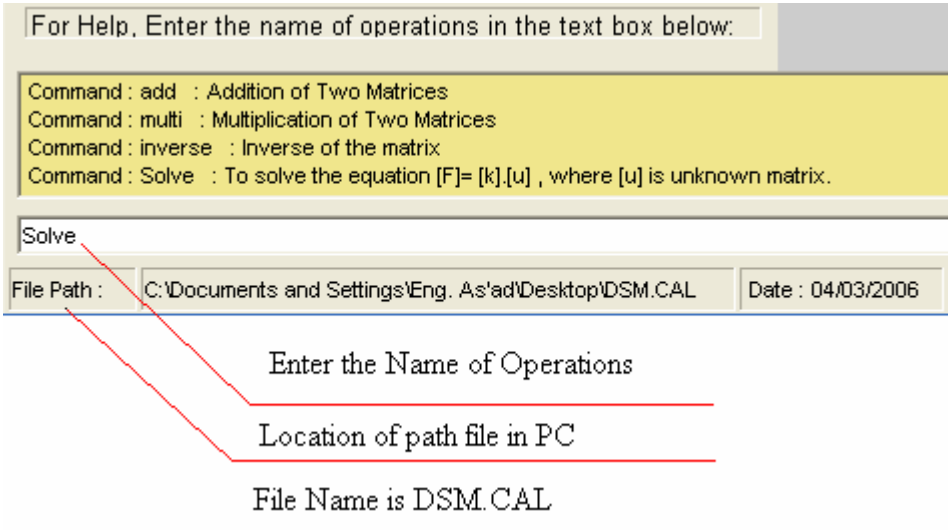


Figure 3.8 Key Windows.

The following is a list of matrix operations which are used for control and general matrix or array manipulation.

- ADD: This operation generates a new matrix, which is the Addition of two previously entered matrices as shown in Figure 3.9.

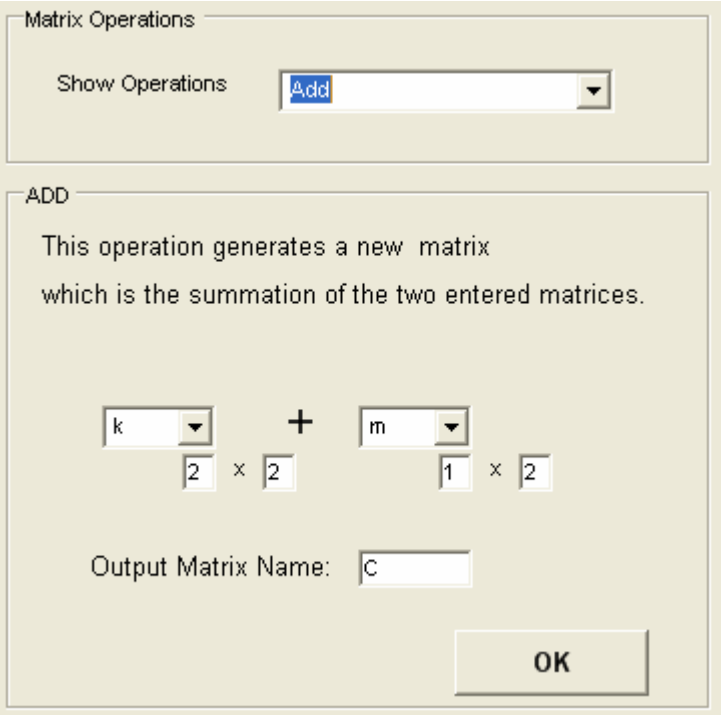


Figure 3.9 ADD Operation.

- DUP: This operation will form a new matrix which is identical to a previously entered matrix as shown in Figure 3.10.

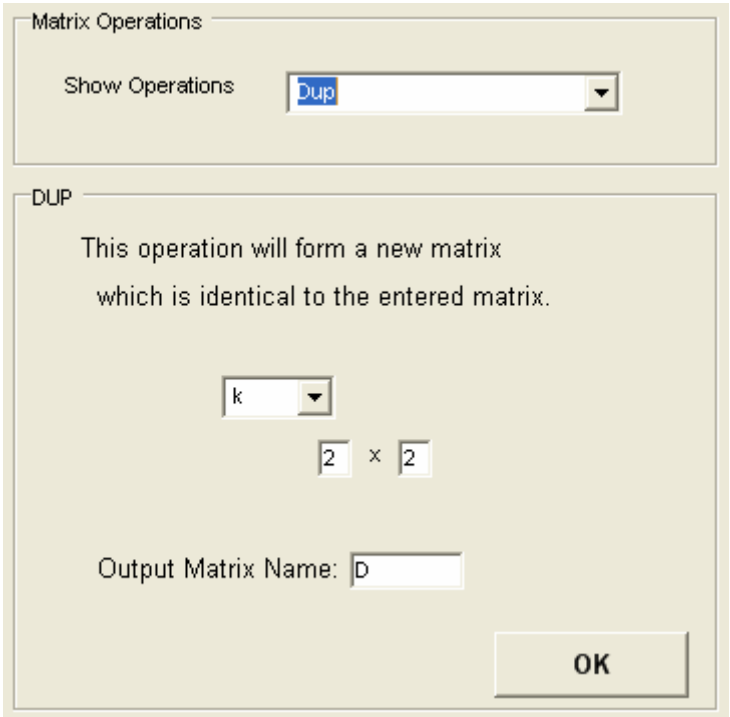


Figure 3.10 DUP Operation

- DUPDG: This operation forms a new row matrix from the diagonal terms of a previously entered matrix as shown in Figure 3.11.

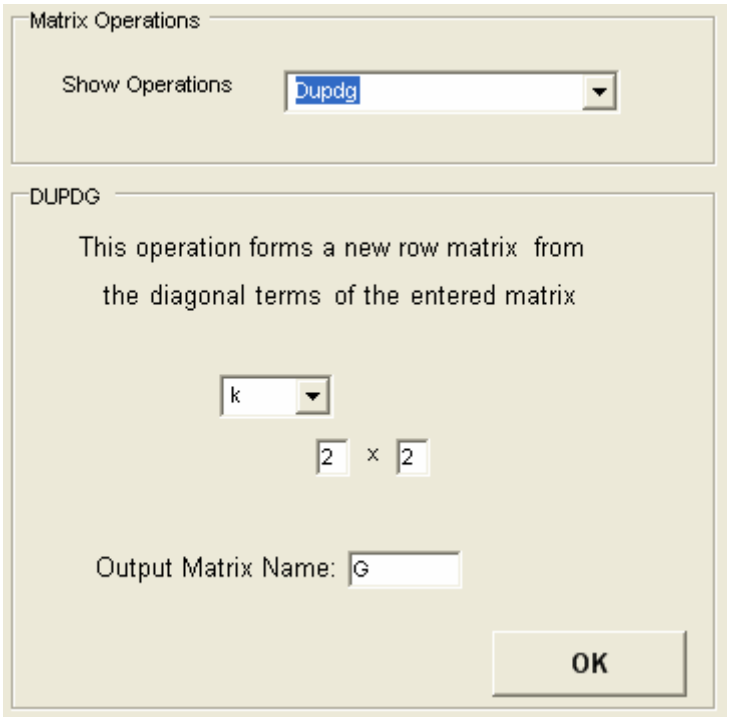


Figure 3.11 DUPDG Operation

- DUPSM: This operation forms a new sub matrix from a main matrix according to the columns and rows ranges as shown in Figure 3.12.
- INVERSE: This operation generates a new matrix which is the inverse of a previously entered matrix as shown in Figure 3.13.
- MAX: This operation forms a column matrix in which each row contains the maximum absolute value of the corresponding row in an entered matrix as shown in Figure 3.14.
- MULTI: This operation generates a new matrix which is the multiplication of two previously entered matrices as shown in Figure 3.15.

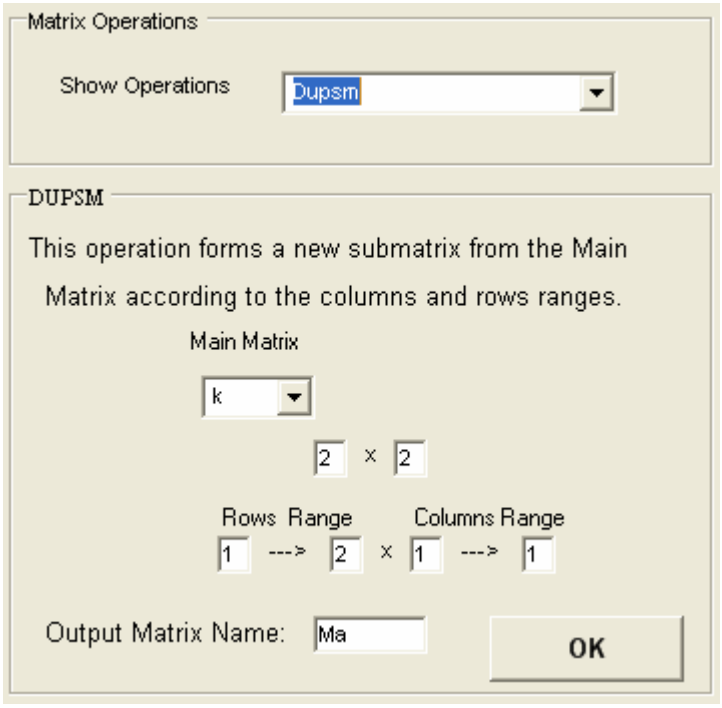


Figure 3.12 DUPSM Operation

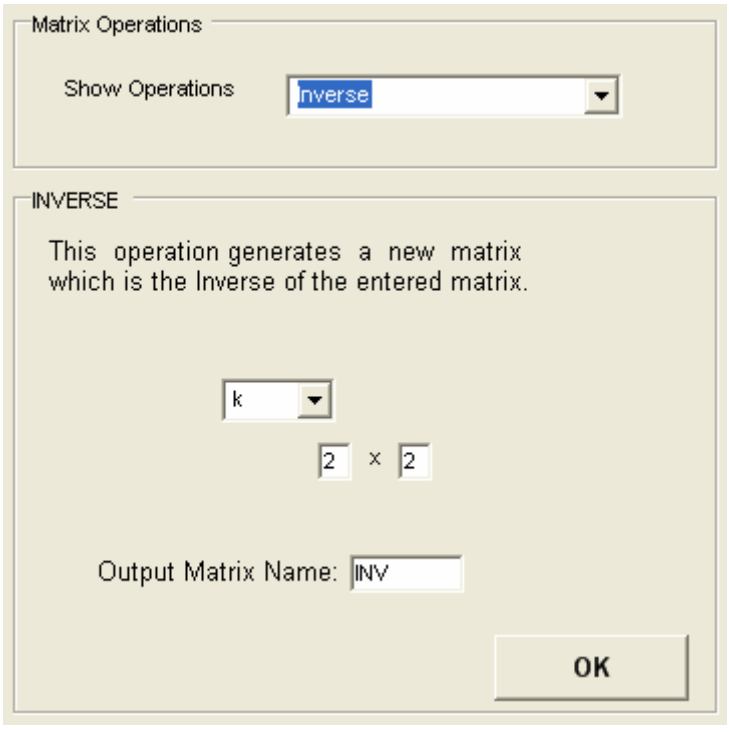


Figure 3.13 INVERSE Operation

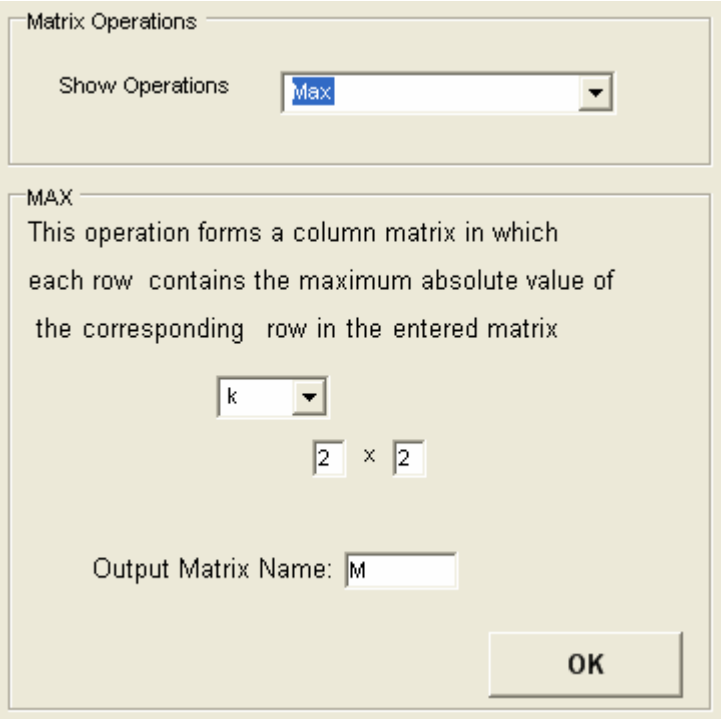


Figure 3.14 MAX Operation

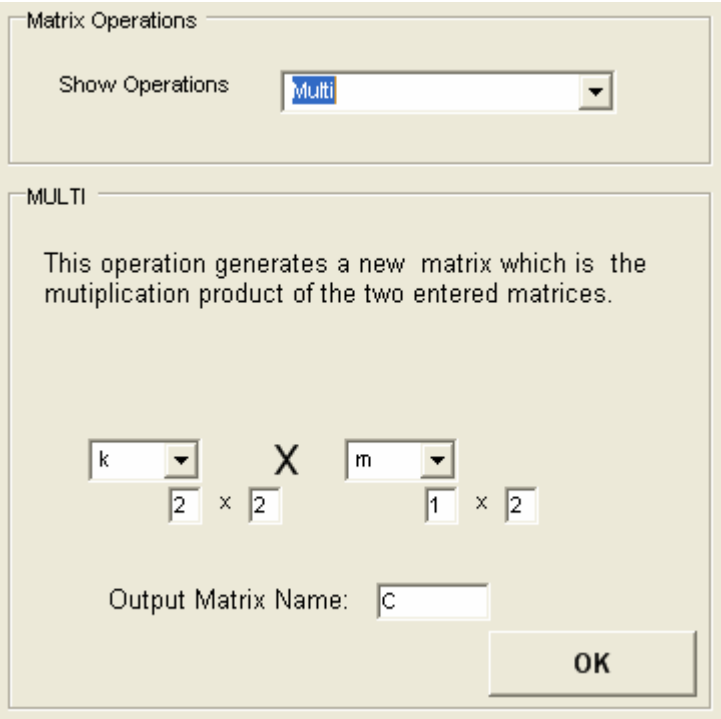


Figure 3.15 MULTI Operation

- NORM: This operation forms a row matrix in which the column contains the sum of the absolute values or the square roots of sum of squares (SRSS), for the corresponding columns of the entered matrix as shown in Figure 3.16.

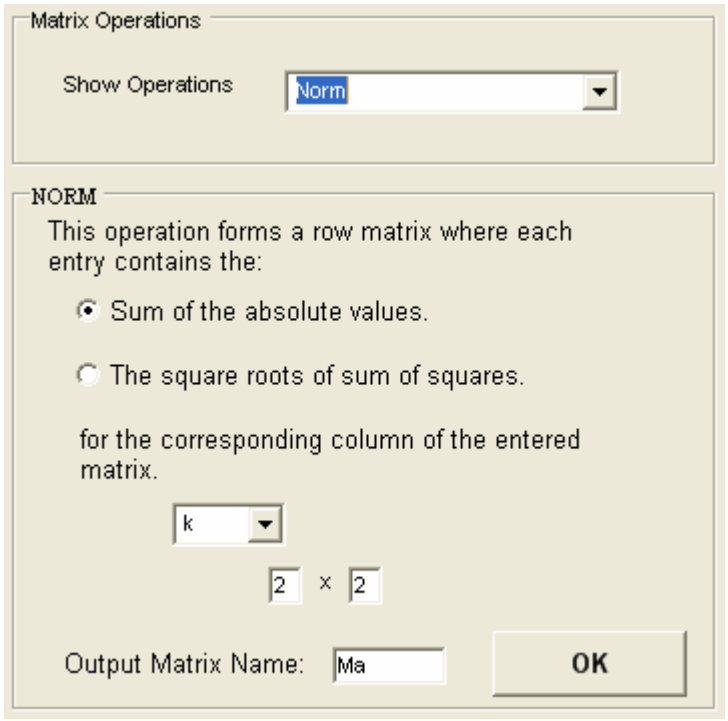


Figure 3.16 NORM Operation

- LOG: This operation replaces each element in the entered matrix with the logarithm of each term as shown in Figure 3.17.
- PRINT: This operation is used to print any matrix as a report as shown in Figure 3.18.
- SCALE: This operation replaces each element in the entered matrix with the scaled value of each term as shown in Figure 3.19.
- SOLVE: This operation solves the equation [F] = [K].[u] as shown in Figure 3.20.

Where: [u]: Displacement and/or joint rotation matrix which is usually unknown.

[F]: Load Matrix.

[K]: The Global Stiffness Matrix of the structure to be solved for displacements and rotations.

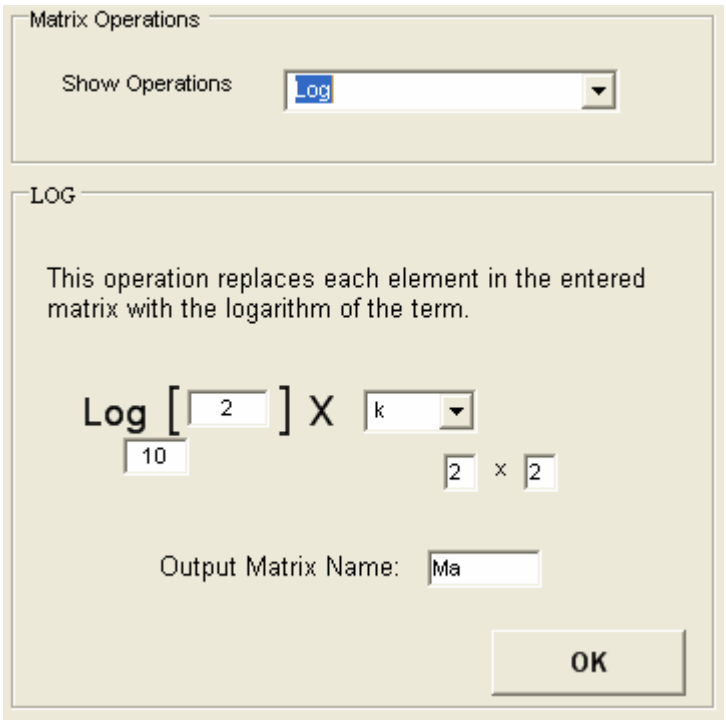


Figure 3.17 LOG Operation

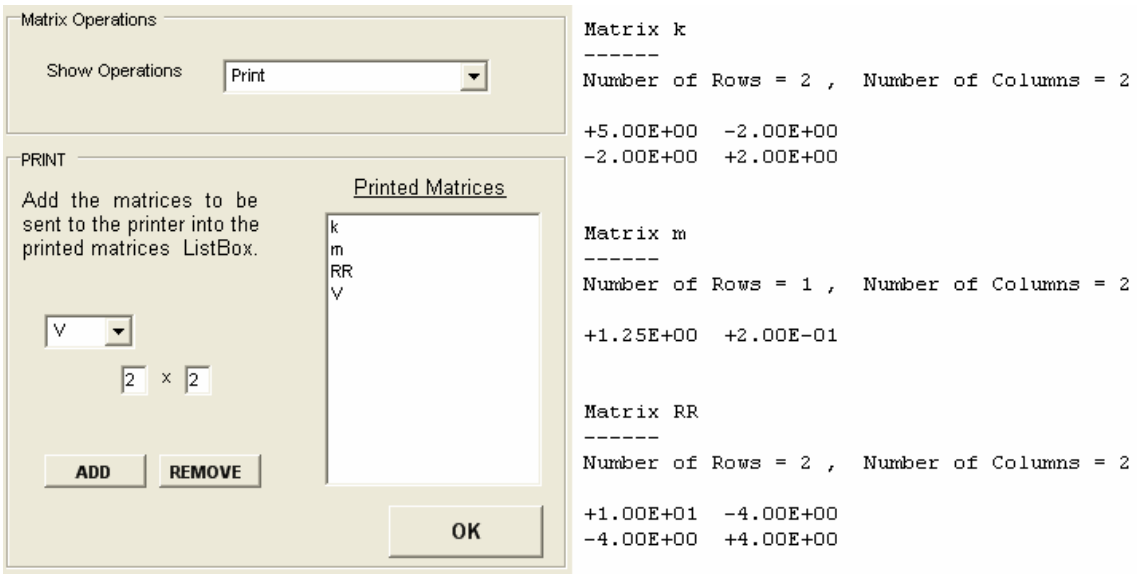


Figure 3.18 PRINT Operation

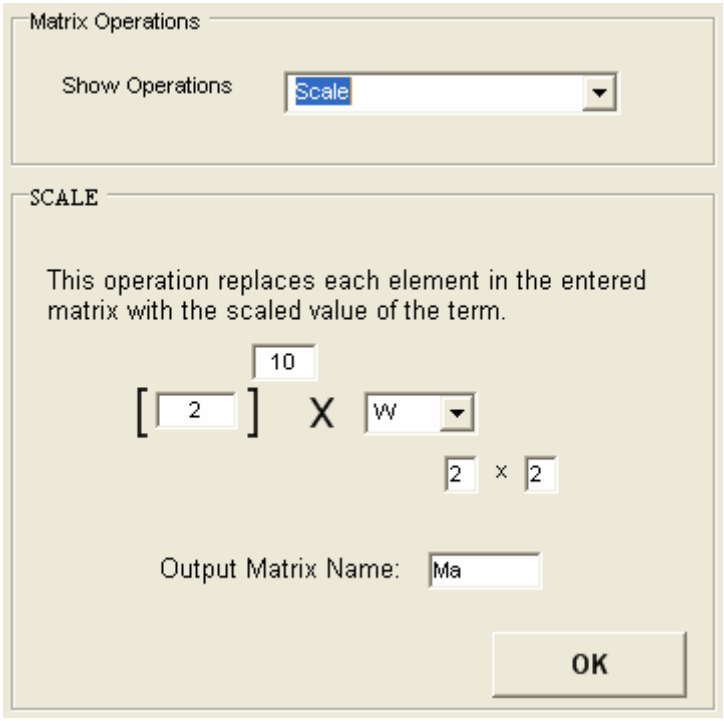


Figure 3.19 SCALE Operation

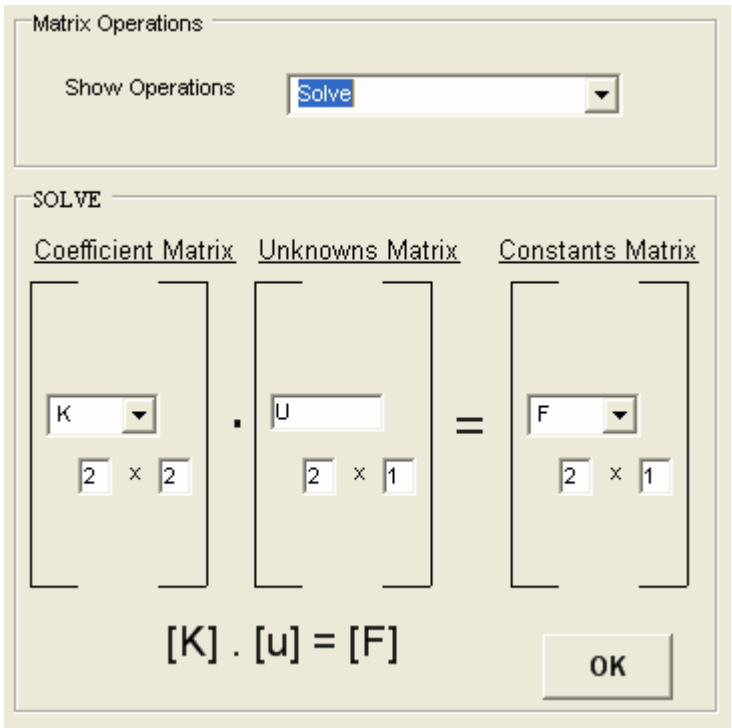


Figure 3.20 SOLVE Operation

- STODG: This operation stores a row or column matrix at the diagonal location of the main matrix in a new matrix as shown in Figure 3.21

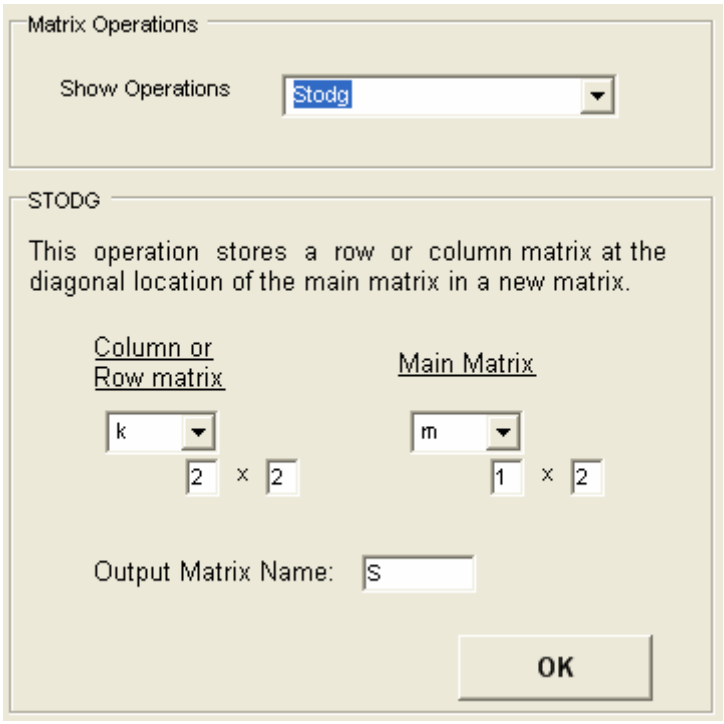


Figure 3.21 STODG Operation

- STOSM: This operation stores the sub matrix in the main matrix and saves the output in a new matrix as shown in Figure 3.22

- SUBTRACTION: This operation generates a new matrix which is the subtraction of two previously entered matrices as shown in Figure 3.23.

- TRANSPOSE: This operation generates a new matrix which is the transpose of a previously entered matrix as shown in Figure 3.24.

- INITIATE: This operation can be used to form a new null or unit matrix as shown in Figure 3.25.

- DETERMINANT: This operation evaluates the determinant of a matrix as shown in Figure 3.26.

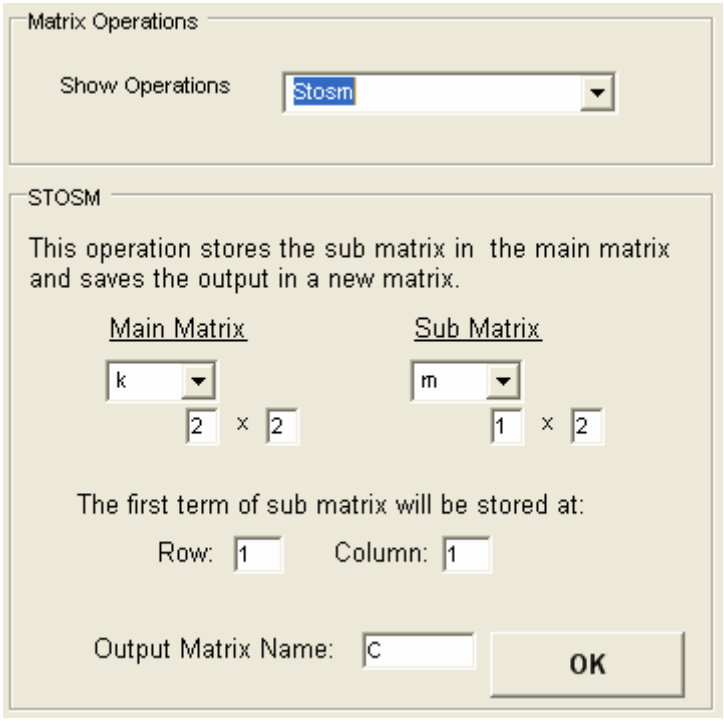


Figure 3.22 STOSM Operation

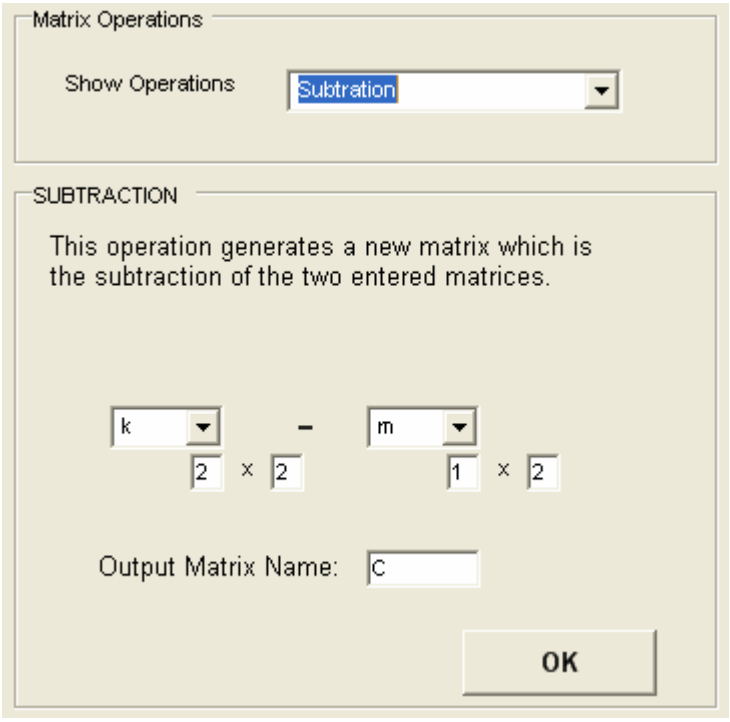


Figure 3.23 SUBTRACTION Operation.

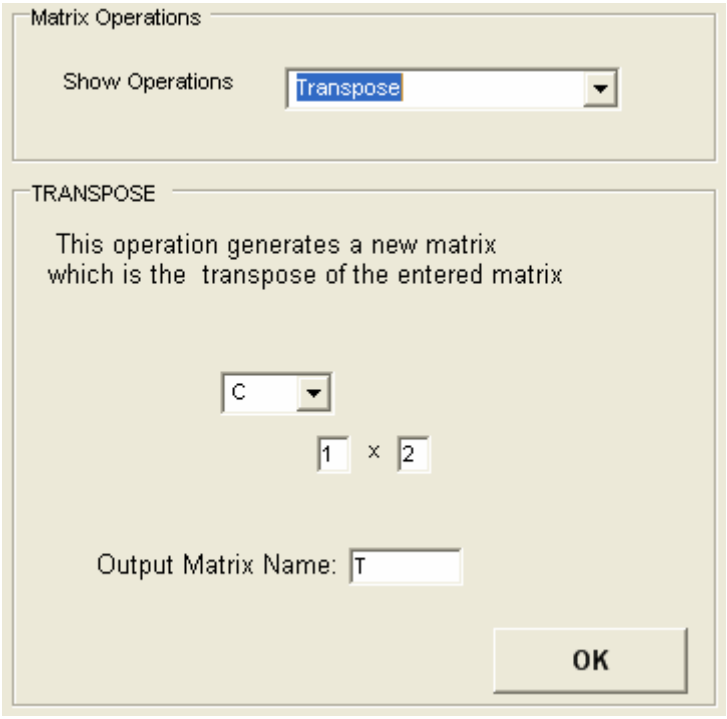


Figure 3.24 TRANSPOSE Operation

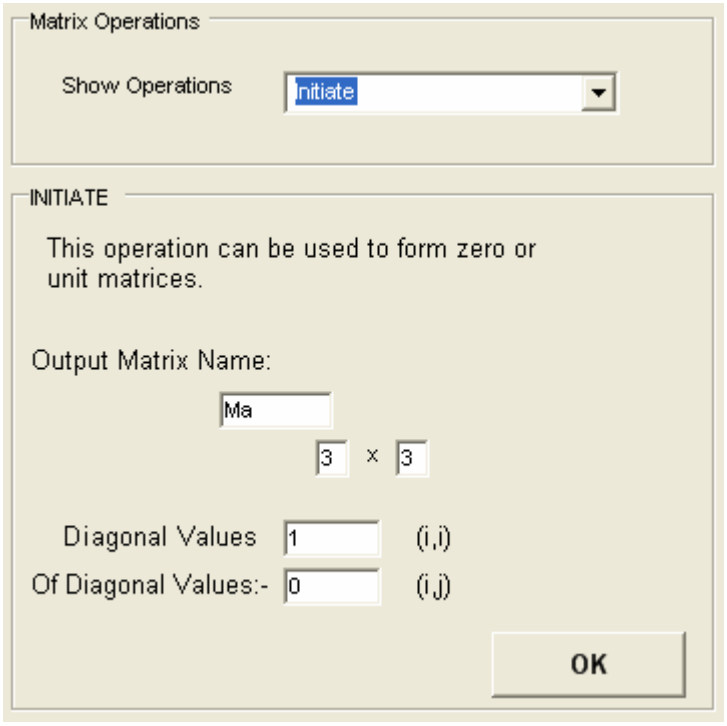


Figure 3.25 INITIATE Operation

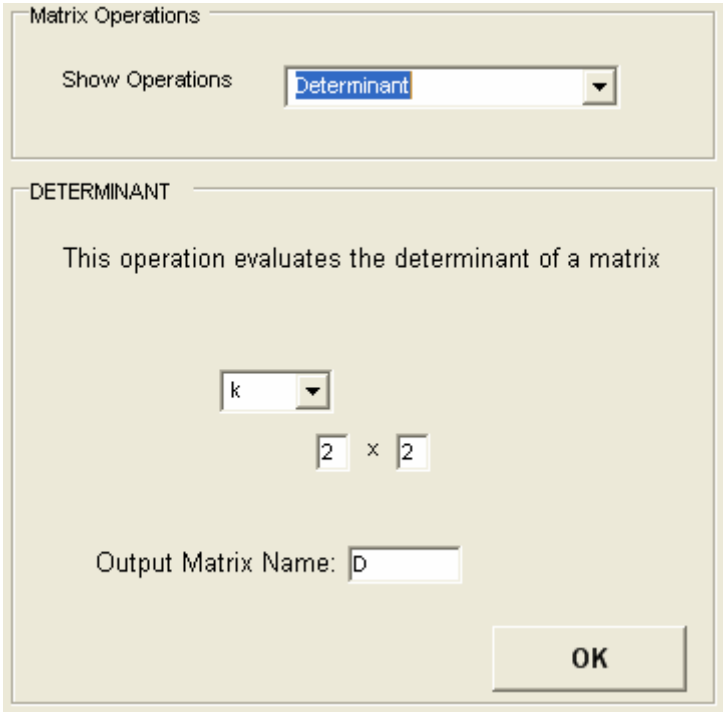


Figure 3.26 DETERMINANT Operation

3.2.3 Matrix Operations Classes

The structural classes that were developed for this program are shown in Table 3.2. They include the Matrix, Direct Stiffness and Dynamic classes. A list of the methods used in the Matrix Operations Classes is shown in Figure 3.27. As mentioned in the introduction, with object oriented programming, once a class has been developed it can be used when necessary in other programs. The listings of these classes are given in appendix A.

Table 3.2 JU.CAL Classes.

Class	Description
Matrix Operations	All the matrix manipulation is done here. This class contains functions of matrix analysis method.
Direct Stiffness	Represents the stiffness elements of a structure for beams, columns and frames. This class contains data members for adding stiffness matrix of elements to the global stiffness matrix of the structure, then the forces in each member are calculated.
Dynamics	This class contains operations to solve the eigenvalue problem, The dynamic analysis uses step by step method and plots functions in order to solve the classical equation of motion in the form $\underline{M}\ddot{\underline{U}} + \underline{C}\dot{\underline{U}} + \underline{K}\underline{U} = \underline{R}(t) = \underline{PF}(t)$

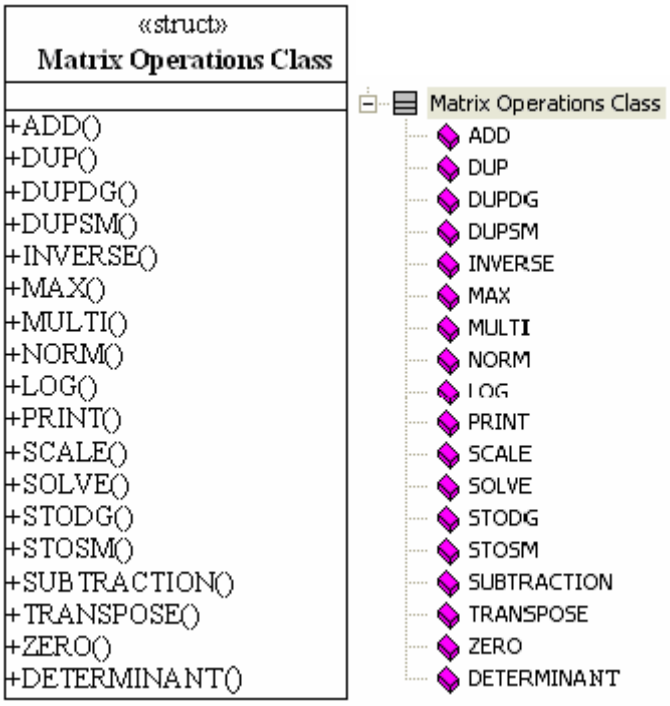


Figure 3.27 Matrix Operations Class

3.2.4 Verifications of Matrix Operations

Several example matrices were run and support values obtained from JU.CAL were verified with results obtained from other sources. In this section, three of these example problems are presented.

- Multiplication, Transpose and Inverse of Matrices:

  The matrices shown in Figure 3.28, in which the input data is taken from Kassimali (1999) page 29, were calculated by JU.CAL. The results of the calculation are shown in Figure 3.29. When the two sets of results are compared, it is concluded that the results obtained by JU.CAL are the same as those resulted by Kassimali (1999).

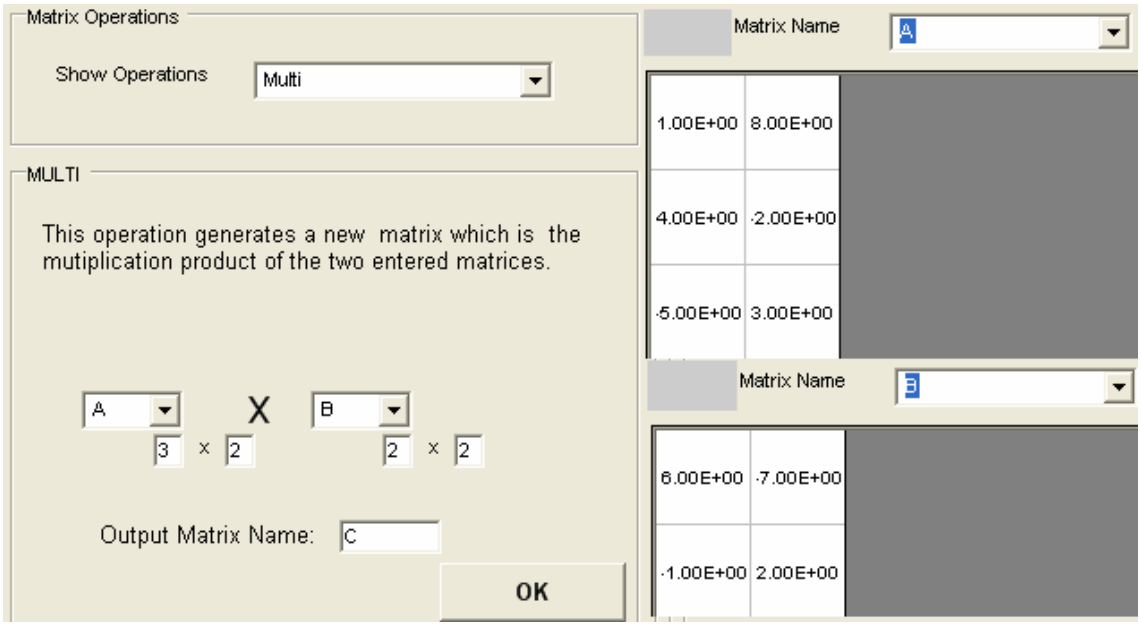


Figure 3.28 Example of Multiplication

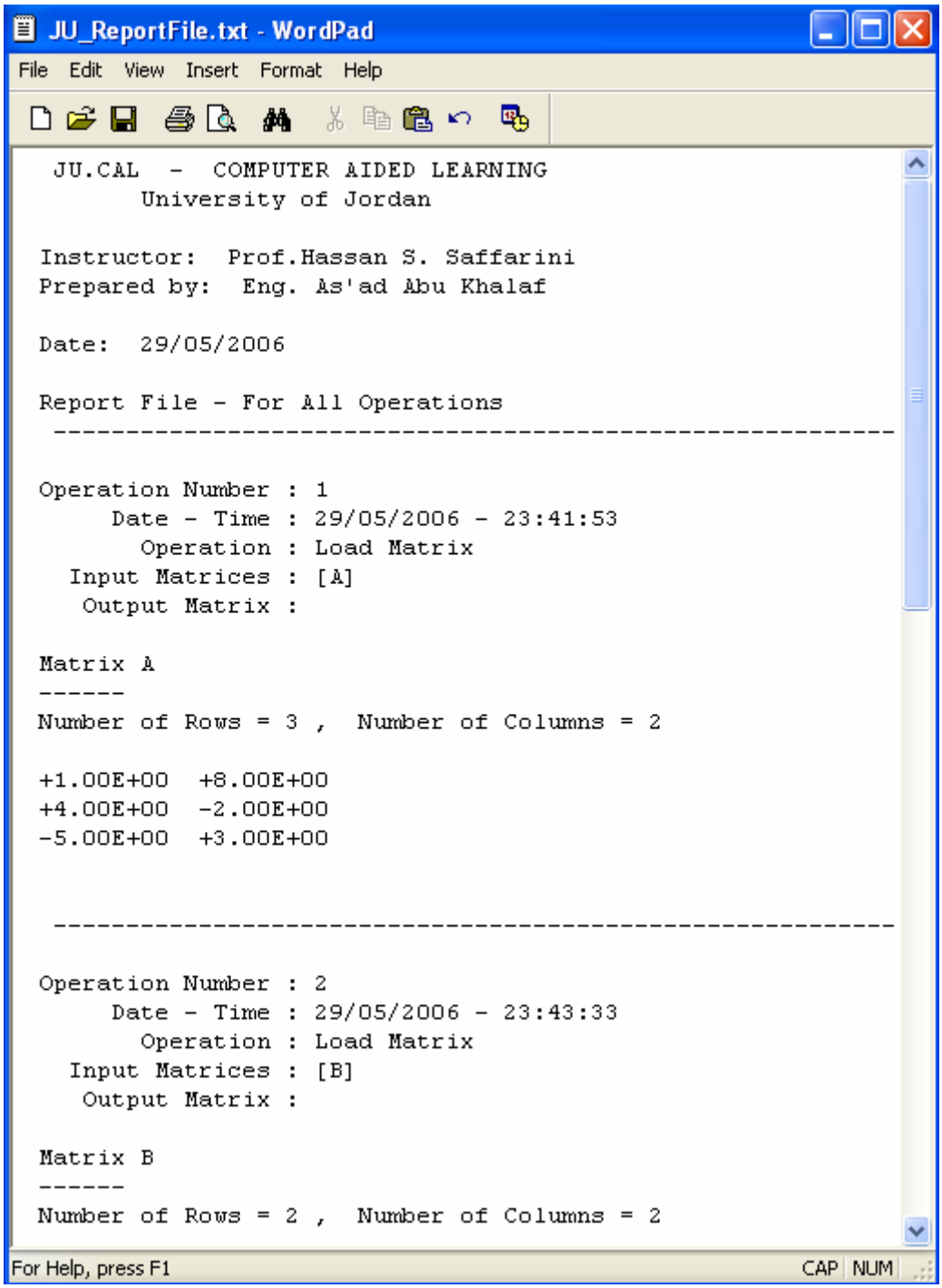


Figure 3.29 Results of Matrix Operations

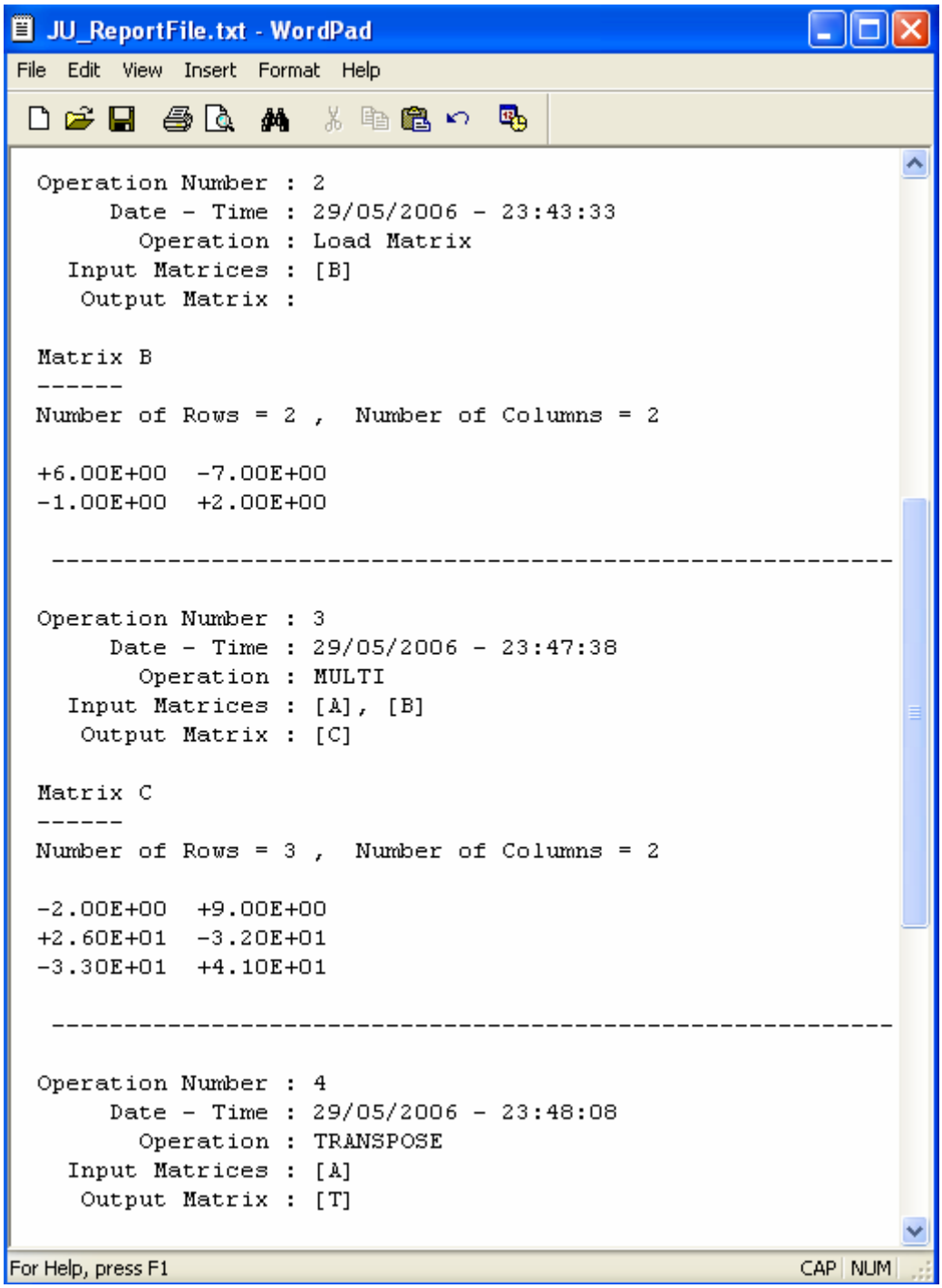


Figure 3.29-Cont. Results of Matrix Operations

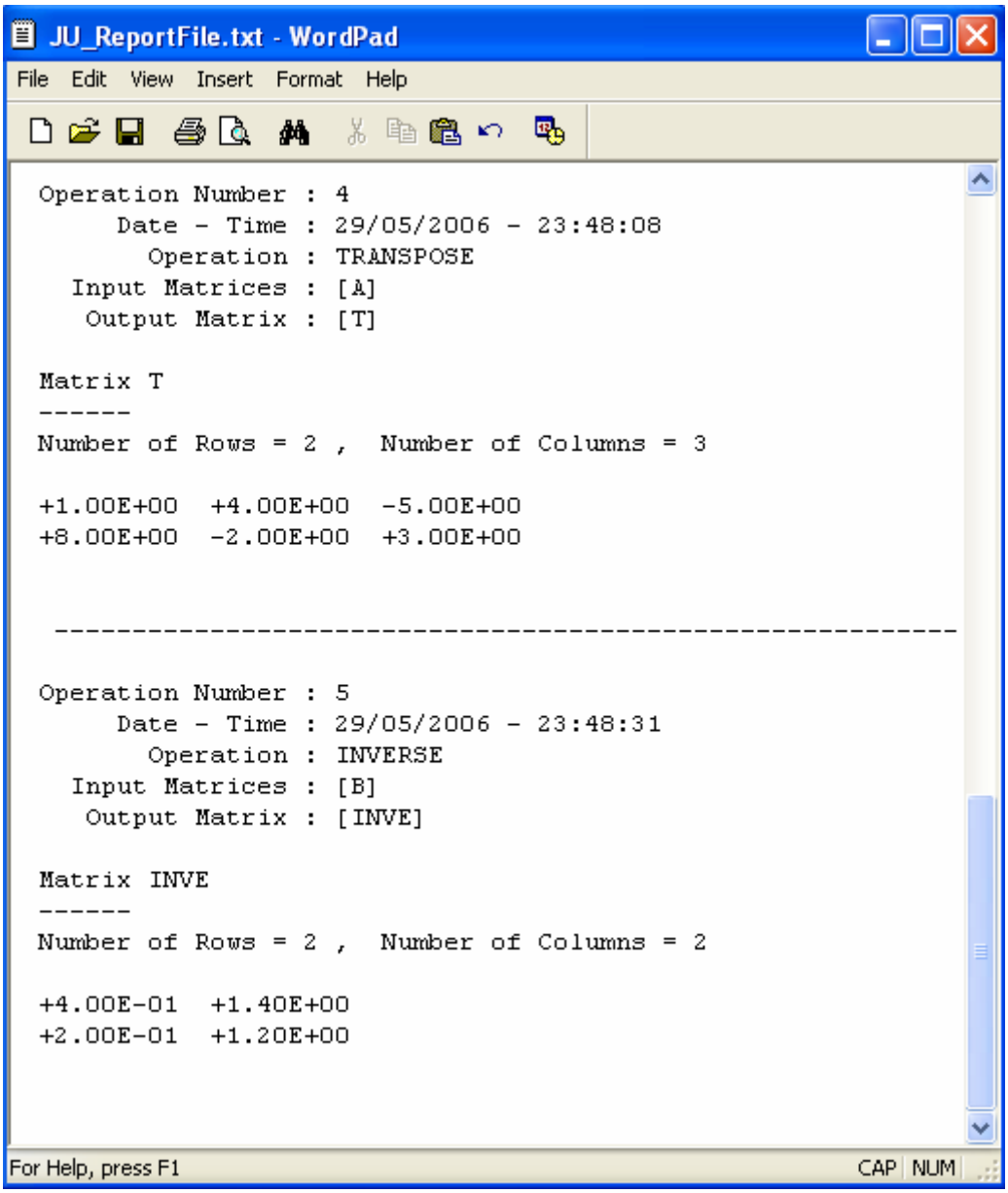


Figure 3.29-Cont. Results of Matrix Operations

3.3 Direct Stiffness Operations

3.3.1 Analysis Process

This section contains a series of operations which requires the user to identify the displacement degree of freedom at the end of each member and joint. The user must separately form and identify the stiffness of each element within a structure. After the total (global) stiffness matrix is initialized to zero, The ADDK operation is applied to each element stiffness matrix to form the total (Global) stiffness matrix. The MEMFRC operation is used to evaluate member forces after the joint displacements are evaluated. Only two different types of two-dimensional frame bending members are included, horizontal or vertical members with zero axial deformation and an arbitrary bending member with axial deformations.

The SLOPE operation is applied to form a 4 x 4 stiffness matrix for a beam or column member from the classical slope deflection equations. The properties of the member are defined using the interface frame. The sign conversion is defined as shown in Figure 3.30. The member forces are defined in terms of joint displacement by the following slope deflection equations.

$$M_1 = \frac{EI}{L}\left[4\theta_1 + 2\theta_2 - \frac{6}{L}[v_1 - v_2]\right] \tag{3.1}$$

$$M_2 = \frac{EI}{L}\left[2\theta_1 + 4\theta_2 - \frac{6}{L}[v_1 - v_2]\right] \tag{3.2}$$

$$v_1 = -v_2 = \frac{M_1 + M_2}{L} \tag{3.3}$$

Or in matrix form

$$\begin{bmatrix} M1 \\ M2 \\ V1 \\ V2 \end{bmatrix} = \frac{EI}{L} \begin{bmatrix} 4 & 2 & -\frac{6}{L} & \frac{6}{L} \\ 2 & 4 & -\frac{6}{L} & \frac{6}{L} \\ -\frac{6}{L} & -\frac{6}{L} & \frac{12}{L^2} & -\frac{12}{L^2} \\ \frac{6}{L} & \frac{6}{L} & -\frac{12}{L^2} & \frac{12}{L^2} \end{bmatrix} \begin{bmatrix} \theta_1 \\ \theta_2 \\ v_1 \\ v_2 \end{bmatrix}$$

Or symbolically [F] = [K].[U], where [K] is the 4 x 4 stiffness matrix.

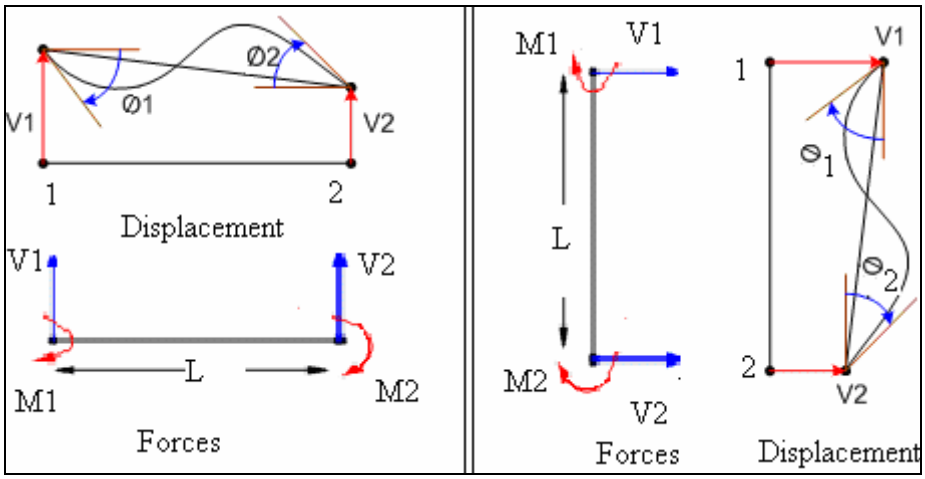


Figure 3.30 Sign Conversion of SLOPE Operation

The FRAME operation is applied to form a 6 x 6 member stiffness matrix for the two-dimensional frame member shown in Figure 3.31. [M_2] is 3 x 6 force displacement transformation matrix which is based on the positive definition of the element forces shown in Figure 3.31. These forces can be calculated from the following matrix equation, with the MEMFRC operation.

$$\begin{bmatrix} M_i \\ M_j \\ P \end{bmatrix} = [M_2] \begin{bmatrix} U_{xi} \\ U_{Yi} \\ U_{\theta i} \\ U_{xj} \\ U_{yj} \\ U_{\theta j} \end{bmatrix}$$

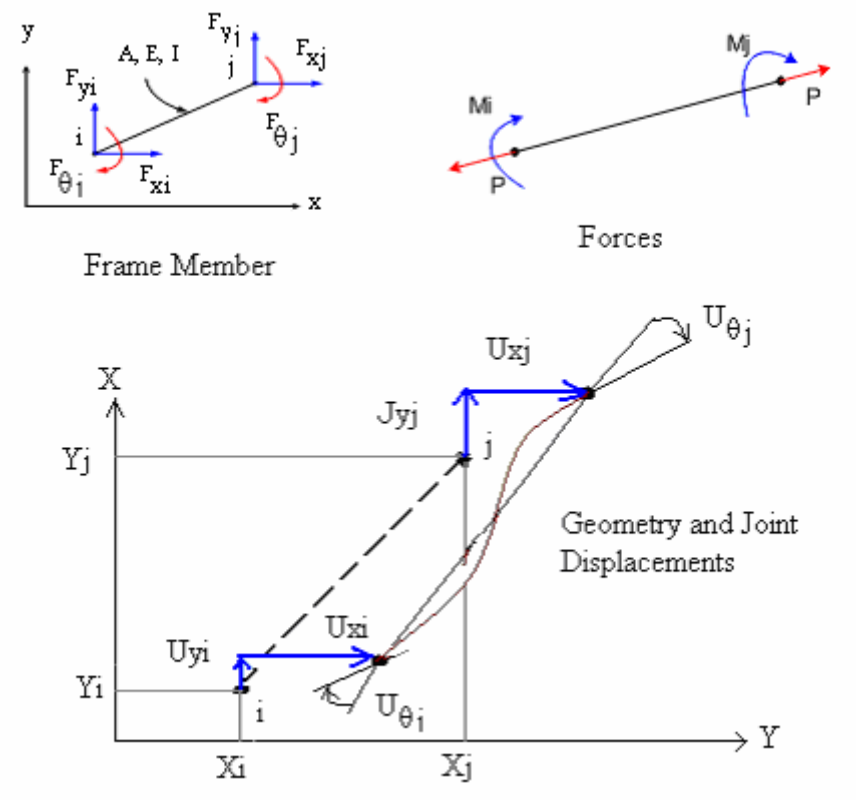


Figure 3.31 Sign Conversion of Frame Operation

3.3.2 Direct Stiffness Operations Interface

The following is a list of Direct Stiffness operations which are used to solve structural analysis problems.

- SLOPE: This operation is applied to form 4 x 4 stiffness matrices for a beam or column member from the classical slope deflection equations as shown in Figure 3.32.
- FRAME: This operation is applied to form 6 x 6 member stiffness matrices for the two-dimensional frame member as shown in Figure 3.33.

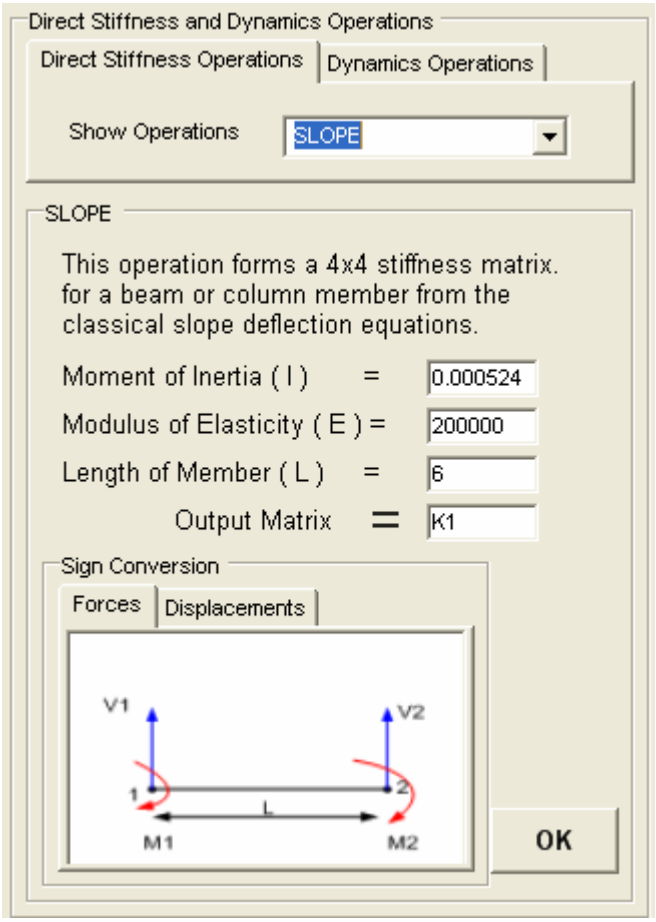


Figure 3.32 SLOPE Operation

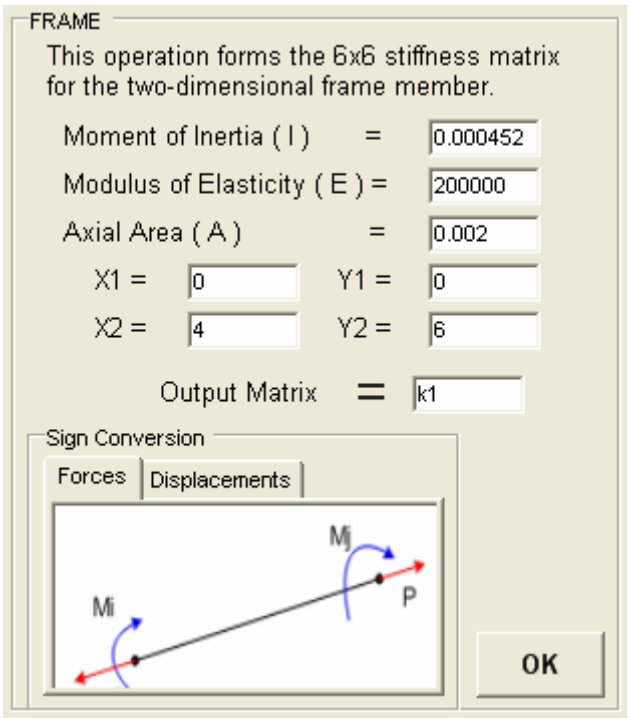


Figure 3.33 FRAME Operation

- ADDK: This operation adds the element stiffness matrix named k_1 to the total (Global) stiffness matrix named (K Global) as shown in Figure 3.34, where K Global was previously defined and initially set to zero. ID matrix (location matrix) is the name of the integer array in which the column number N1 contains the column numbers in the total stiffness matrix where the element stiffness terms are to be added.

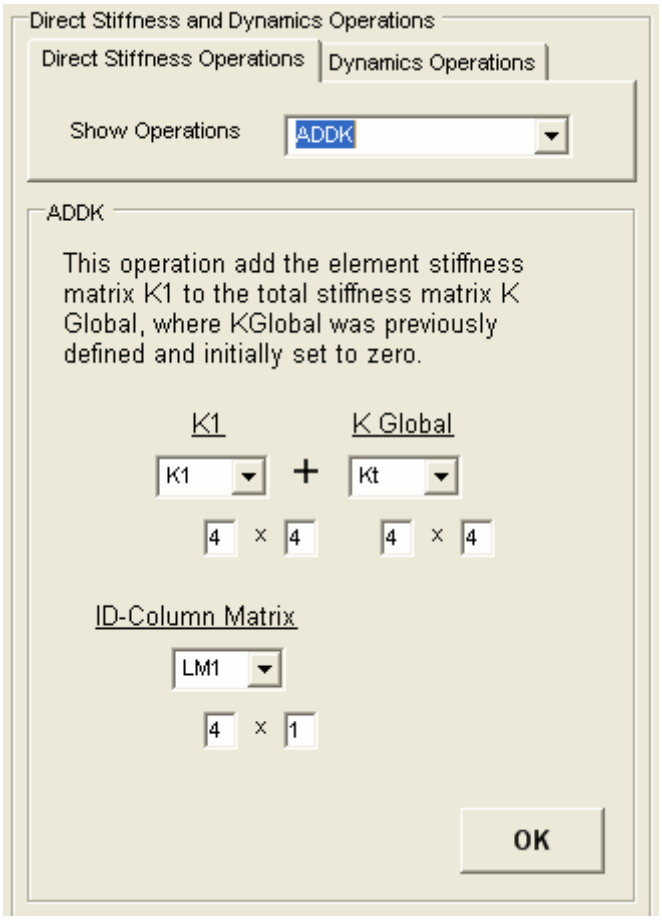


Figure 3.34 ADDK Operation

- MEMFRC: This operation is used to evaluate the member forces after the joint displacements are evaluated by using the SLOVE operation in the submenu of the matrix operations under the operations main menu. MEMFRC multiplies the member stiffness matrix by the joint displacements at that member and calculates the joint forces, moments at the member. The ID

column matrix is an integer array in which the column number (N) contains the row numbers in the displacement matrix as shown in Figure 3.35.

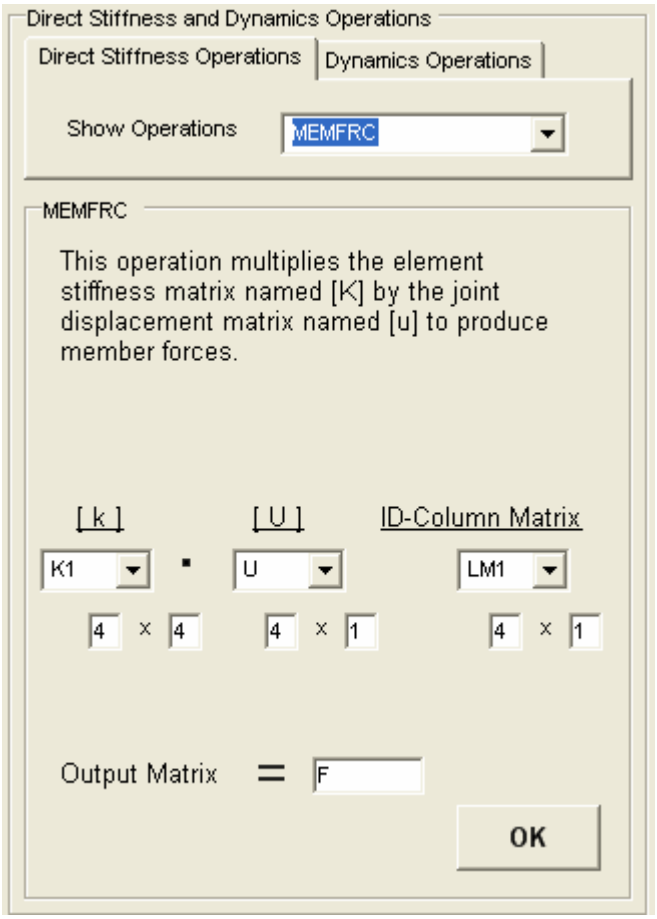


Figure 3.35 MEMFRC Operation

3.3.3 Direct Stiffness Operations Classes

A list of the methods used in the Direct Stiffness Operations Classes is shown in Figure 3.36. It contains some of the operations employed in structural analysis and it can be used for future development of software by using the concepts of object oriented programming to add more operations like finite elements or other developments. The listings of these classes are given in appendix A.

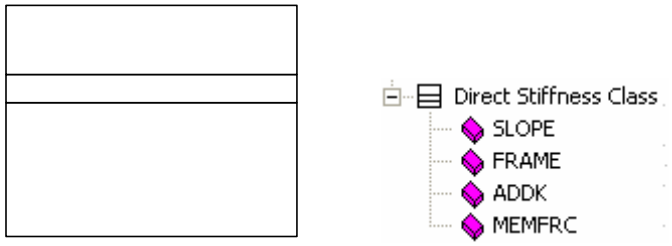


Figure 3.36 Direct Stiffness Class

3.3.4 Verifications of Direct Stiffness Operations

Several example frames were analyzed and the member forces and support values obtained from the DSM operation were verified with results obtained from other sources. In this section, one of these example problems is presented.

Beam Analysis Example:

The beam analysis example shown in Figure 3.37 was taken from Hibbeler (2002). This beam was analyzed by JU.CAL and the results of the analysis are shown in Figure 3.38. The reactions at the supports of the beam from Hibbeler (2002) are shown in Table 3.3. When the two sets of results are compared as shown in Table 3.3, it can be seen that the reaction forces results obtained from JU.CAL are the same as those given by Hibbeler (2002).

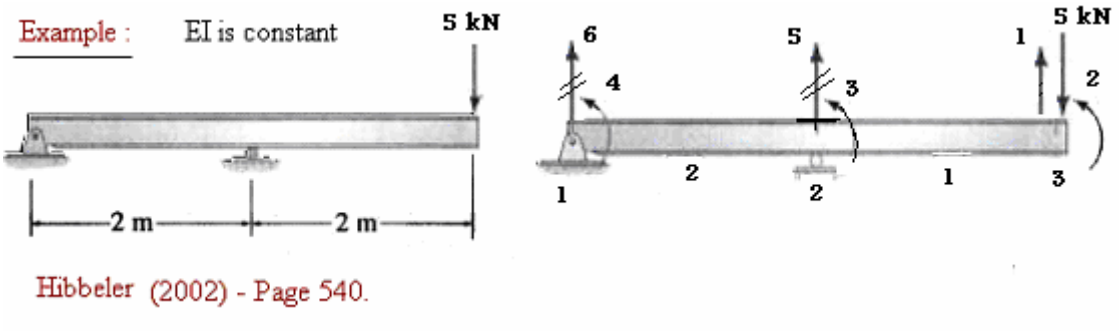


Figure 3.37 Beam Analysis Example

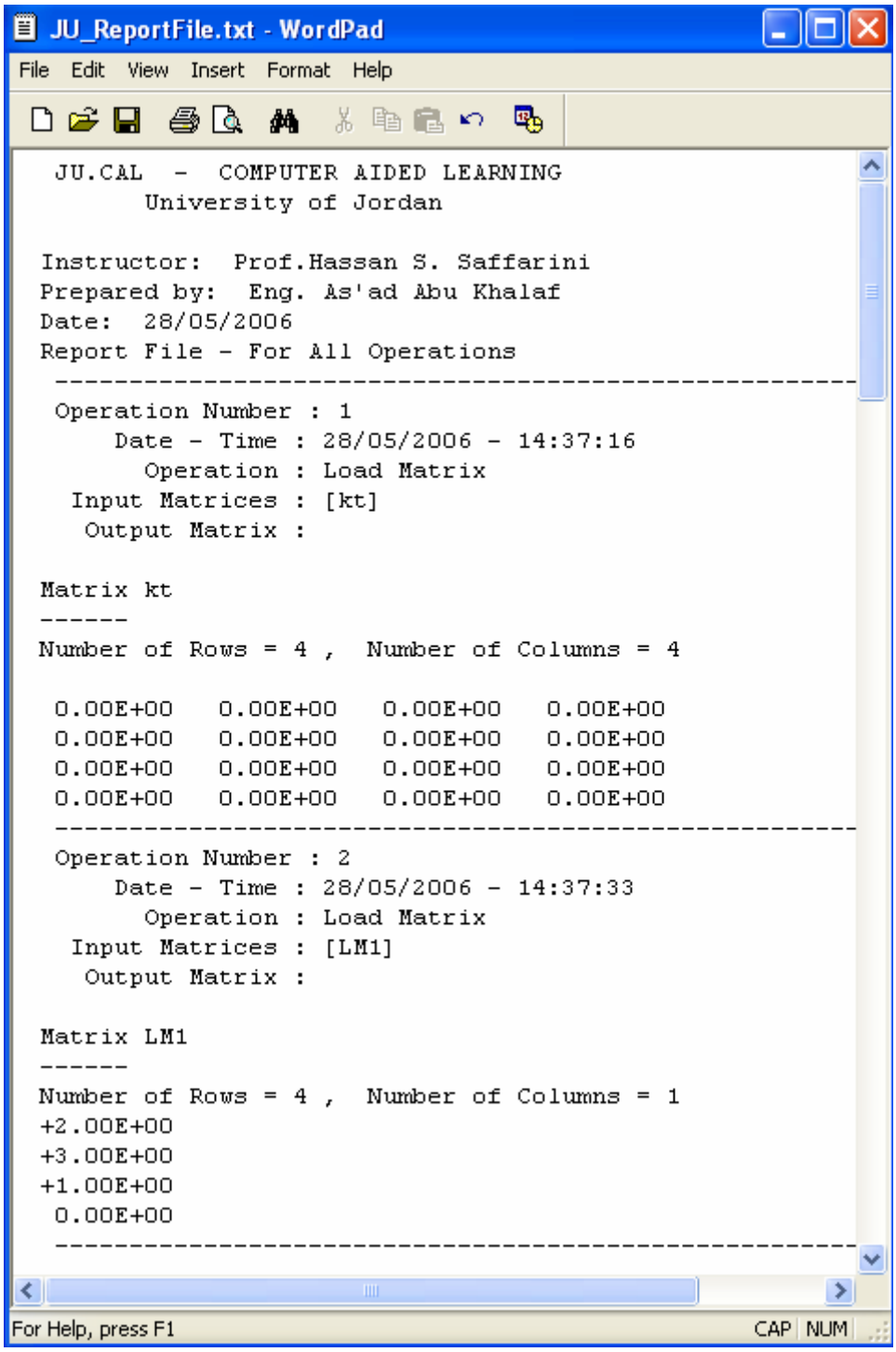


Figure 3.38 Beam Analysis Results

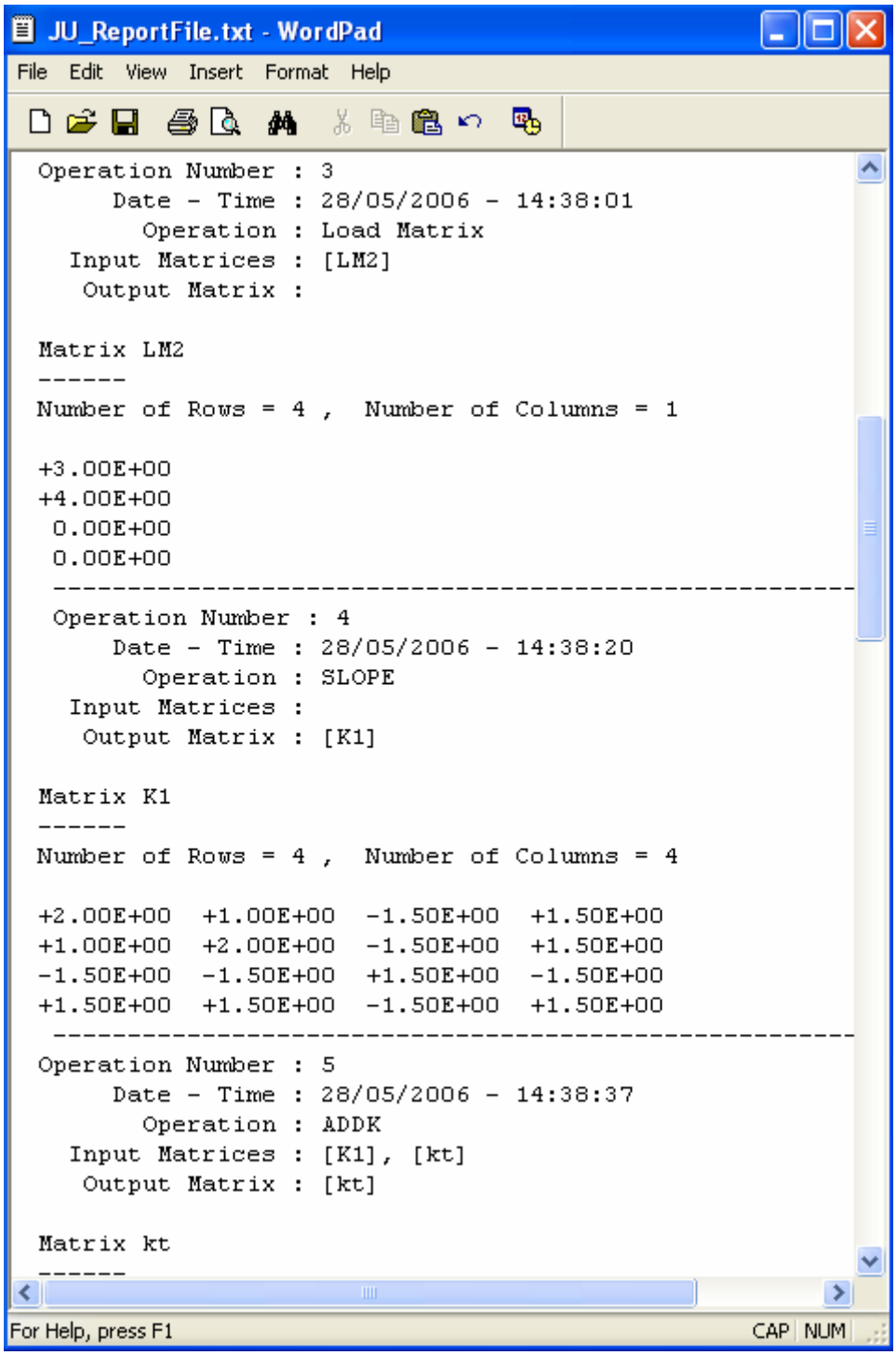


Figure 3.38-Cont. Beam Analysis Results

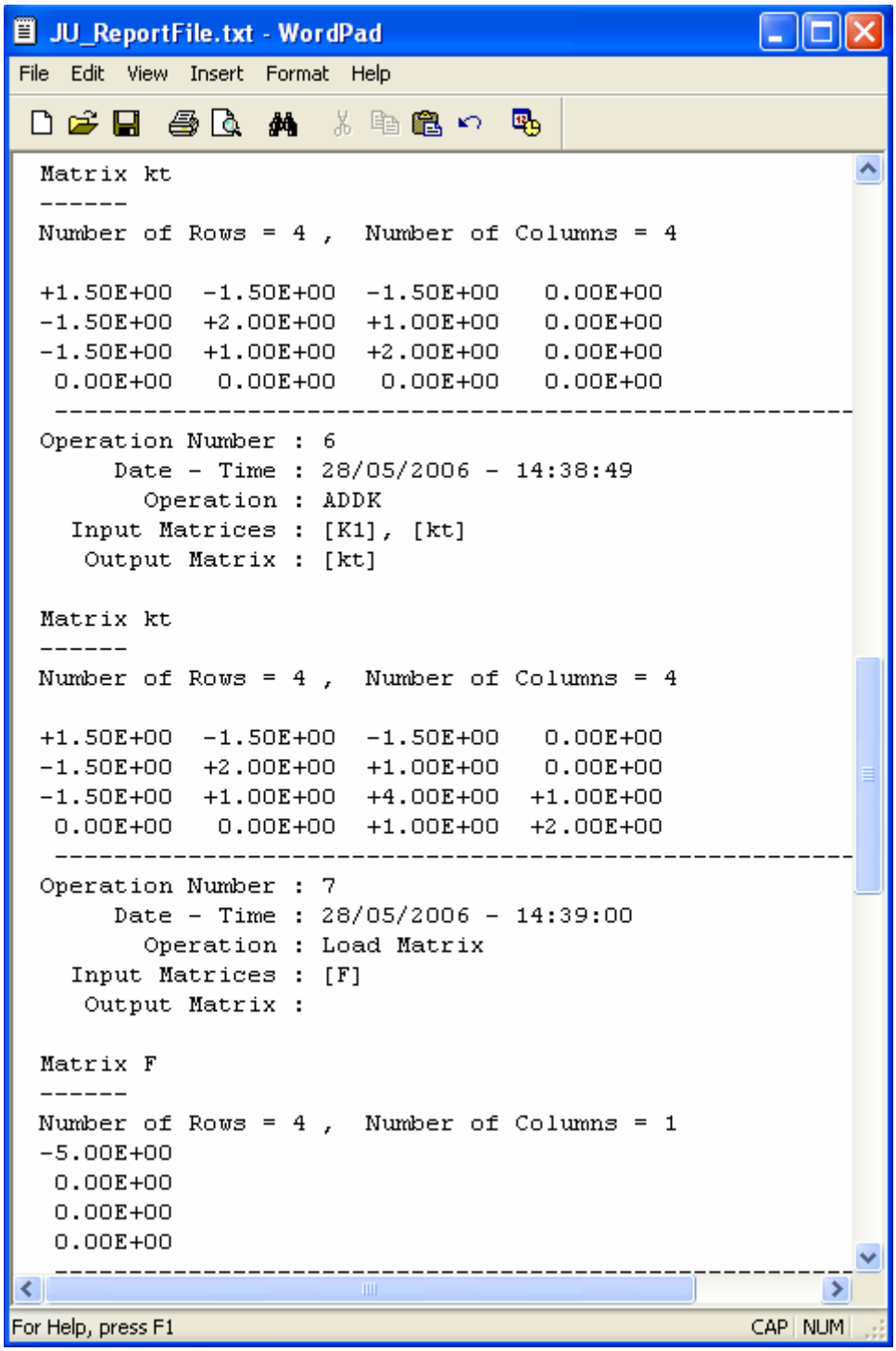


Figure 3.38-Cont. Beam Analysis Results

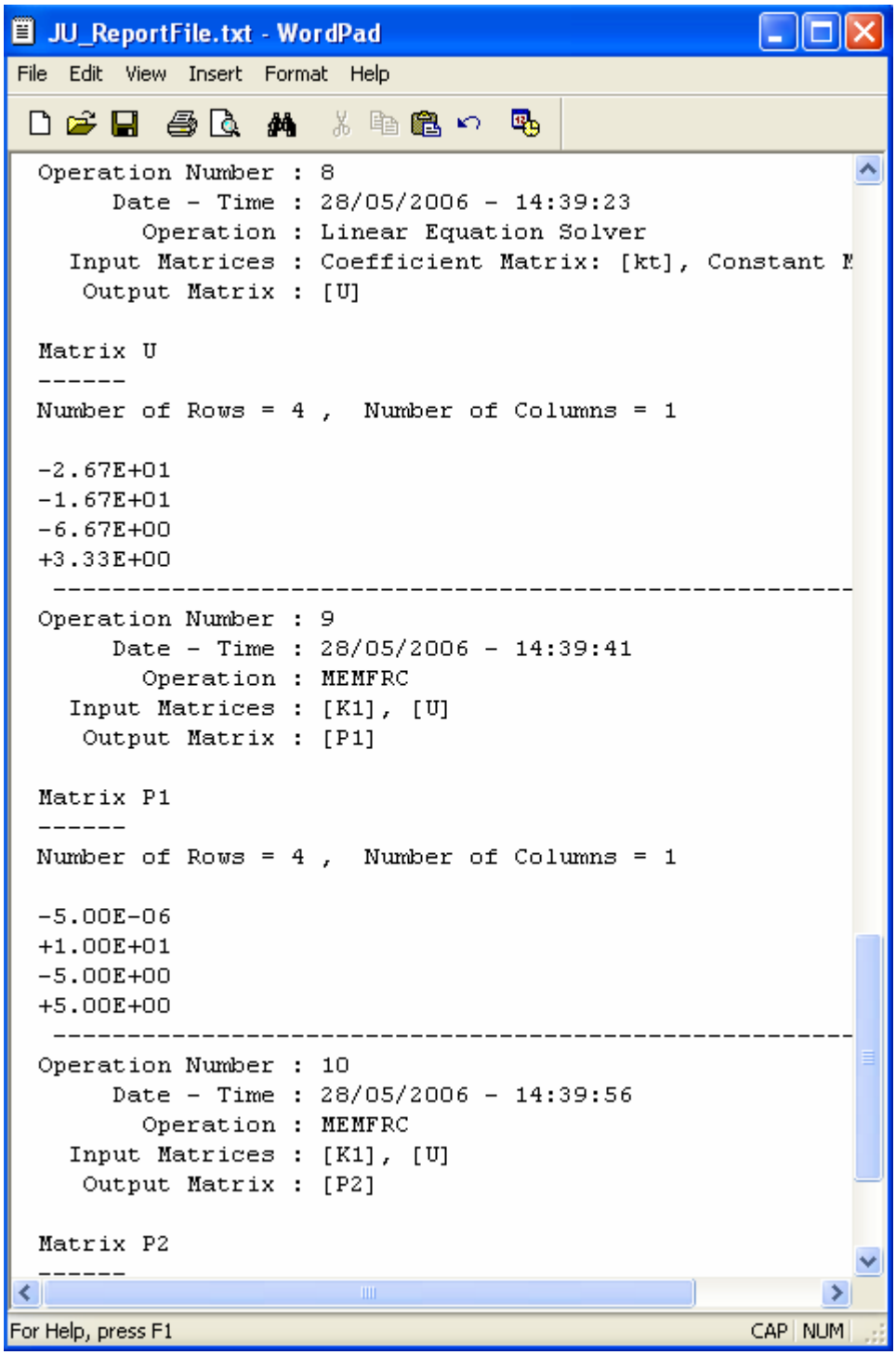


Figure 3.38-Cont. Beam Analysis Results

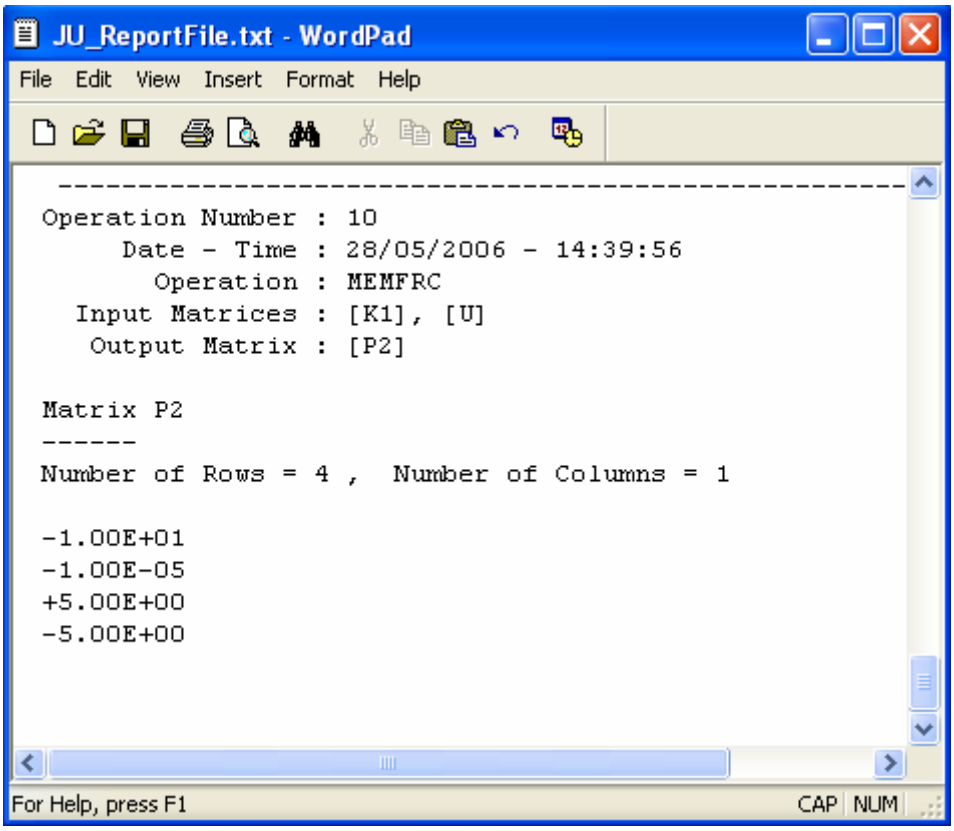


Figure 3.38-Cont. Beam Analysis Results

Table 3.3 Example- Comparison Results.

Support	Reaction (Hibbeler)	Reaction (JU.CAL)
1	10 kN	10 kN
2	-5 kN	-5 kN

3.4 Dynamic Operations

3.4.1 Analysis Process

The following operations are designed to evaluate the dynamic response of structures subjected to arbitrary time dependent loads. If these operations are used in connection with the standard matrix operations and the structural analysis operations, a dynamic analysis is a relatively simple procedure. The user may be using a direct step by step integration method to solve the equations of motion. In addition, the Eigenvalue problem can be solved. The most common and convenient form for time dependent data is to be specified as straight line segments of load functions between given time points. Therefore, an operation which generates values at equal interval is necessary. Another common characteristic of time varying loads on structures is that it is normally possible to represent the load at all points on the structure by the product of two matrices. A column matrix indicating the spatial distribution of loads multiplied by a row matrix which indicates the values of a function at various times. If a more complicated loading is required it is possible to perform more analysis, each within the restrictions of the program, and then add the results of each analysis.

In addition to several other matrix operations within JU.CAL the next section describes operations that have been added for the major purpose of performing dynamic analysis like FUNG, STEP and EIGEN operations.

3.4.2 Step-by-Step Integration Method

The direct integration of the linear dynamic equation of motion is a simple approach which can have considerable advantages for some problems. The basic equation is satisfied at discrete point in time, 0, Δt, $2\Delta t$, $3\Delta t$, …t, t+ Δt. The solution starts from a point in time where the displacements, velocities and accelerations are known. Based on

an assumption on the behavior of the system during the next small increment of time the displacements, velocities and accelerations at the next point in time can be evaluated. Many different methods have been developed for this purpose. However, Newmark-Wilson method for STEP operation was used to solve dynamic response of structural system of the following linear matrix equations of motion:

$$\underline{M}\ddot{\underline{U}} + \underline{C}\dot{\underline{U}} + \underline{K}\underline{U} = \underline{P}(t) \qquad (3.4)$$

Where:

[M] is a mass matrix.

[C] is a damping matrix.

[K] is a stiffness matrix

$[U_o]$ is a vector of displacements.

$[\dot{U}_o]$ is a vector of velocities.

$[\ddot{U}_o]$ is a vector of accelerations.

P(t) is a vector of load.

This method is based on the following expressions for the velocity and displacement at the end of the time interval:

$$\dot{U}_{t+\Delta t} = \dot{U}_t + \Delta t(1-\delta)\ddot{U}_t + \Delta t\delta \ddot{U}_{t+\Delta t} \qquad (3.5)$$

$$U_{t+\Delta t} = U_t + \Delta t\dot{U}_t + \Delta t^2(\frac{1}{2}-\alpha)\ddot{U}_t + \Delta t^2\alpha \ddot{U}_{t+\Delta t} \qquad (3.6)$$

Where α and δ are selected to produce the desired accuracy and stability. If $\delta = \frac{1}{2}$ and $\alpha = \frac{1}{6}$ the well known linear acceleration is produced, which is also a conditionally stable method. One of the most widely used methods is the constant average acceleration method

($\delta = \frac{1}{2}$ and $\alpha = \frac{1}{4}$) which is an unconditionally stable method without numerical damping. This method is called an implicit integration method since it satisfies the equilibrium equations of motion at time t+ Δt, or

$$\underline{M}\underline{\ddot{U}}_{t+\Delta t} + \underline{C}\underline{\dot{U}}_{t+\Delta t} + \underline{K}\underline{U}_{t+\Delta t} = \underline{F}_{t+\Delta t} \qquad (3.7)$$

An implicit method is defined as one in which the new response values calculated in each step depend only on quantities obtained in the preceding step, so that the analysis proceeds directly from one step to the next. The expressions giving the new values for a given step include one or more values pertaining to that same step, so that the trial values of the necessary quantities must be assumed and then these are refined by successive iterations (Clough, 1993).

Equation (3.7) can be solved by iteration; however Equation (3.5), (3.6) and (3.7) can be combined into a step-by-step algorithm which involves the solution of a set of equations at each time step of the form:

$$K^* U_{t+\Delta t} = F^* \qquad (3.8)$$

The Wilson θ method is a technique which can be used to modify the basic Newmark method in order to increase the stability limits and to add numerical damping. The θ method was first applied to the linear acceleration method in order to improve stability and has been used to damp out high frequency oscillations which often develop in linear and non-linear step-by-step integration. The technique involves using the Newmark method to find the solution at t+ θ Δt, then, based on linear acceleration, calculating the results at t+Δt for use as initial conditions for the next time step. With θ =1 the approach is the standard Newmark method. This method is unconditionally stable if θ > 1.37.

3.4.3 Dynamic Operations Interface

The following is a list of Dynamic operations which have been added for the main purpose of performing dynamic analysis.

- FUNG: This operation generates the matrix named [M_2] as shown in Figure 3.39 which contains values at equal time intervals, of the function specified in the array named [M_1]. The array must be a 2 by K array of the form

$$M_1 = \begin{bmatrix} t_1 & t_2 & t_3 & t_4 & \dots & t_k \\ f_1 & f_2 & f_3 & f_4 & \cdots & f_k \end{bmatrix}$$

$$M_2 = \begin{bmatrix} t_1 & t_1 + \Delta t & t + 2\Delta t & \cdots \\ f_1 & f(t_1 + \Delta t) & f(t_1 + 2\Delta t) & \cdots \end{bmatrix}$$

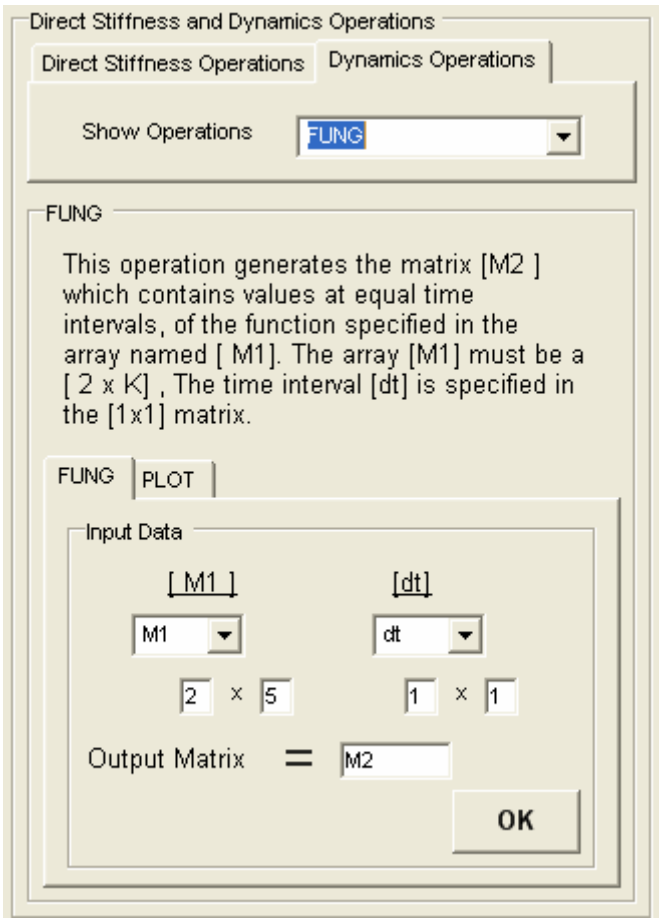


Figure 3.39 FUNG Operation

This numerically represents a function of the form as shown in Figure 3.40.

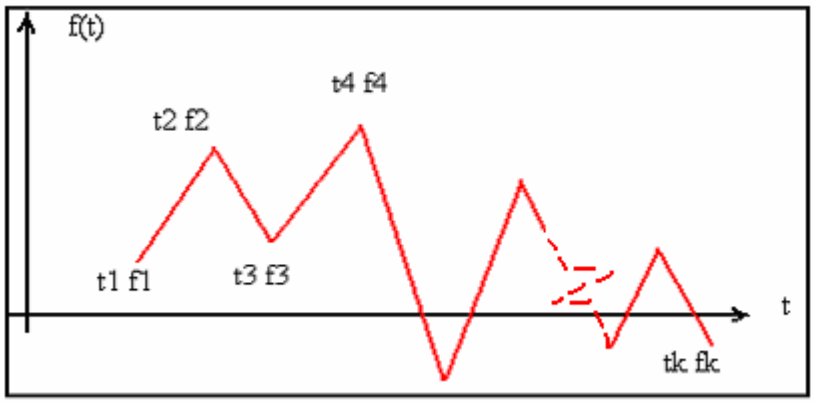


Figure 3.40 Functions in FUNG Operation

- STEP: This operation calculates the dynamic response of a structural system using direct step by step integration of the equation of motion 3.4 as shown in Figure 3.41.

  Different values of δ, α and θ will allow the user to select different methods of step by step integration as shown in Table 3.4.

Table 3.4 Methods of Step by Step Integration.

	δ	α	θ
Newmarks average Acceleration	1/2	1/4	1.00
Linear Acceleration	1/2	1/6	1.00
Wilson's θ Method (low damping)	1/2	1/6	1.42
Wilson's θ Method (high damping)	1/2	1/6	2.00

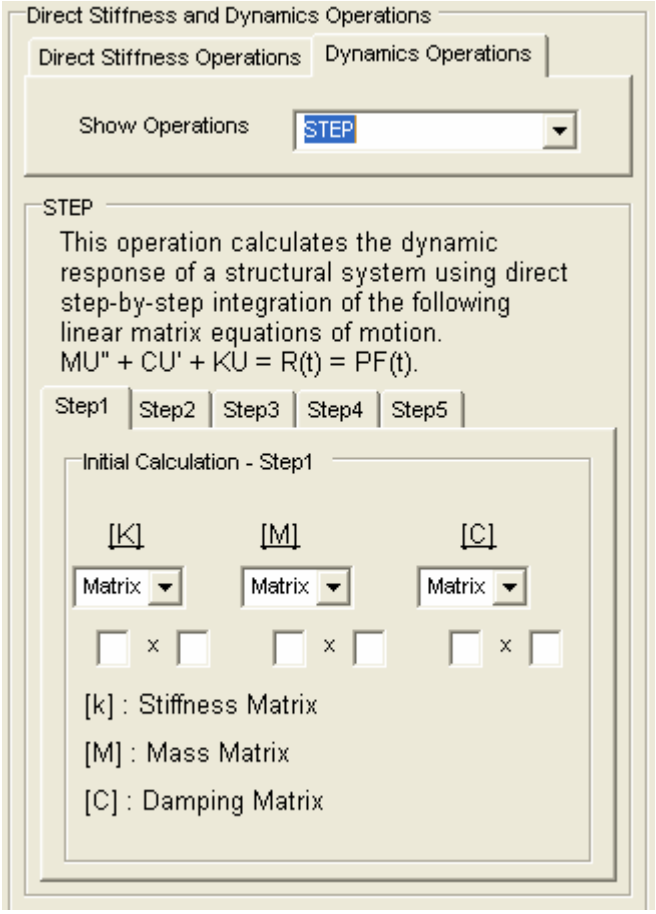


Figure 3.41 STEP Operation

- EIGEN: This operation solves the following eigenvalues problem:

$$K\phi = M\phi\lambda \tag{3.10}$$

The program reduces the problem to standard eigenvalue form by the following transformation:

$$K^* = m^T Km \tag{3.11}$$

Where

$$I = m^T M.m \tag{3.12}$$

In which

$$m_i = \frac{1}{\sqrt{M_{ii}}}$$

The calculated mode shapes ϕ are normalized as follows:

$$\phi^T M.\phi = I \tag{3.13}$$

$$\phi^T K.\phi = \lambda \tag{3.14}$$

The program uses the standard Jacobi diagonalization method to solve for all eigenvalues and eigenvectors. Figure 3.42 shows EGIEN operation.

- PLOT: This operation is used to plot any function as shown in Figure 3.43.

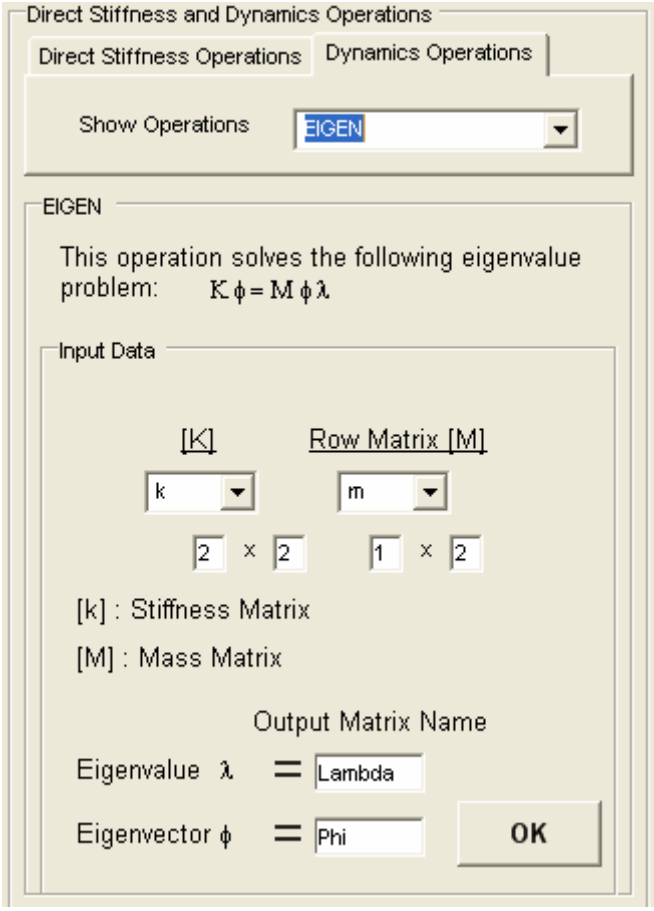


Figure 3.42 EIGEN Operation

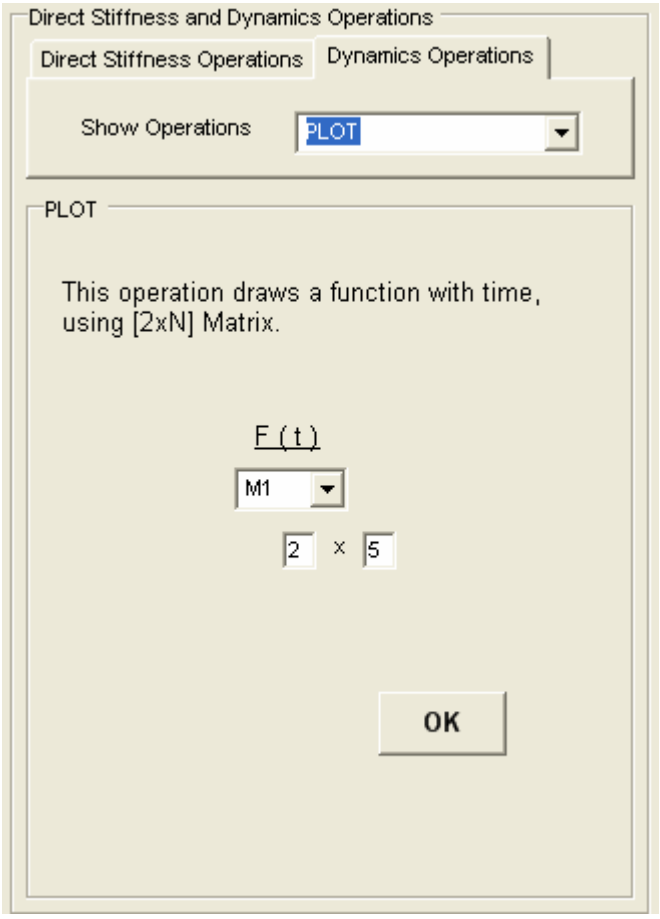


Figure 3.43 PLOT Operation

3.4.4 Dynamic Operations Classes

The Dynamic Operations Classes are shown in Figure 3.44. It contains the major functions for performing dynamic analysis to understand the behavior of structure under time dependent loading. These operations can be used to develop the program future employing OOP to add new tools and operations for dynamic nonlinear analysis. The listings of these classes are given in appendix A.

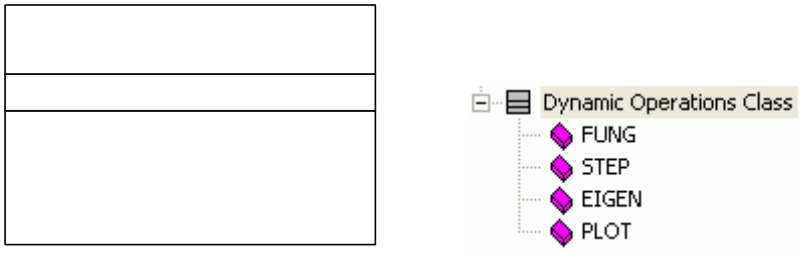


Figure 3.44 Dynamic Operations Class

3.4.5 Verifications of Dynamic Operations

Consider the eigenproblem $K\phi = M\phi\lambda$ taken from Wilson (1976) page 367, where:

$$K = \begin{bmatrix} 5 & -2 \\ -2 & 2 \end{bmatrix} \ldots M = \begin{bmatrix} \frac{5}{4} & 0 \\ 0 & \frac{1}{5} \end{bmatrix}$$

This problem is solved by JU.CAL. The results of the analysis are shown in Figure 3.45. The results of the eigenproblem from Wilson (1976) are shown in Table 3.5. When the two sets of results are compared as shown in Table 3.5, it can be seen that the eigenvalue and eigenvector results obtained from JU.CAL are close to those given by Wilson (1976).

Table 3.5 Results of Eigenproblem from Wilson (1976).

Results by	λ_1	λ_2	ϕ_1	ϕ_2
Wilson	2	12	0.8	0.4
			1	-2
JU.CAL	2.03	12	0.777	0.444
			1.11	-1.94

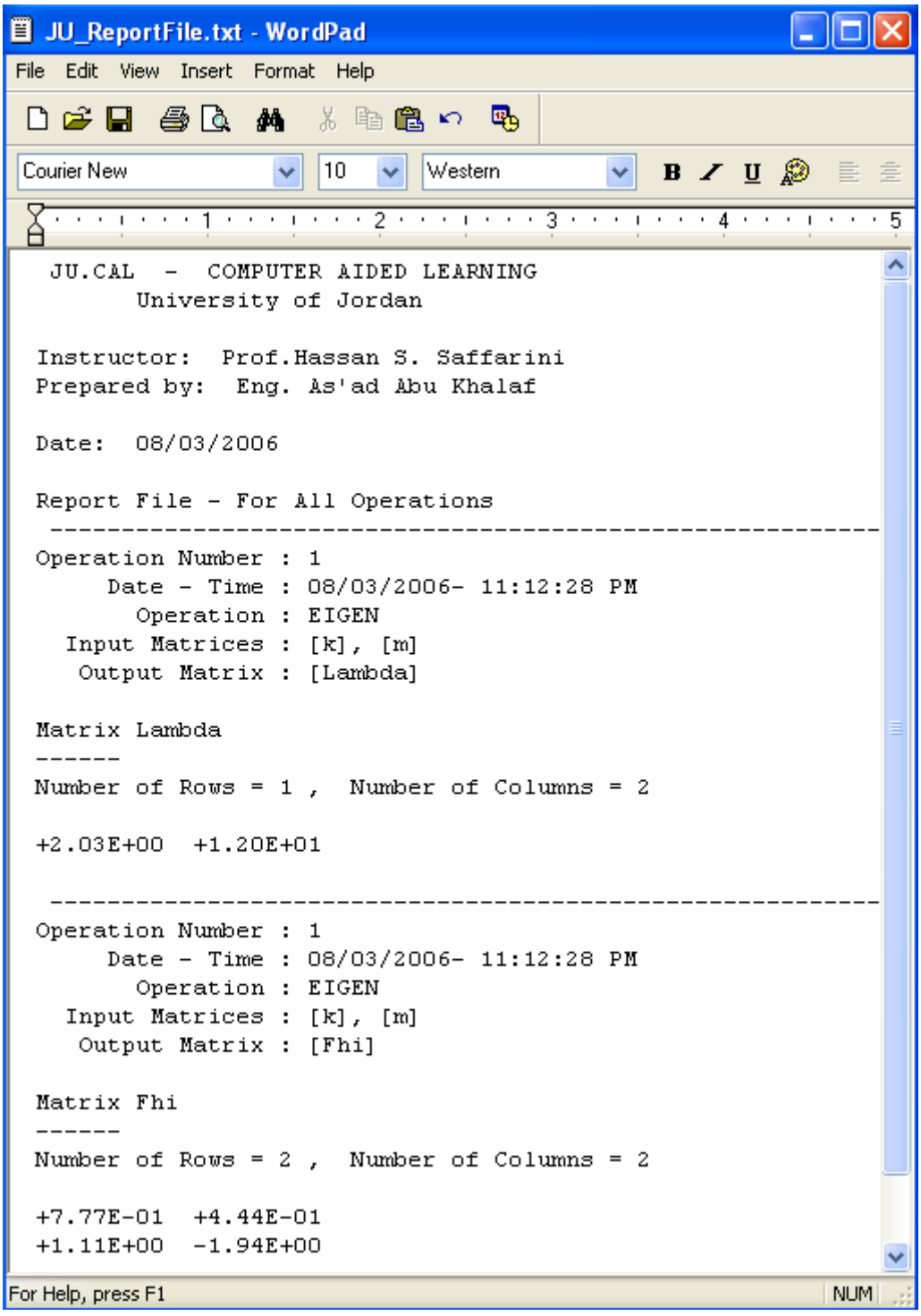


Figure 3.45 Example of EIGEN Operation

Dynamic Analysis of Multistory Frames (DAF)

4.1. Introduction

In this chapter, the programming of a new software is introduced and presented. JU.DAF is a program designed for the purpose of performing dynamic analysis for various types of portal frames in the elastic range. A brief description of DAF is presented. The step-by-step analysis method was implemented in the program to perform the necessary analysis. The program interface is also presented. Finally, verification of some known results from references is presented and including the comparison for the results of DAF with SAP2000.

4.2 Overview of DAF

DAF is used to solve any type of general or prescribed time history dynamic loading for the analysis of multistory frames that consists of user defined bays and stories of the desired height and width. The girders are assumed to act as rigid, and their movements are restricted in the horizontal direction as shown in Figure 4.1.

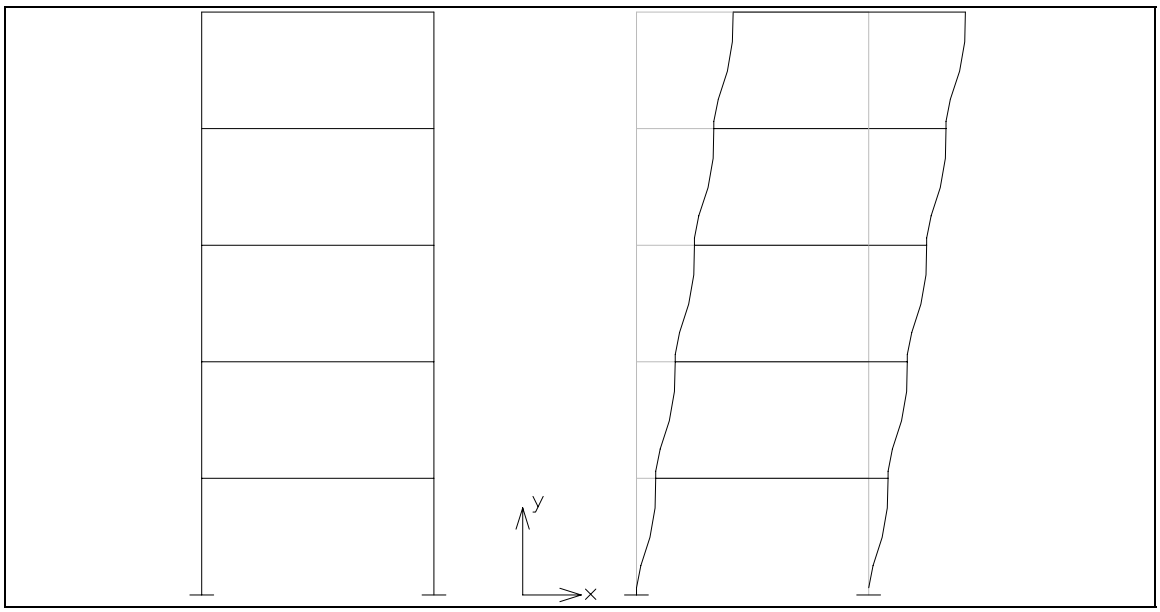


Figure 4.1 Multistory Frames.

The total mass is assumed to be lumped at the girder's position. Columns will be assumed to be weightless as compared with the high mass of the floors. The axial deformations will also be neglected owing to their large axial stiffness (EA).

4.3 Theories used in DAF

4.3.1 Rayleigh Method

The Rayleigh method is employed for selecting the true vibration shape. However, prior to commencing the Rayleigh method calculations, an assumed deflected shape will be used. This shape will be the result of loading each floor by its own mass. Subsequently, the mass for each floor will be modified by multiplying by the resulting normalized shape and reloading again to evaluate the final vibration shape. This was only the first improvement in the Rayleigh method (R01) and no further improvements are made (Clough and Penzien, 1993). The stiffness matrix of a frame member is evaluated as follows:

$$K_{storey} = \Sigma\ (12EI/L^3) \qquad \text{for all columns in the storey} \qquad (4.1)$$

Where:

K = stiffness

E = modulus of elasticity

I = moment of inertia

L = length of member

4.3.2 Generalized SDOF Systems

After the selection of the Rayleigh vibration shape and assigning problem features such as mass, damping, loads and stiffness, the generalized properties will be evaluated as follows:

$$m^* = \Sigma\, mi * \psi_i^2$$

$$K^* = \Sigma\, Ki * (\psi_i - \psi_{i-1})^2$$

$$C^* = \Sigma\, Ci * \psi i^2$$

$$P(t)^* = f(t)\ \ \Sigma\, Pi * \psi_i^2$$

Where:

m^* = generalized mass

C^* = generalized damping

K^* = generalized stiffness

$P(t)^*$ = generalized effective load

ψi = shape function

mi = mass

Ki = stiffness

Ci = damping

Pi = applied loading

f(t) = dynamic load variation

Recommended damping values are given in Table 4.1 for two levels of motion.

4.3.3 Step-By-Step analysis using the Newmark Beta method

This method is presented and discussed in chapter 3, but here are some important features that were used in the program:

i. Linear acceleration assumption rather than constant acceleration was used.

ii. The linear acceleration method is only conditionally stable, it will be unstable unless $\Delta t/T \leq 0.55$; the program uses $\Delta t /T < 0.2$ for reliable results. Where: Δt = Time increment in sec, T = Fundamental period of vibration.

Table 4.1 Damping Ratio

Stress Level	Type and Condition of Structure	Damping Ratio (%)
Working stress, no more than about (0.5) yield point	Welded steel, prestressed concrete, well-reinforced concrete (only slight cracking)	2 - 3
	Reinforced concrete with considerable cracking	3 - 5
	Bolted and/or riveted steel, wood structures with nailed or bolted joints	5 - 7
At or just below yield point	Welded steel, prestressed concrete (without complete loss in prestress)	5 - 7
	Prestressed concrete with no prestress left	7 - 10
	Reinforced concrete	7 - 10
	Bolted and/or riveted steel, wood structures with bolted joints	10 - 15
	Wood structures with nailed joints	15 - 20

Source: N.M. Newmark, and W.J.Hall, Earthquake Spectra and Design, Earthquake Engineering Research Institute, Berkeley, Calif.,1982

4.4 DAF Interface

This program is also intuitive and easy to use. The user follows logical steps in solving any problem as to simplify the usage of the program; The Rayleigh vibration shape must be selected by the program before assigning the storey masses, damping and loads, otherwise, the program will not respond (of course the program will choose the shape, but the user must activate the process). A preview of DAF main screen is shown in Figure 4.2.

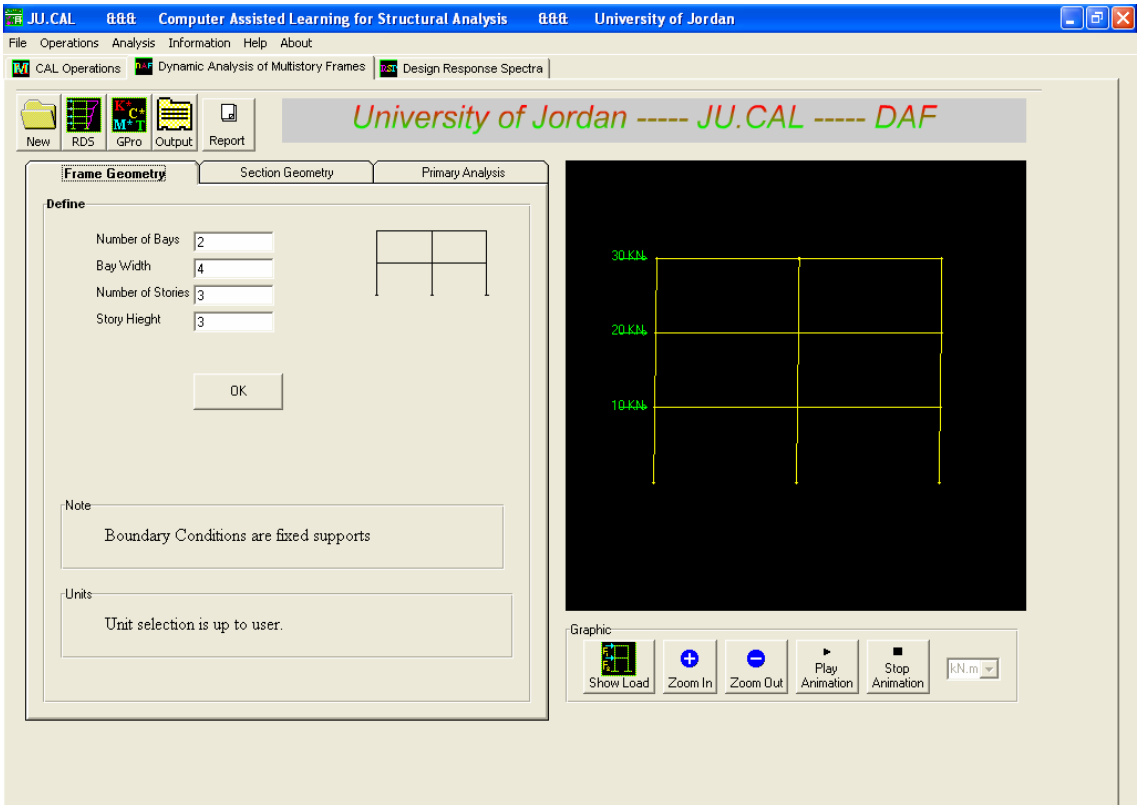


Figure 4.2 DAF Interface

The following is a brief description of the interface of JU.DAF:

Figure 4.3 shows the structure geometry input screen, which enables the user to enter the number of bays and stories as well as bay widths and storey heights.

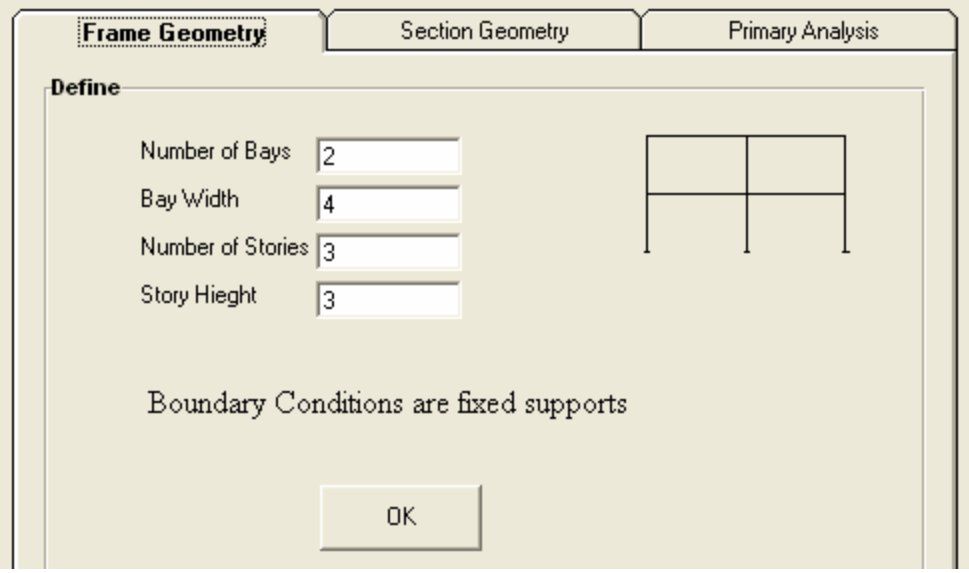


Figure 4.3 Structure Geometry.

The Section geometry screen shows the moment of inertia for columns at each storey as shown in Figure 4.4. This enables the user to define the columns moment of inertia; the user should sum up all the moments of inertia of all the column sections in the same storey and place the sum as a one column**.**

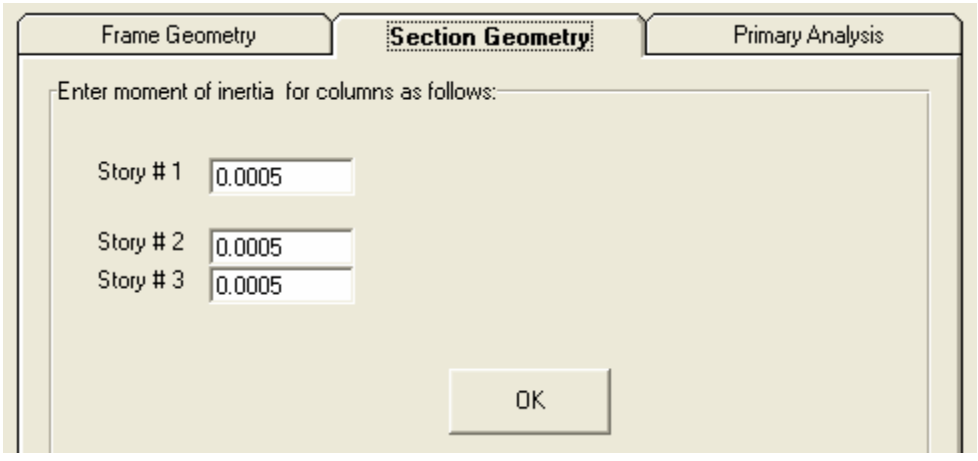


Figure 4.4 Section Geometry.

Assign modulus of elasticity, damping ratio and primary analysis as shown in Figure 4.5. Primary analysis enables the computer to calculate a good approximation of the

vibration shape as explained earlier. This sub menu will be disabled if the user does not provide any information about the modulus of elasticity and section properties of columns.

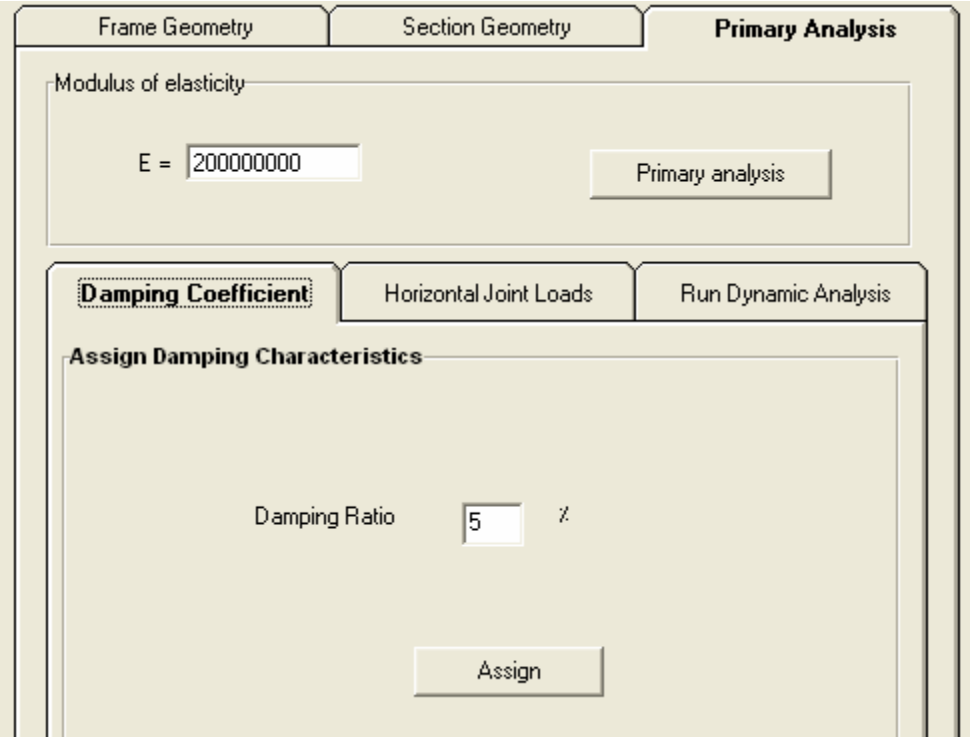


Figure 4.5 Primary Analysis.

Assign horizontal joint loads and storey mass for each storey as shown in Figure 4.6 and Figure 4.7 respectively. Assign Load Multiplier for the structure as shown in Figure 4.8, which consists of two columns, the first one is for the time increment and the second one is for providing the multiplier of the previously entered horizontal loads at each time increment, as those loads were considered to be maximum amplitudes of the force and this subroutine enables them to be functions of time (time history loads). Finally, the Run Dynamic Analysis button command executes the Step-By-Step analysis method and draws the deflected vibration shape as shown in Figure 4.9. This button will be disabled if the user does not provide any information about the modulus of elasticity and section properties of columns; and also the program will not run the analysis unless the user assigns mass, loads, …etc.

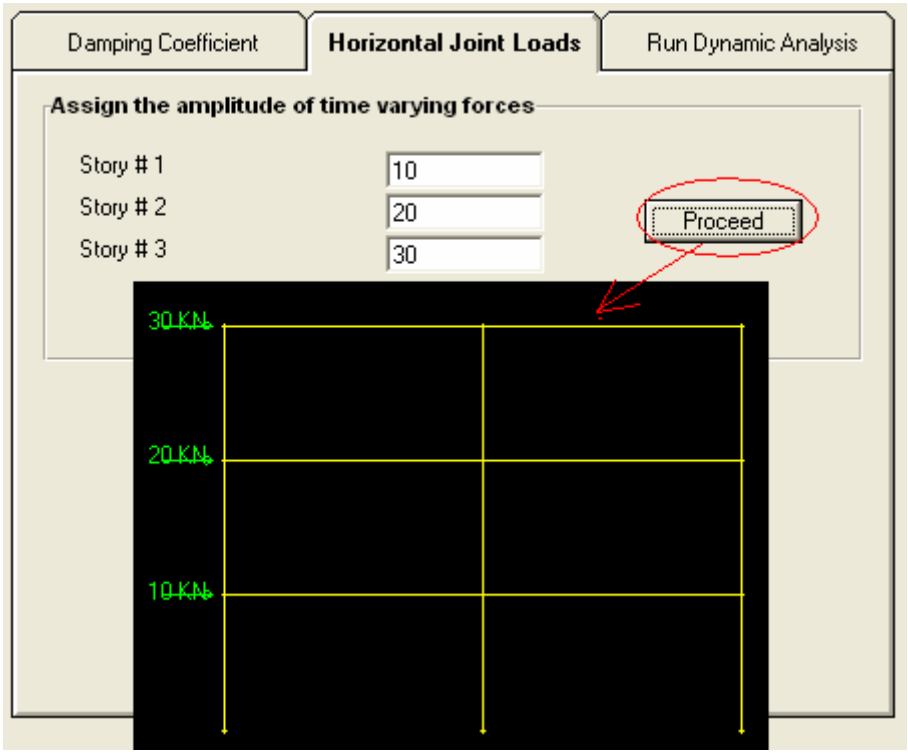


Figure 4.6 Horizontal Joint Loads

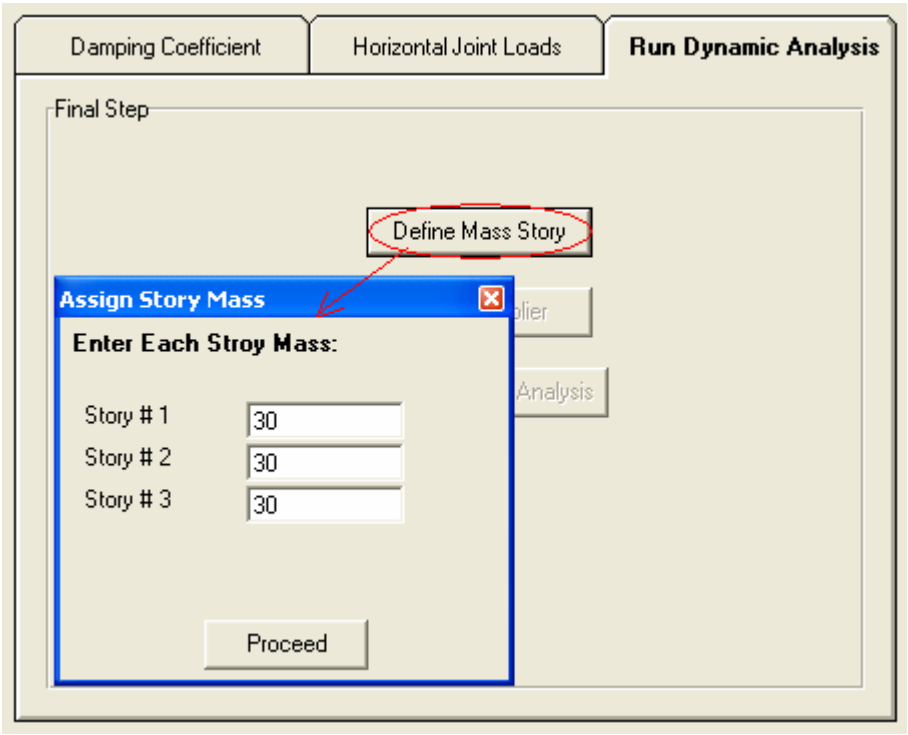


Figure 4.7 Define Mass Storey.

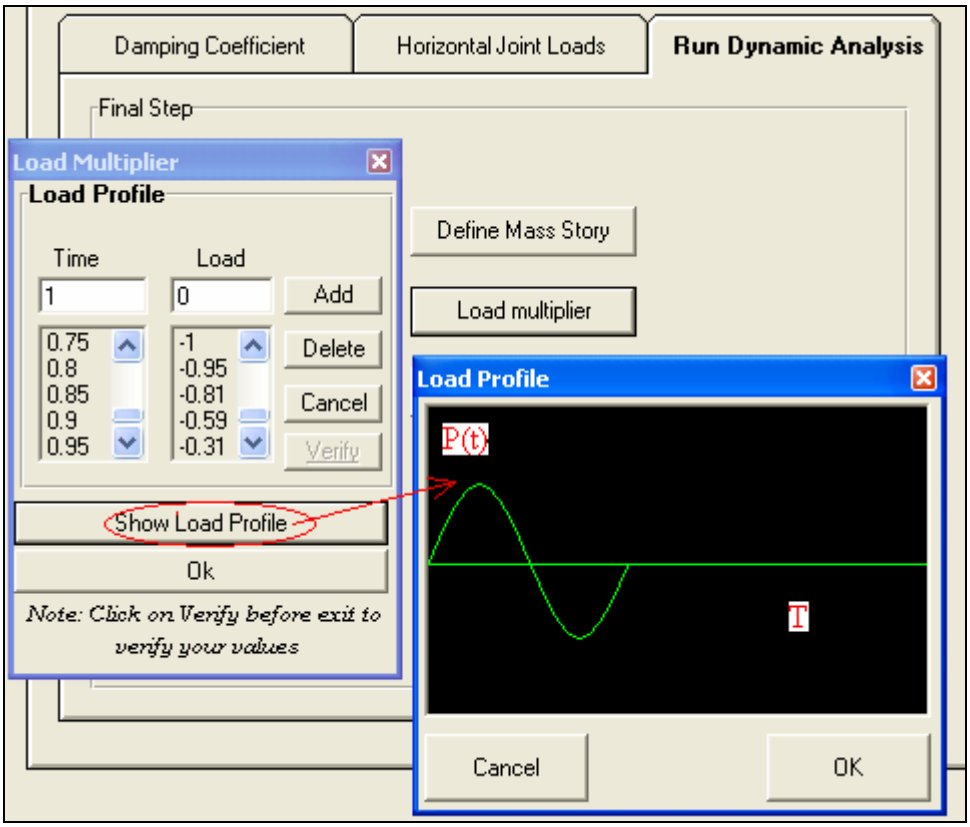


Figure 4.8 Load Multiplier.

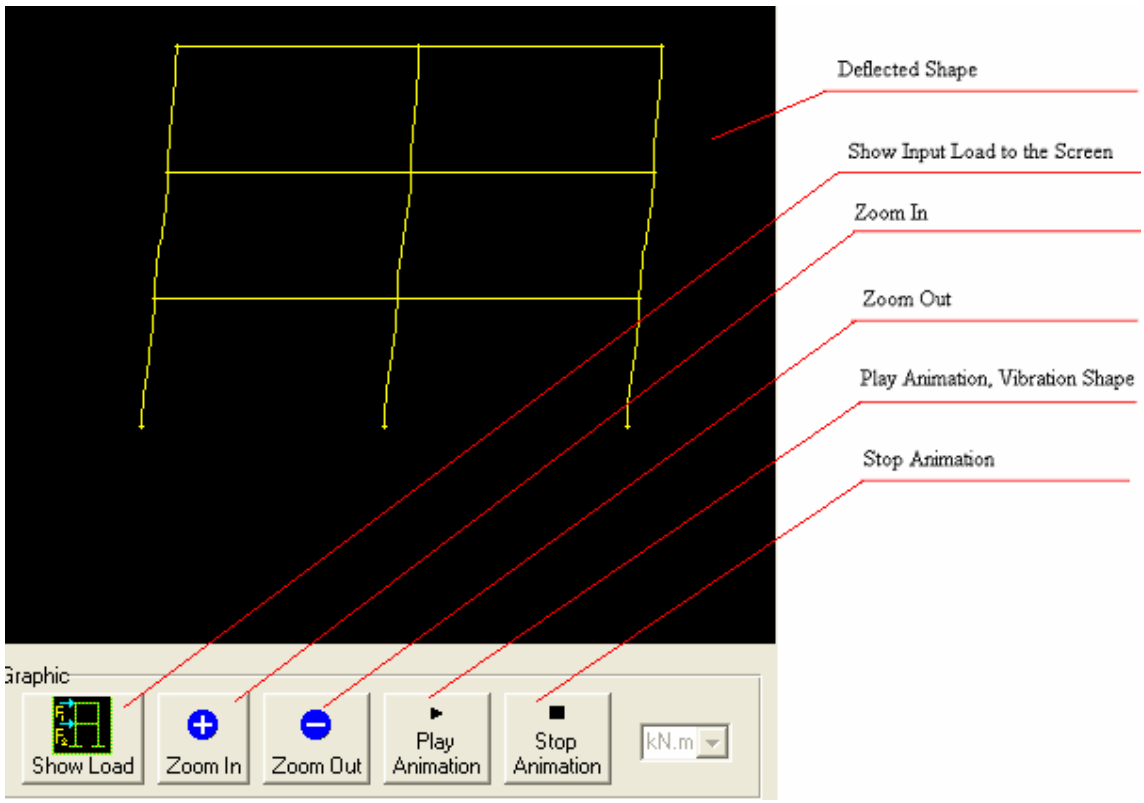


Figure 4.9 Deflected Shape.

The Button Tools of DAF contain RDS, GPro, Output and Report icons as shown in Figure 4.10. There icons display the results of DAF and are enabled only after the user runs the program by clicking the Run Dynamic Analysis command. RDS displays the Rayleigh Deflected Shape as shown in Figure 4.10, which shows the values of the primary selected shape and the improved Rayleigh shape. GPro displays the generalized properties and also the period of the structure as shown in Figure 4.11. The Output icon displays the displacements in millimeters and the time increment chosen with extra time increments as to locate the maximum values of displacements. This icon also contains a command button (Display) that enables the user to view the displacement table in a graphical format within a new form as shown in Figure 4.12. The Report icon displays the summary input and output data of DAF program in a report file as shown in Figure 4.13.

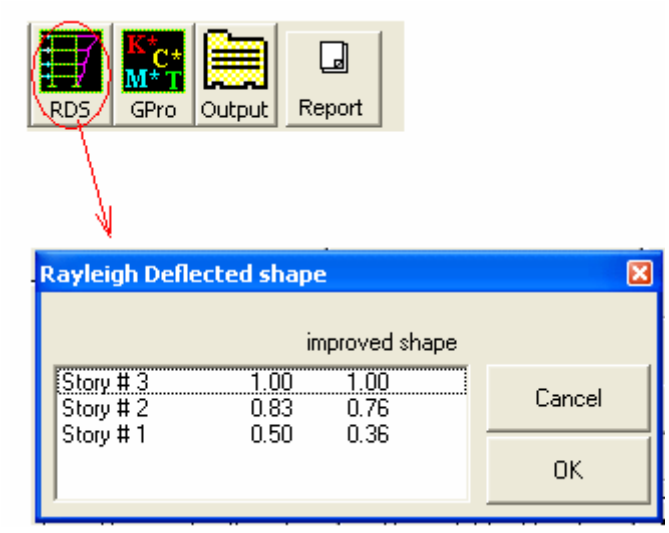


Figure 4.10 Rayleigh Deflected Shape.

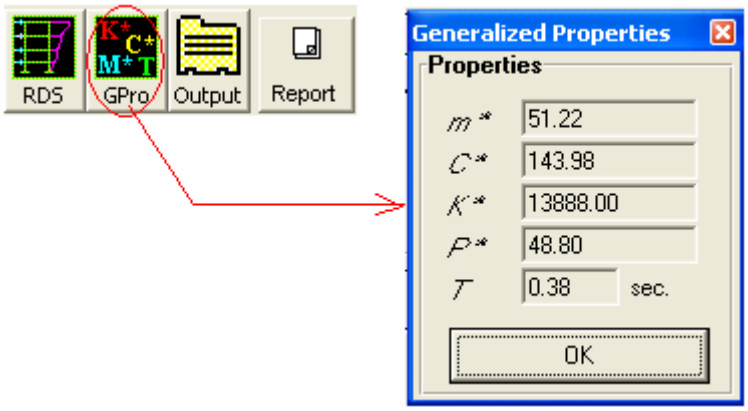


Figure 4.11 Generalized Properties.

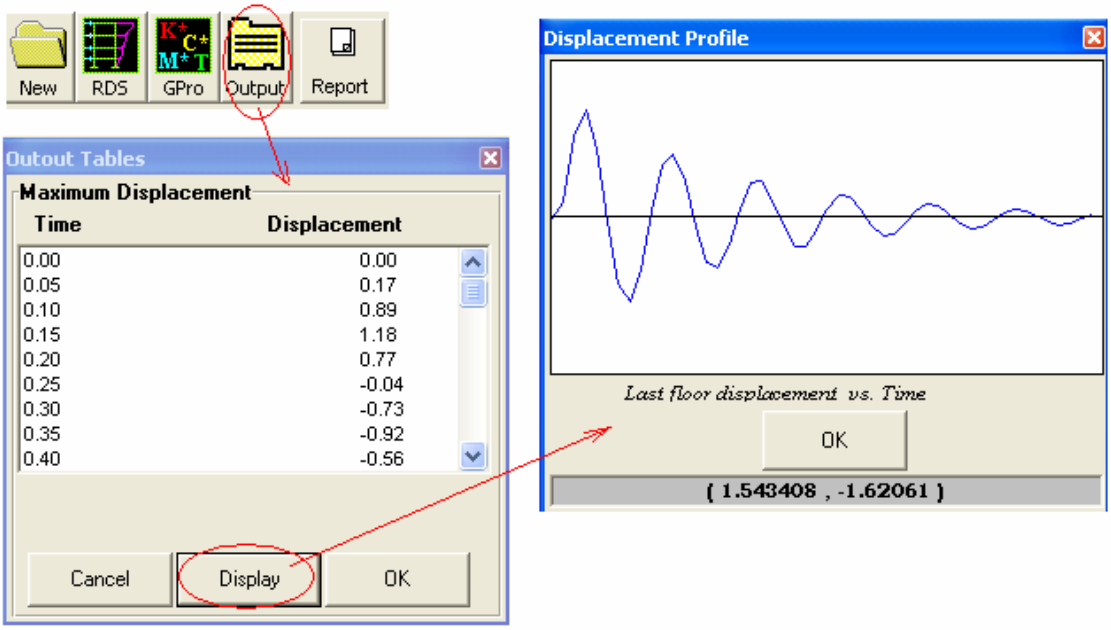


Figure 4.12 Maximum Displacement.

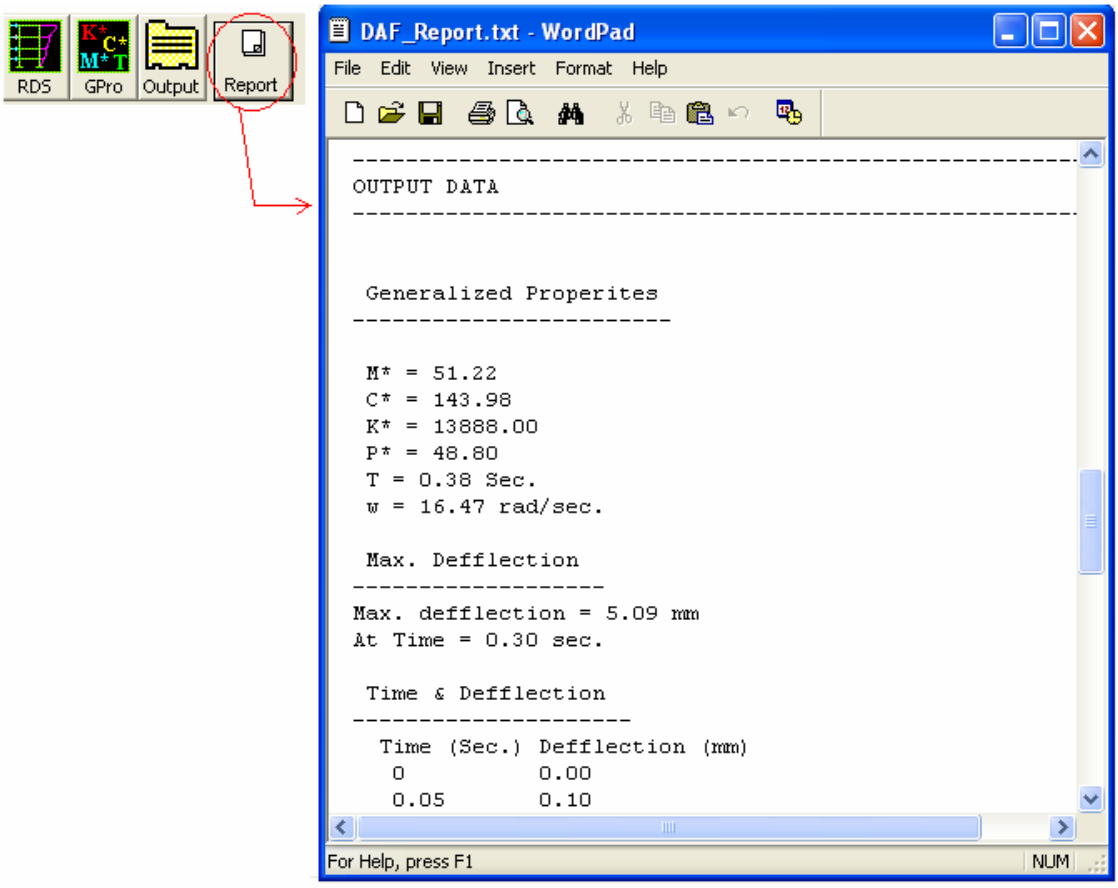


Figure 4.13 Output Data for DAF.

4.5 Verifications of DAF

Example statement and solution for comparing results between DAF and SAP2000:

Steel

E = 200,000,000 kN/m^2

Fixed base

All beam-column connections are rigid

Masses

Lumped mass at Girder is 30 ton for each girder

Loads

Horizontal loads in kN are shown in the Figure 4.14.

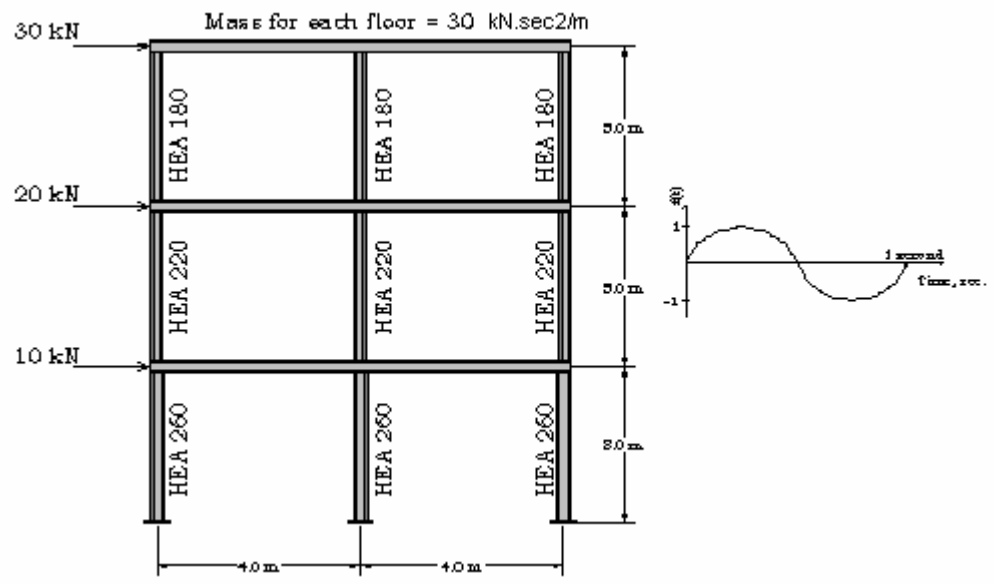


Figure 4.14 Example for DAF

The load profile used in the example is shown in Table 4.2.

Results of DAF program are shown in Figure 4.15; these can be summarized as follows:

T = 0.60 seconds

Max. Deflection = 22.0 mm at top storey level

Results of SAP2000 software are shown in Figure 4.16 and Figure 4.17. The listings of these output file are given in appendix B, the results can be summarized as follows:

T = 0.61 seconds

Max. Deflection = 23.7 mm at top storey level.

Table 4.2 Load Profile.

Time Increment	Load Multiplier
0	0
0.05	0.31
0.1	0.59
0.15	0.81
0.2	0.95
0.25	1
0.3	0.95
0.35	0.81
0.4	0.59
0.45	0.31
0.5	0
0.55	-0.31
0.6	-0.59
0.65	-0.81
0.7	-0.95
0.75	-1
0.8	-0.95
0.85	-0.81
0.9	-0.59
0.95	-0.31
1	0

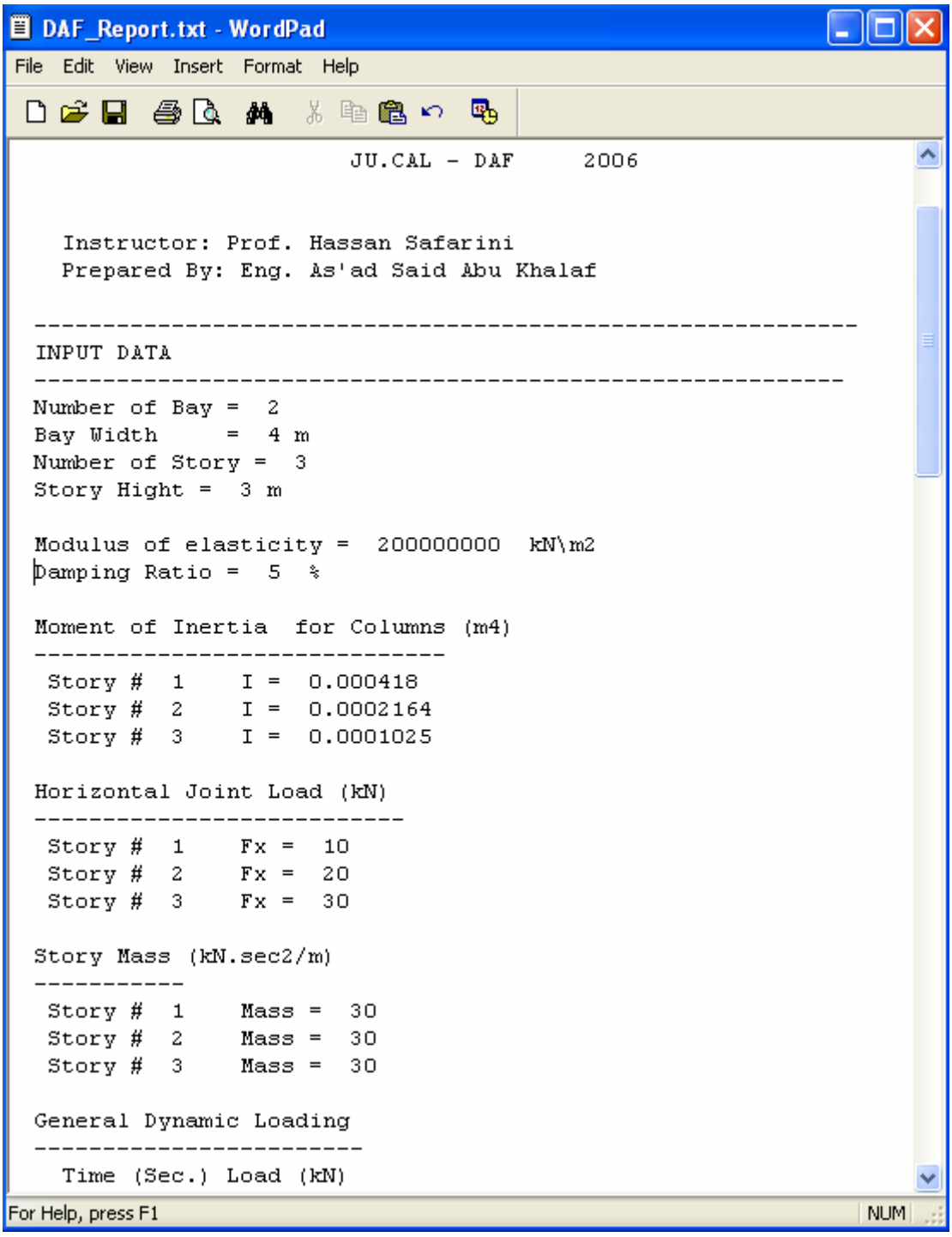


Figure 4.15 Example Report.

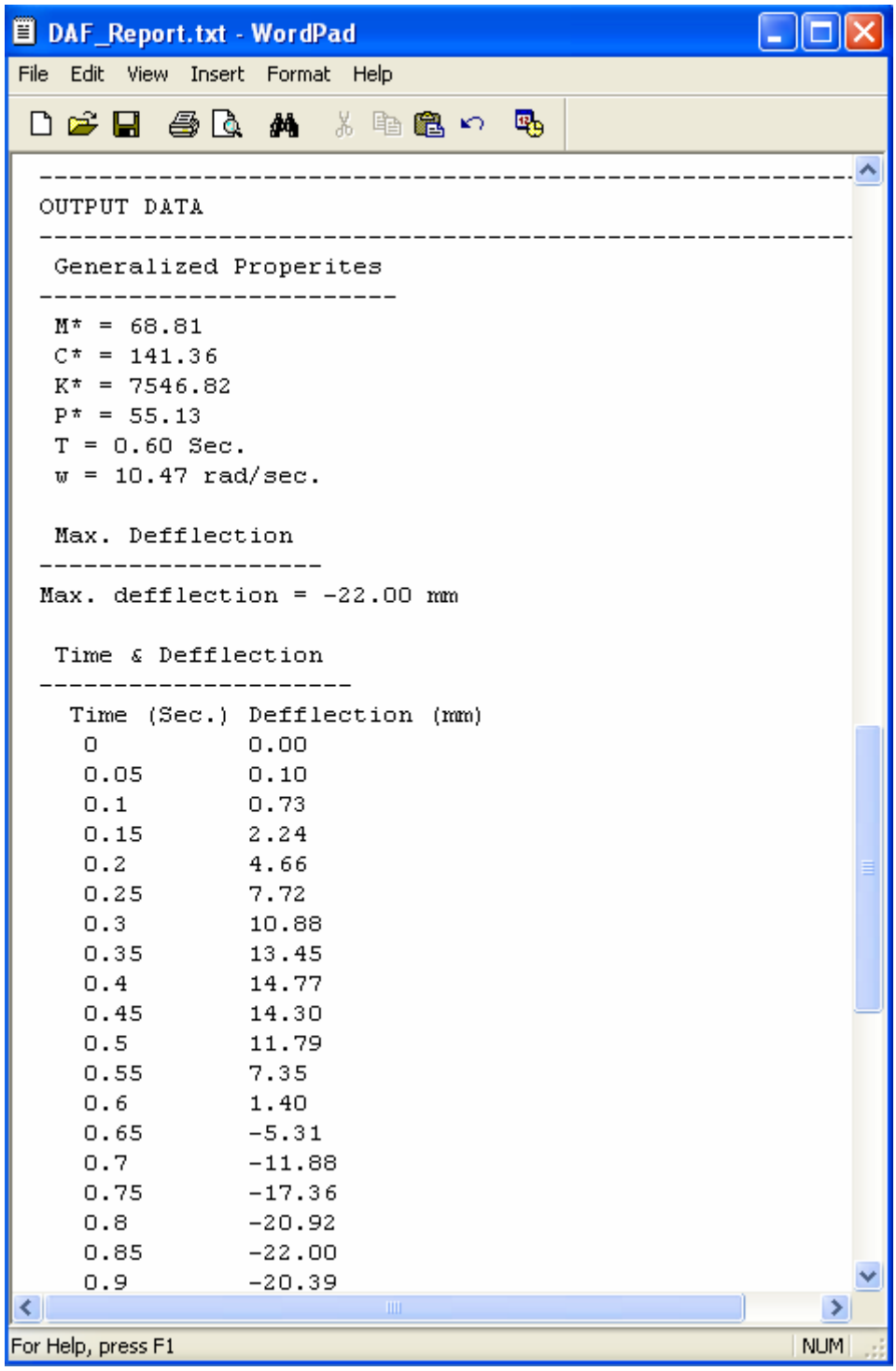


Figure 4.15-Cont. Example Report.

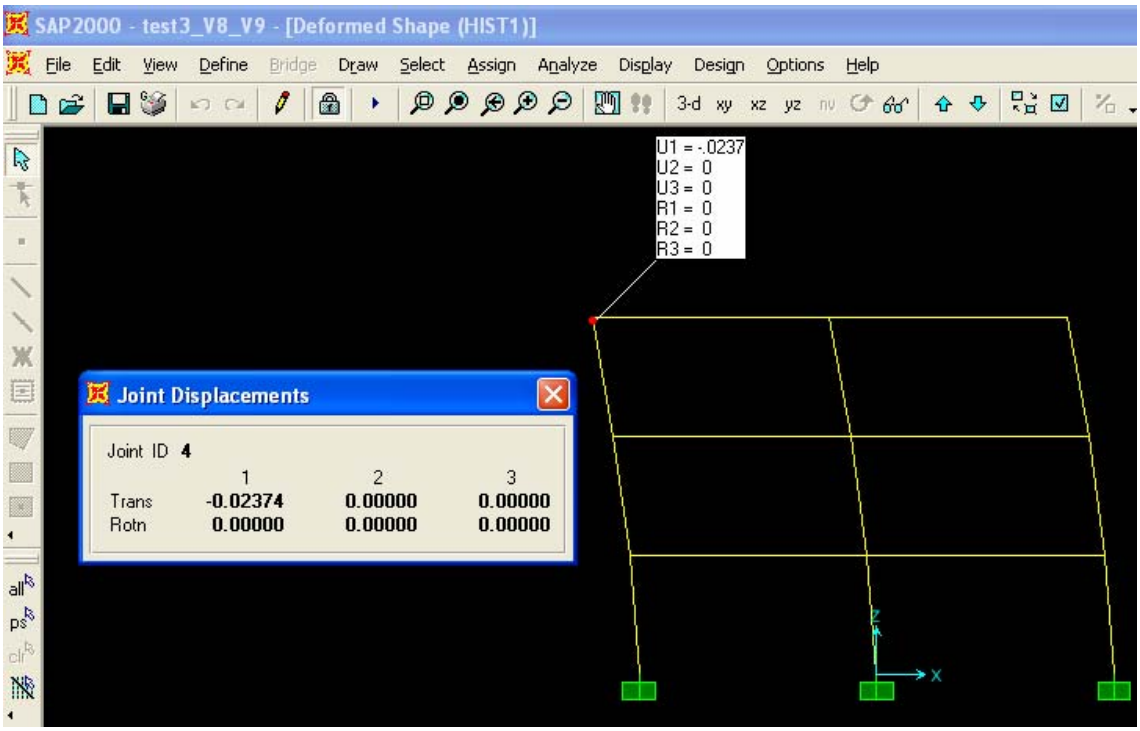


Figure 4.16 Example Sap2000- Displacement.

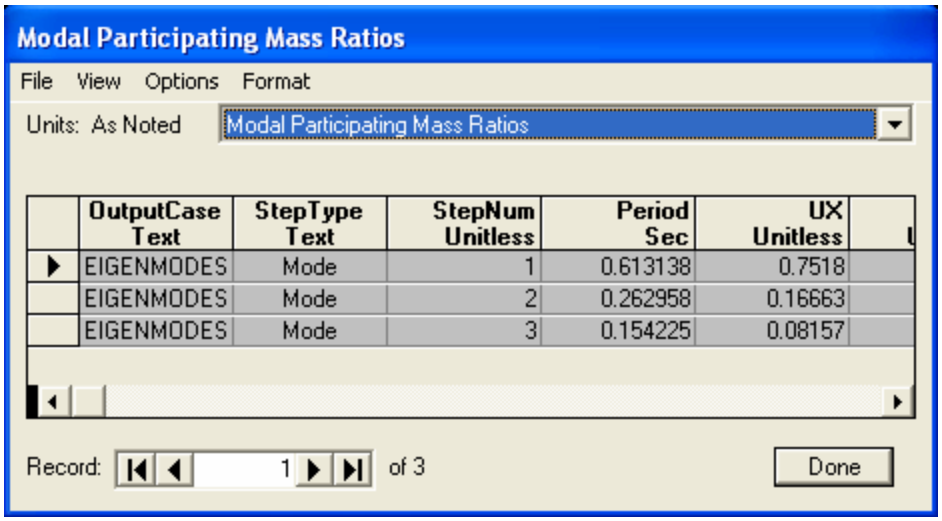


Figure 4.17 Example Sap2000- Period.

Design Response Spectra (DRS)

5.1. Introduction

In this chapter an overview of the generation design response spectra is presented. The graphical user interface for JU.DRS includes the computation of the response spectrum and the software can be readily used for analyzing results and presenting them in the form of tables and graphs. Finally, verification of some known results from references is presented.

5.2 Overview of Design Response Spectra

The most common form of data bank used in the design of structures to resist earthquakes is response spectra. A response spectra is a curve that shows how the maximum displacement response, velocity or acceleration of oscillators with the same damping ratio, but with different natural frequencies, respond to a specified earthquake. The numerical method such as Central Difference method, Newmarks methods, and Duhamel's integral method may be used to calculate the maximum displacement, velocity and acceleration of a single oscillator for a given earthquake record such as that shown in Figure 5.1. It represents the North-South component of the ground motion recorded at a site in EI Centro California during the Imperial Valley, California, Earthquake of May 18, 1940 (Chopra, 1995). The equation of motion of a linear SDOF system shown in Figure 5.2, relative to the support, when subjected to ground acceleration $\overset{**}{X}(t)$, can be written as:

$$M\overset{**}{X}+C\overset{*}{X}+KX=M\overset{**}{X}_g(t) \qquad (5.1)$$

Where: M is the mass of the system

C its damping coefficient

K its stiffness

X the relative displacement of the mass with respect to the ground.

$\overset{**}{X}_g(t)$ the ground acceleration.

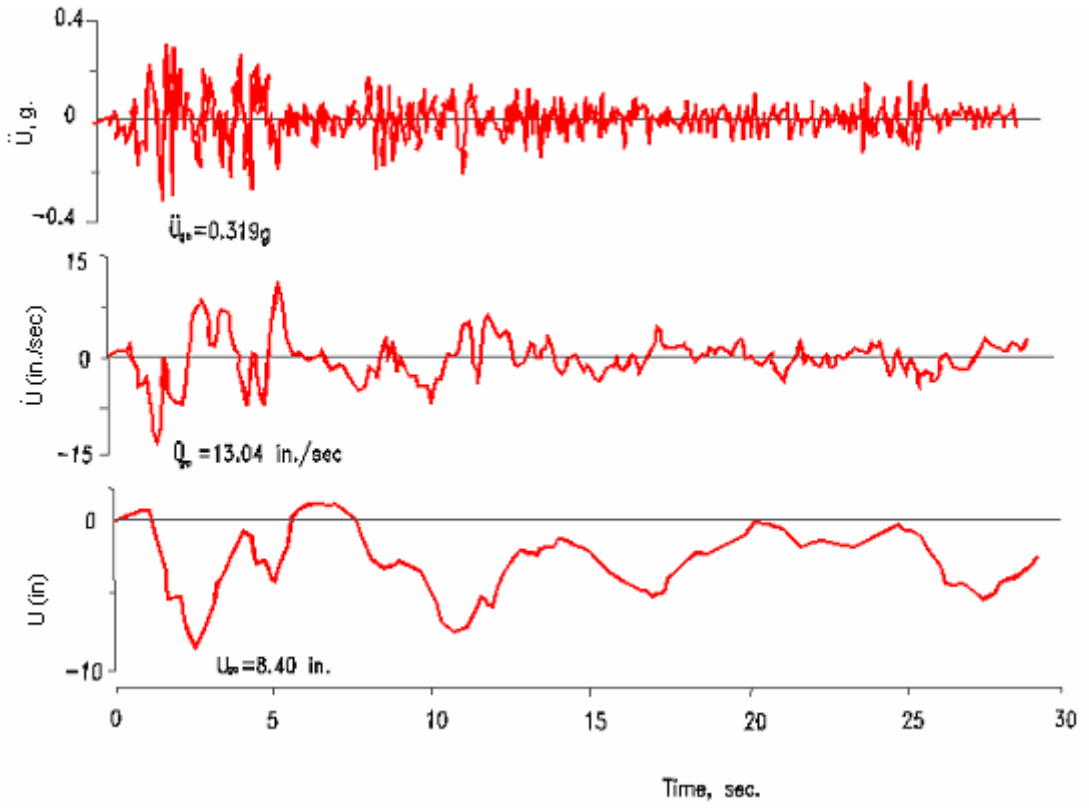


Figure 5.1 North-South component of the horizontal ground acceleration, recorded at Imperial Valley, EI Centro, California, 1940. The ground velocity and ground displacements were computed by integrating the ground acceleration.

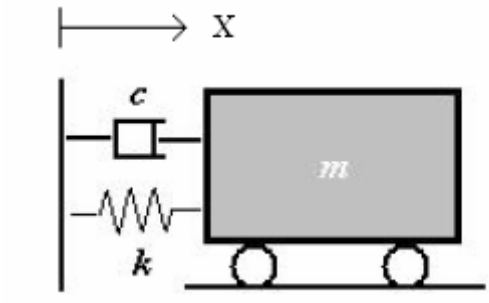


Figure 5.2 A single degree of freedom system.

After dividing both sides of this equation by the mass, it follows that

$$\ddot{X} + 2\xi\omega_n \dot{X} + \omega^2{}_n X = \ddot{X}_g(t) \qquad (5.2)$$

Where $\omega_n^2 = K / M$ and $2\xi\omega_n = C / M$.

Thus by calculating the maximum response of oscillators with different frequencies, but with the same damping, it is possible to construct a response spectrum in the frequency domain for oscillators with the same damping ratio. By repeating this process for oscillators with different damping ratios, it is possible to construct a number of response spectra for the same record (Buchholdt, 1997).

The numerical solution of equation 5.2 provides the results for the relative displacement, X(t). Using dynamic analysis of structures to evaluate the deformation response history X(t) for any damping ratio ξ and natural vibration period T_n, the maximum absolute value of the response can be determined, and this response is called the spectral displacement, S_d, and it can be written as:

$$S_d = |X(t)|_{max} \qquad (5.3)$$

The presence of the absolute value means that all values of S_d are positive even if the maximum value is a maximum negative value. In other word, max[X(t)] represents the absolute maximum value.

In a similar manner, the maximum values for each relative velocity, $\dot{X}(t)$, and acceleration, $\ddot{X}(t)$, can be determined. These response quantities are called the spectral velocity, S_v, and spectral acceleration, S_a, respectively, and they can be written as:

$$S_v = |\dot{X}(t)|_{max} \qquad (5.4)$$

$$S_a = |\ddot{X}(t)|_{max} \qquad (5.5)$$

The spectral displacement, spectral velocity and spectral acceleration can be calculated using computer software for a given earthquake ground motion time history for different values of T_n and ξ.

The maximum velocity response can be approximated by multiplying the spectral displacement by the circular frequency. This response parameter is defined as the spectral pseudo velocity and is expressed as:

$$S_{pv} = \omega S_d \qquad (5.6)$$

The maximum total acceleration can be approximated as the spectral displacement multiplied by the square of the circular frequency. This product is defined as the spectral pseudo acceleration and is expressed as:

$$S_{pa} = \omega^2 S_d \qquad (5.7)$$

Figure 5.3 shows plots of the corresponding spectral displacement, spectral velocity and spectral acceleration versus T_n For North-South component of 1940 EI Centro earthquake (Hart, 2000).

The three response parameters for the EI Centro motion are shown in Figure 5.4. For a SDOF system having a given period and damping, the three spectral response parameters for this earthquake can be read directly from the graph as shown in Figure 5.5 (Naeim, 1989).

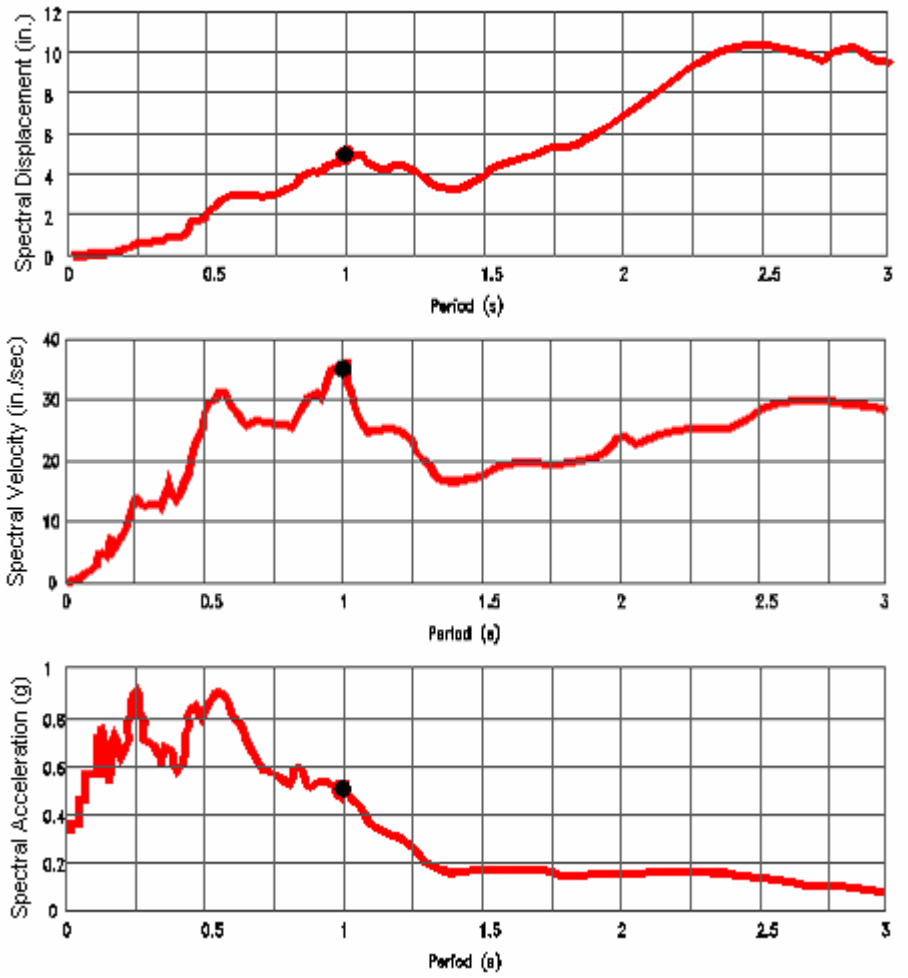


Figure 5.3 Acceleration, Velocity and Displacement Spectra for North-South component of 1940 EI Centro earthquake (5% Damping).

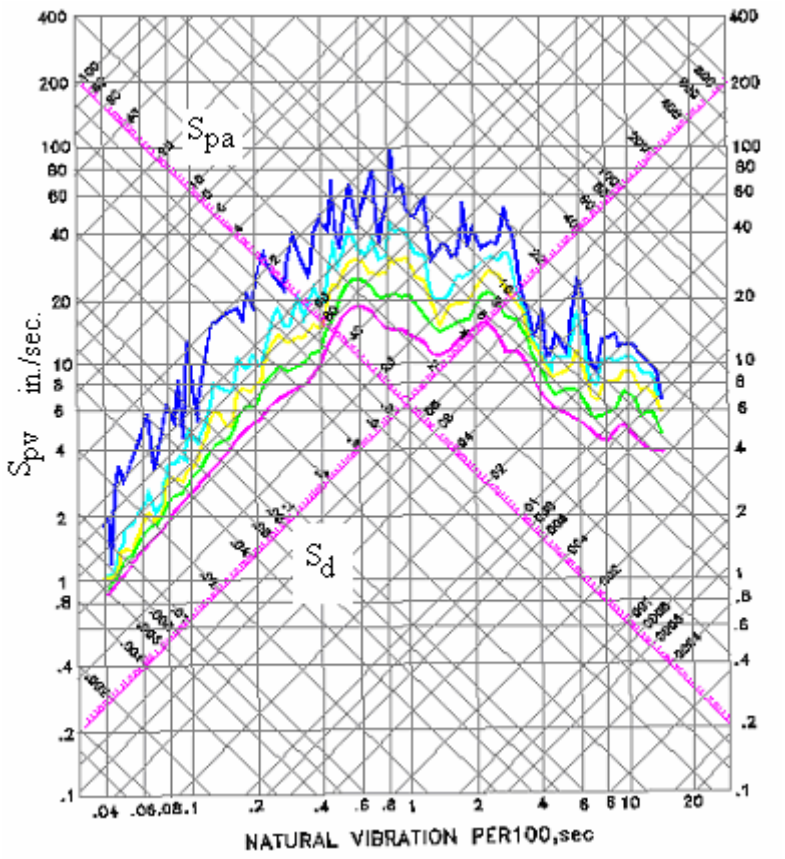


Figure 5.4 Typical tripartite response spectra curves (Naeim, 1989).

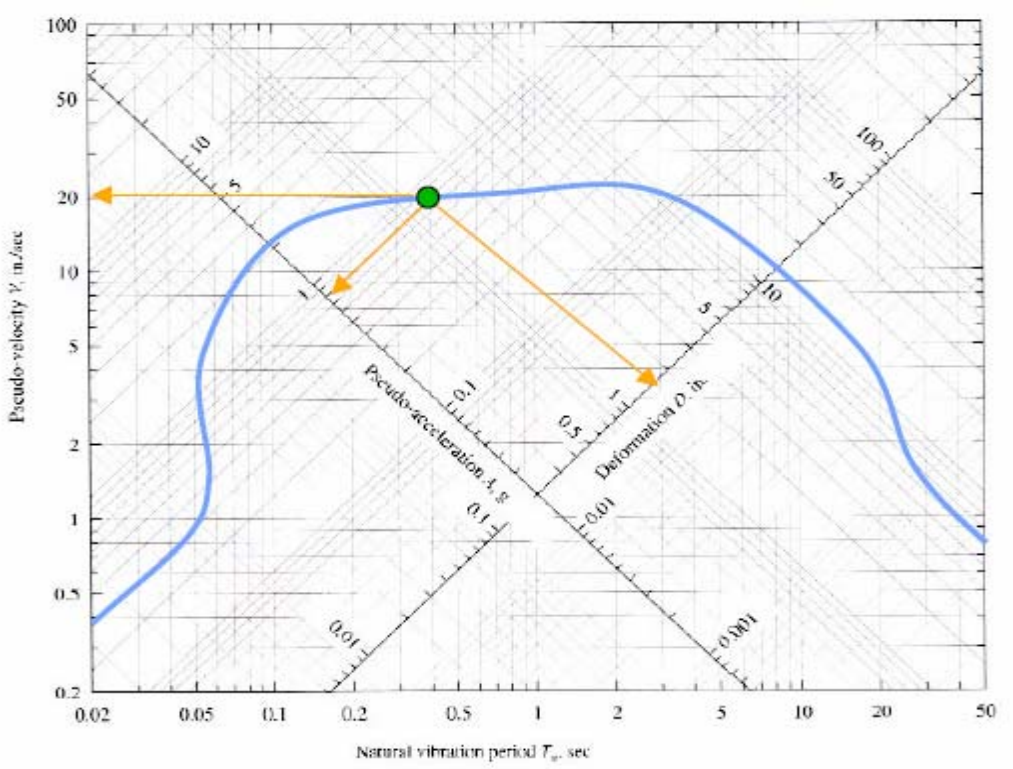


Figure 5.5 Tripartite Response Spectra.

5.3 DRS Interface

The graphical user interface for DRS is shown in Figure 5.6. Several important features of the interface are shown. Those features include main window, toolbar, text box, buttons and combo box. Each of these items is described in this chapter.

Toolbars are comprised of buttons as shown in Figure 5.7. Toolbar buttons provide one-click access to commonly used commands, particularly reading acceleration data from a text file, run analysis, save results and display summarized output results. When we start DRS by selecting Analysis menu > Design Response Spectra or by clicking mouse on the tab menu > Design Response Spectra, all the buttons in the toolbars are disabled except Read Acceleration button. The user must click on this button to read the acceleration time history data as a text file (.txt), then the program is active to use and the time history acceleration data is shown as a graphical form as shown in Figure 5.8. Consequently, the Run buttons in the toolbars are enabled.

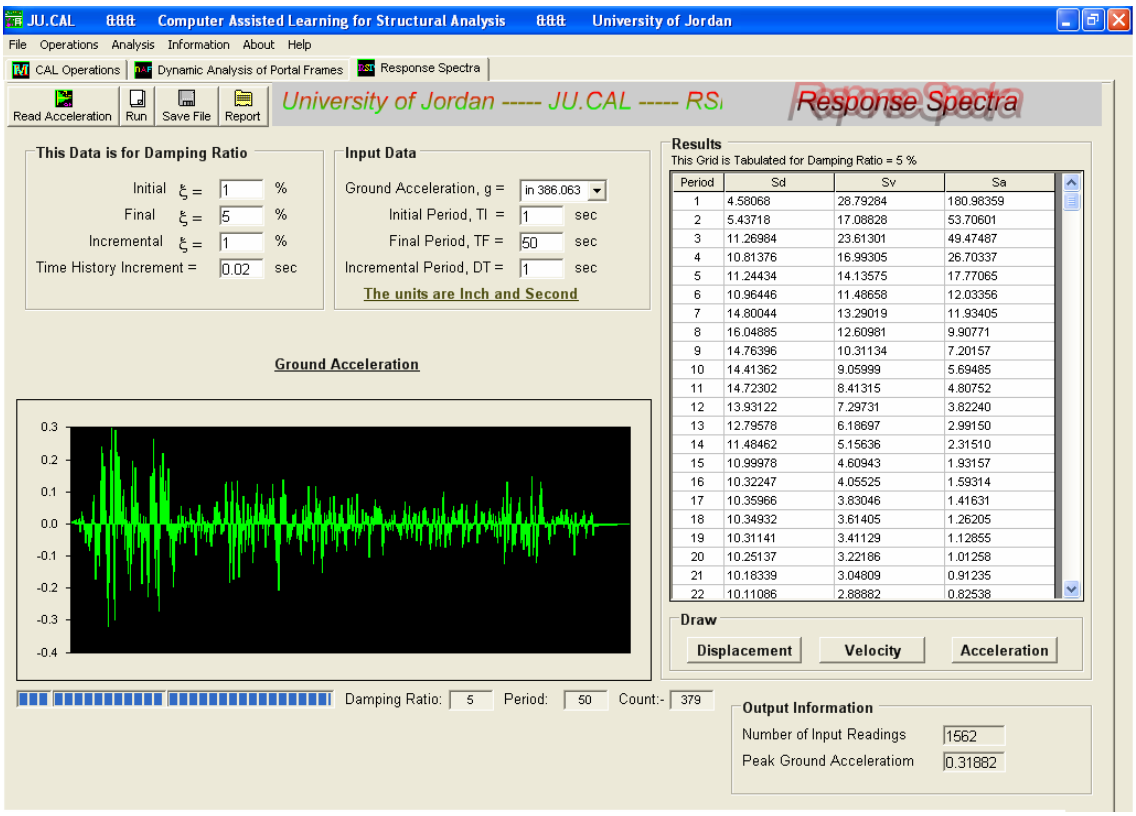


Figure 5.6 DRS Interface.

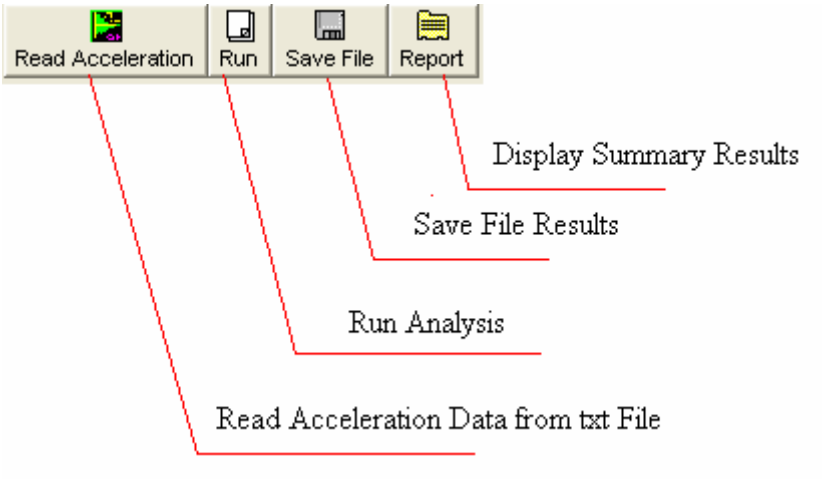


Figure 5.7 Toolbars of DRS.

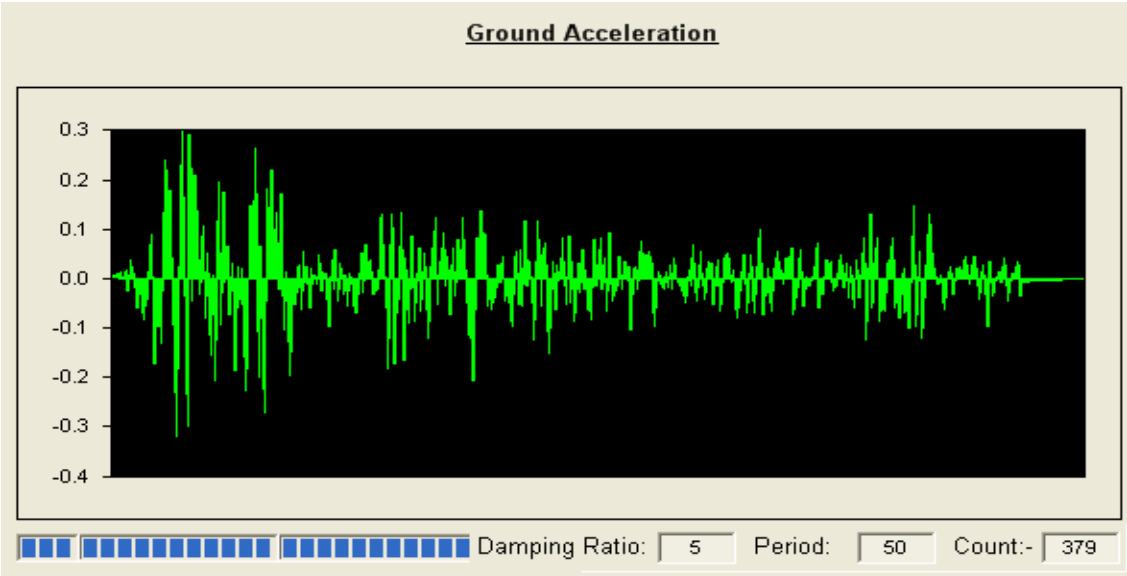


Figure 5.8 Time History Acceleration for DSR

After reading the acceleration time history from a file, the user must assign some properties needed to run the program like damping ratio, period, time history increment and the ground acceleration as shown in Figure 5.9.

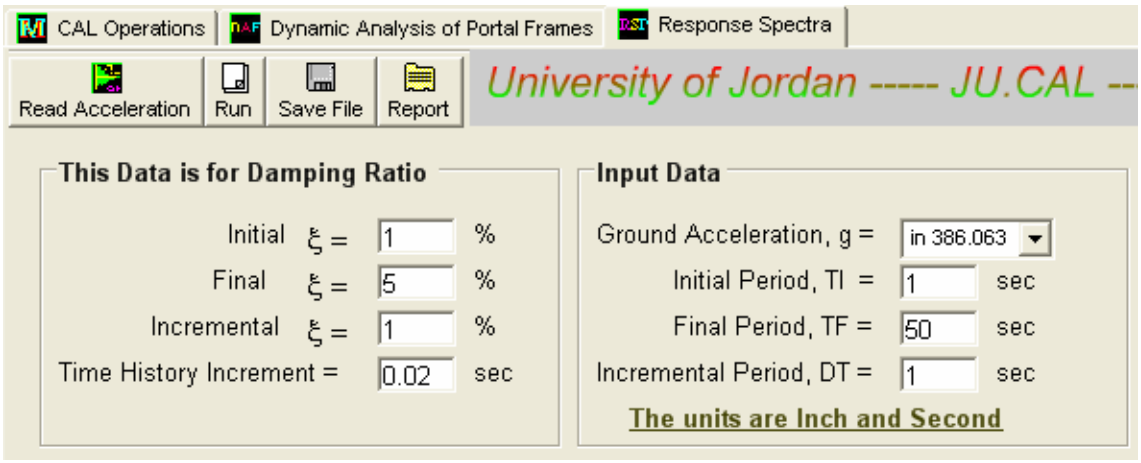


Figure 5.9 Input Data for DRS

Using Run button form toolbars to run analysis, the progress bar in the bottom left corner of the program indicates the looping of operations for all data, this may take some time depending on the size of the input data, number of damping ratios considered and period increment, The results are then summarized in a table form as shown in Figure 5.10 and Figure 5.11. This result is tabulated for a 5% damping ratio. For each period increment, the spectral displacement, spectral pseudo velocity and spectral pseudo acceleration are obtained. Using draw command to plot the S_A-T, S_v-T and S_D-T separately as shown in Figure 5.12, Figure 5.13 and Figure 5.14. DRS summarized input and output data as a report file, this file can be viewed by clicking the report button in the toolbar as shown in Figure 5.15.

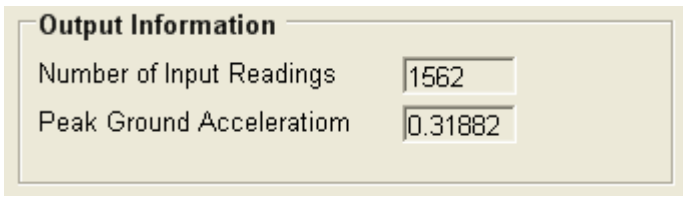
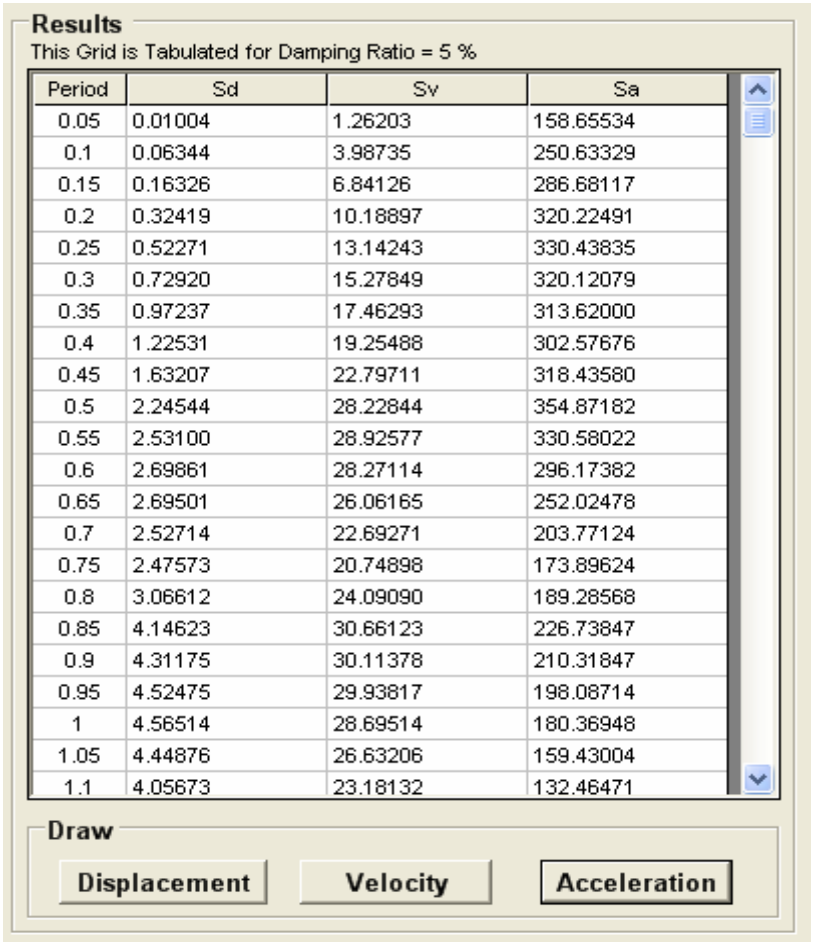


Figure 5.10 Output Information.



Period	Sd	Sv	Sa
0.05	0.01004	1.26203	158.65534
0.1	0.06344	3.98735	250.63329
0.15	0.16326	6.84126	286.68117
0.2	0.32419	10.18897	320.22491
0.25	0.52271	13.14243	330.43835
0.3	0.72920	15.27849	320.12079
0.35	0.97237	17.46293	313.62000
0.4	1.22531	19.25488	302.57676
0.45	1.63207	22.79711	318.43580
0.5	2.24544	28.22844	354.87182
0.55	2.53100	28.92577	330.58022
0.6	2.69861	28.27114	296.17382
0.65	2.69501	26.06165	252.02478
0.7	2.52714	22.69271	203.77124
0.75	2.47573	20.74898	173.89624
0.8	3.06612	24.09090	189.28568
0.85	4.14623	30.66123	226.73847
0.9	4.31175	30.11378	210.31847
0.95	4.52475	29.93817	198.08714
1	4.56514	28.69514	180.36948
1.05	4.44876	26.63206	159.43004
1.1	4.05673	23.18132	132.46471

Figure 5.11 Results for DRS.

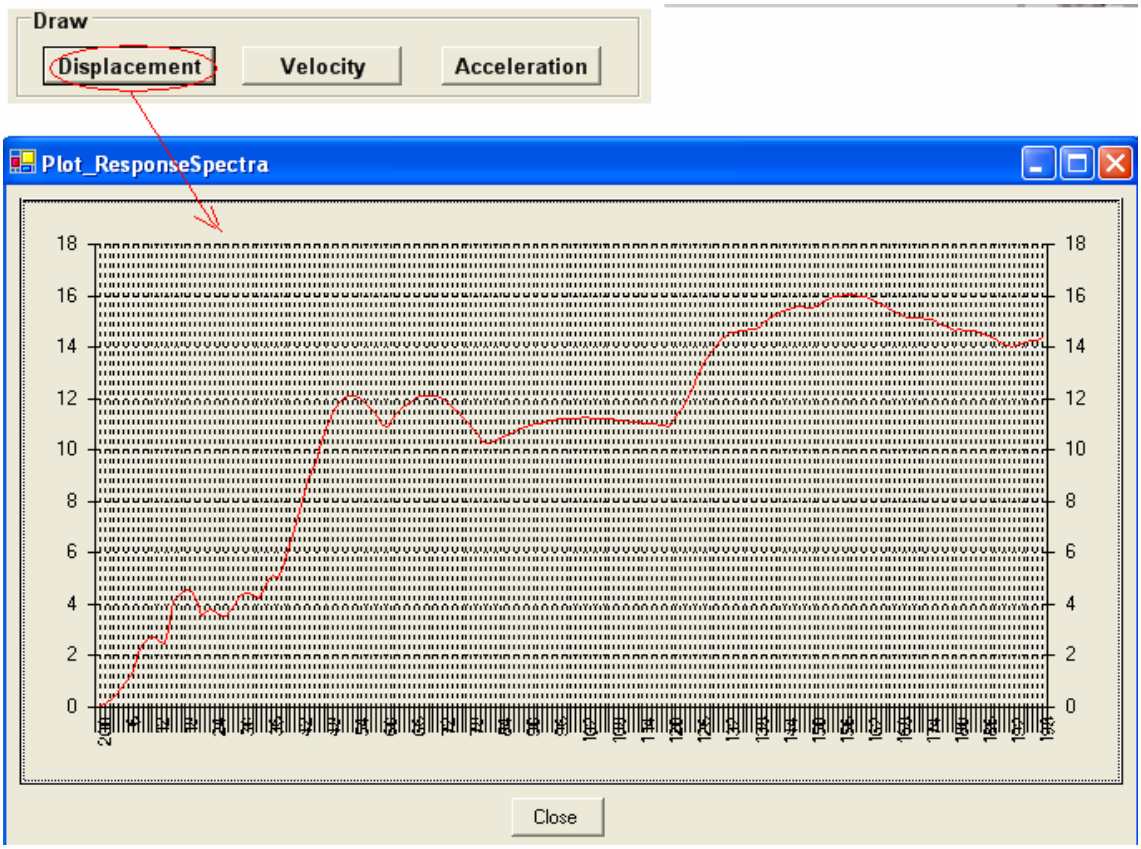


Figure 5.12 S_D-T plot.

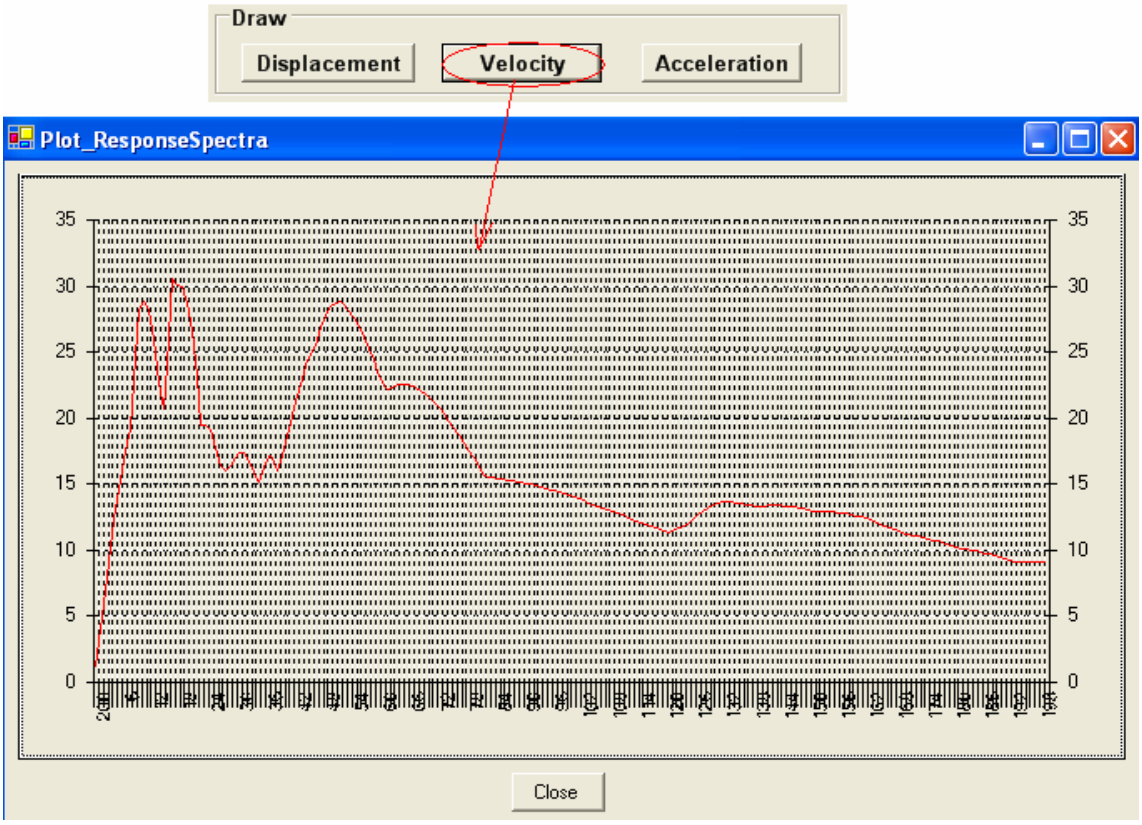


Figure 5.13 S_v-T plot.

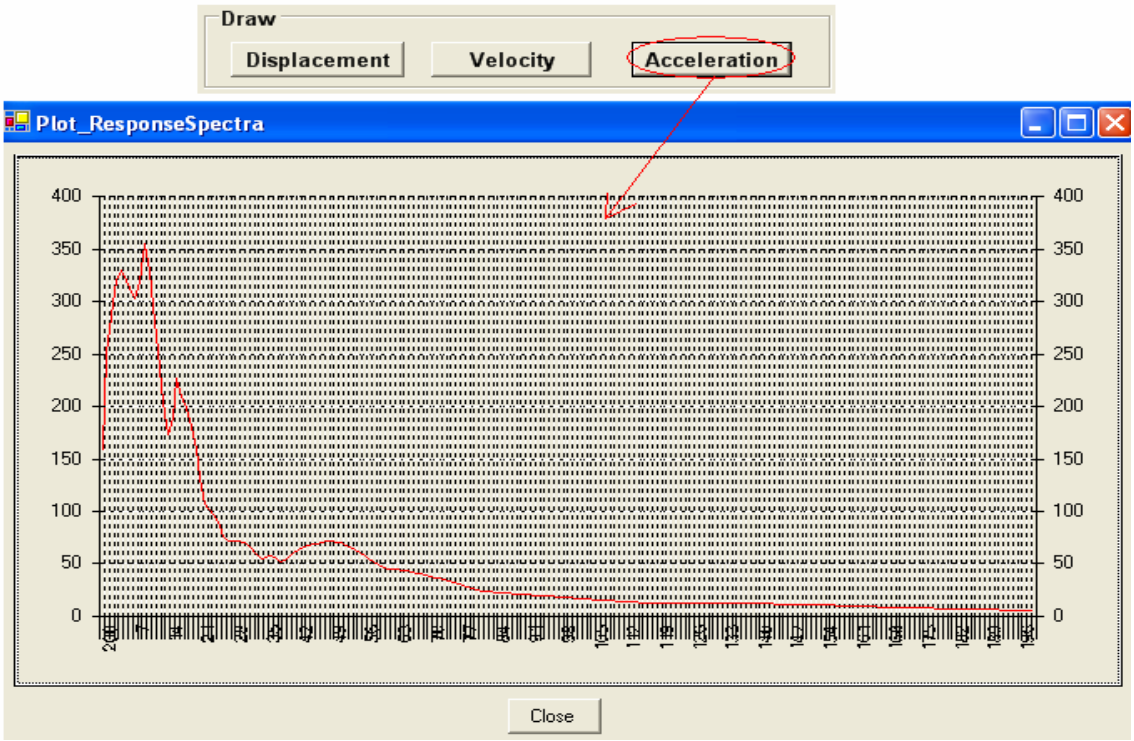


Figure 5.14 S_a-T plot.

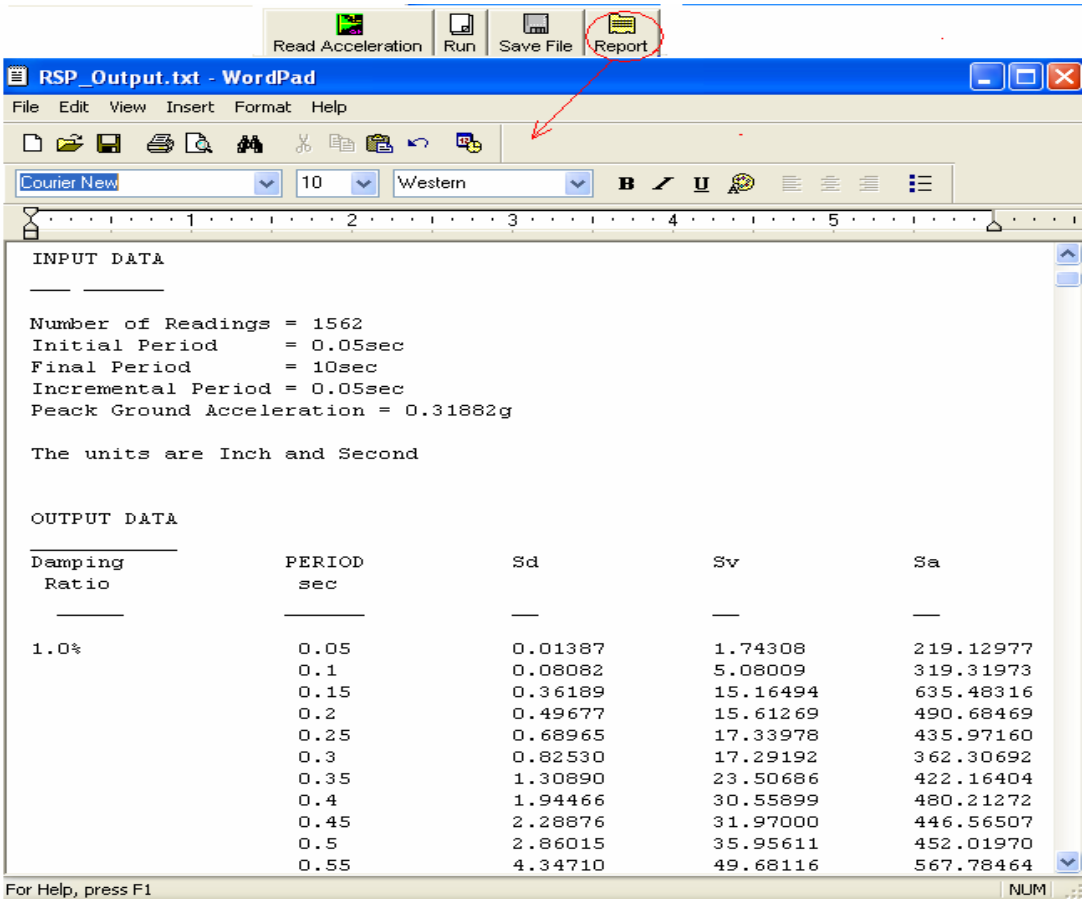


Figure 5.15 Report File for DRS.

5.4 Verifications of DRS

To verify the reliability of the procedure used in DRS, the north-south component of the EI Centro ground motion recorded at Imperial Valley, EI Centro, California 1940 was analyzed.

There are several versions of the EI Centro ground motion. The variations among them arise from differences in how the original analog trace of the acceleration versus time was digitized into numerical data, and the procedure chosen to introduce the missing baseline in the record (Kablawi, 1997).

The version used in this verification is obtained from Chopra (1995). The data is comprised of 1562 points at equal time increments of 0.02 sec. The numerical values of ground accelerations are in units of g, the acceleration due to gravity. The acceleration time history of the EI Centro used here is shown in Figure 5.16.

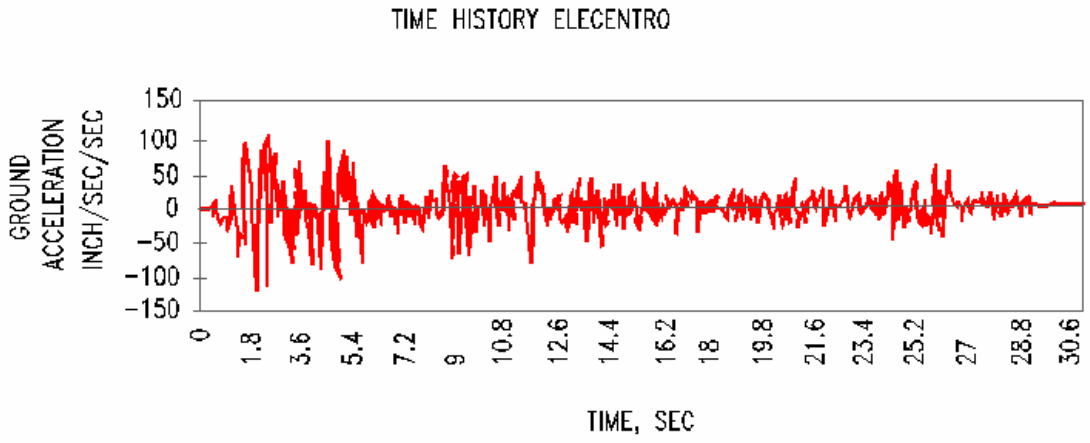


Figure 5.16 Time History Acceleration of the EI Centro.

The acceleration time history for the EI Centro ground motion was read by the computer program JU.DRS, which then calculated the response values for damping ratios of 2,3,4,5% over the period range of 0.05 to 10 sec. The displacement, velocity and

acceleration response to EI Centro ground motion were then tabulated as shown in Figure 5.17.

The response spectrum for the EI Centro ground motion has been computed and studied by many researchers. Typical response spectrum curves are shown in Figure 5.18 after Chopra (1995). Comparisons of the results obtained by JU.DRS and Chopra are shown in Table 5.1.

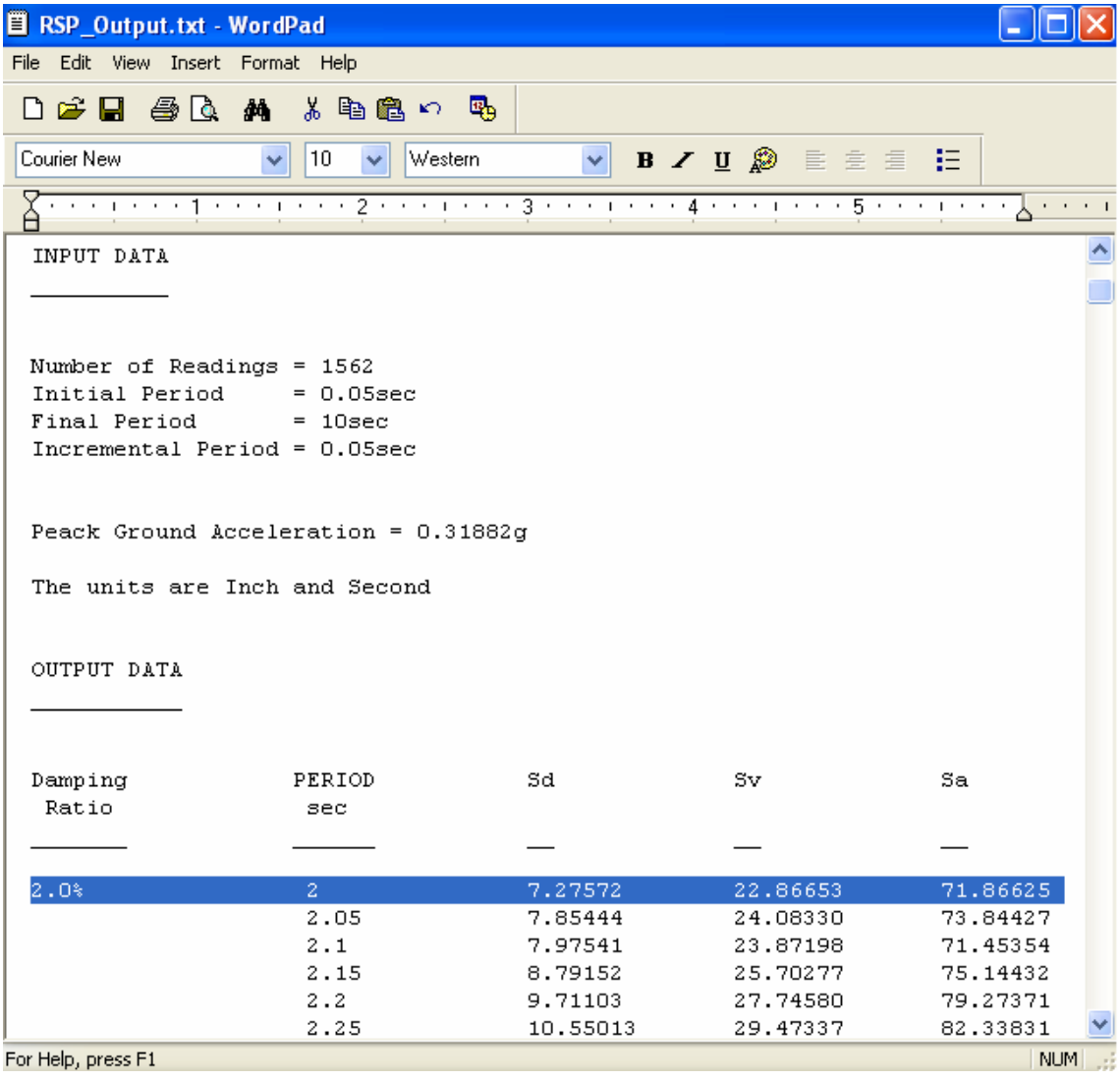


Figure 5.17 Response spectrum for EI Centro ground motion $\xi = 2\%$ by JU.DRS.

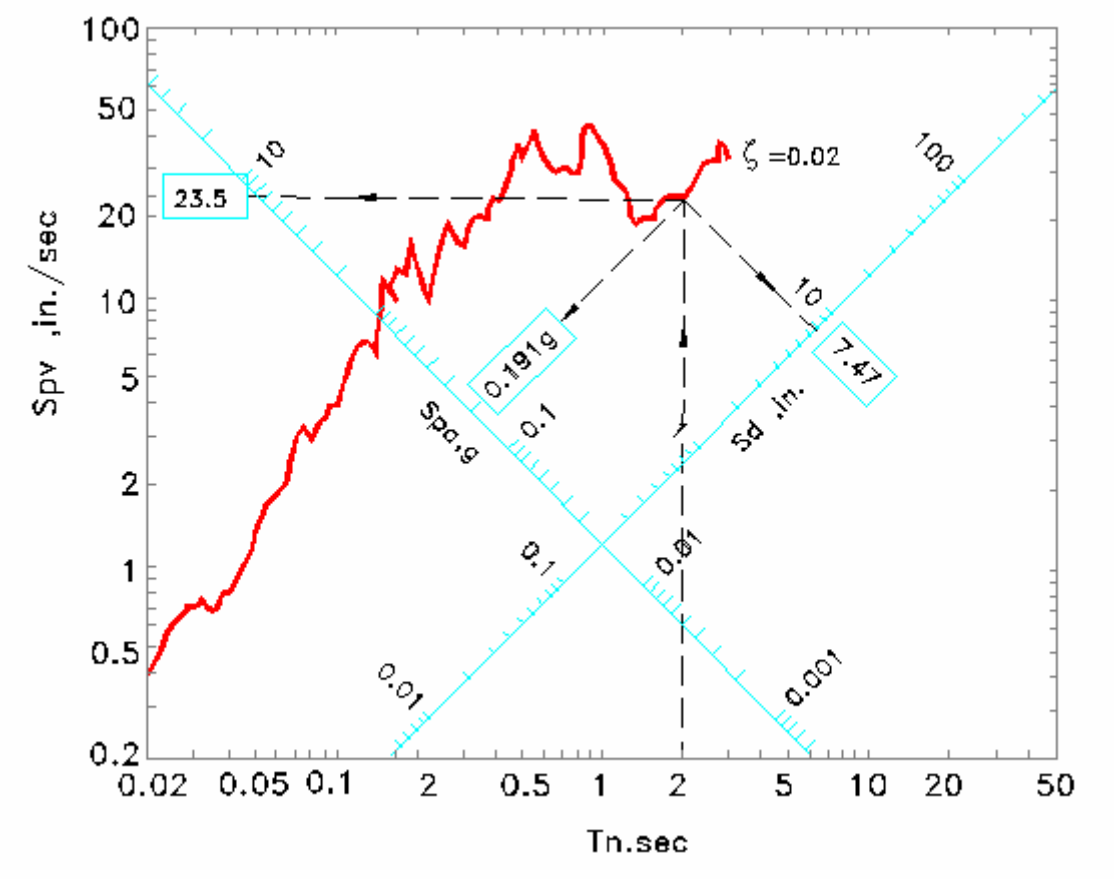


Figure 5.18 Response spectrum for EI Centro ground motion ξ = 2% (Chopra, 1995).

Table 5.1 Comparison Results Between Chopra (1995) and JU.DRS.

Response Spectrum for 2% damping ratio and T_n = 2 Sec.			
Results by	**S_d (in.)**	**S_v (in./sec)**	**S_a (in/sec^2)**
Chopra	7.47	23.50	73.83
JU.DRS	7.28	22.87	71.87

Where S_a = 73.83 in/sec^2 = 0.191 g. (by Chopra).

S_a = 71.87 in/sec^2 = 0.187 g. (by JU.DRS).

Conclusions and Recommendations

The objective of this thesis was to develop a series of tools to assist engineering students in the understanding of fundamental structural engineering concepts. The topics covered include matrix operations, direct stiffness operations, dynamics operations, dynamic analysis of multistory frames and generating response spectra. This chapter presents the conclusions and recommendations of this study.

6.1 Conclusions

1. Using the Object Oriented Programming and Visual Basic .Net language to write JU.CAL program had several benefits. One benefit was the use of real world objects such as joints, supports and members. Another benefit was that a number of classes representing structural objects were reused in creating the different application. Having these classes made up of real world objects made the programs easier to understand and debug. Also, the use of Visual Basic .Net existing libraries reduced the amount of time and effort needed to create the graphical user interface.

2. The inheritance feature in OOP encourages extension and reuse of the code. Software can be easily expanded and upgraded since modifications can be conveniently incorporated by altering only the relevant classes. The programmer can reuse a class created by another programmer by deriving classes with additional features and capabilities required for the specific application.

3. The computer program JU.CAL is designed to be used by students in structural analysis courses for the purpose of understanding modern concepts of structural analysis incorporating matrix operations, direct stiffness methods and dynamics operations.

4. Differences between SAP2000 results and JU.DAF results are in the range of 1% to 5%; the SAP2000 could be more accurate because it uses exact solution methods incorporating MDOF stiffness and several types of mass matrices in evaluating different modal shapes. However, in the case of this software, approximate methods are used such as assuming deflected shapes, shear building model and lumped masses. JU.DAF gives a very good approximation as can be seen from the comparison with SAP2000 results.

5. The underlying purpose behind the work incorporated in this study is to create an educational tool that can form a basis for enhancing the education of structural engineering in order to better prepare young graduates for the challenges and complexities of structural engineering of the 21st century. Although a good basis in mathematics, physics and classical structural methods remain paramount in the understanding of first principals and fundamental theories, the audited migration towards digital processing that is evident in the work place, at this time dictates that future engineers be better prepared for their professional careers. This work is a humble contribution towards such a purpose as it can be notified and developed further to become a highly effective educational tool that can open much wide horizons to future students of

structural engineering and contribute towards the overall improvement of their education and preparation for the engineers market.

6.2 Recommendations

1. The program JU.CAL is in a state of development and it is recommended to use the concept of object oriented programming to add additional operations to upgrade this program in future work like finite elements, nonlinear dynamic analysis, concrete design and other operations.

2. In the final stage of JU.DRS for the design response spectra it is recommended that special class or software be employed for plotting the tripartite response spectra curve.

REFERENCES:

1. Aitken, Peter (2001), “Visual Basic.Net Programming”, First Edition, Coriolis Group, Arizona U.S.A.

2. Armouti, Nazzal S., (2004), “Earthquake Engineering”, Theory and Implementation, University. of Jordan , Amman ,Jordan.

3. Assaf, Adel and Saffarini, Hassan (2004), “Optimization of Slabs using Object Oriented Programming”, Elsevier Ltd., J. Comp. Struc.

4. Assaf, Adel (2001), “Optimization of Reinforced Concrete Slabs”, Master's degree thesis, University of Jordan, Jordan.

5. Bathe and Wilson (1976), “Numerical Methods in Finite Element Analysis”, Prentice Hall Inc, USA.

6. Bittencourt, Marco L., and Feijoo, Raul A. (2001), “Object-Oriented Programming Applied to the Development of Structural analysis and Optimization Software”, J. Braz. Soc. Mech. Sci., 23 (3).

7. Buchholdt (1997), “Structural Dynamics for Engineers”, First Edition, Thomas Telford, London.

8. Chapra, Steven C. and Canale, Raymond P. (1998), “Numerical Methods for Engineers”, Third Edition, McGraw-Hill Inc.

9. Chopra (1995), "Dynamics of Structures Theory and Applications to Earthquake Engineering”, Prentice Hall Inc, New Jersey.

10. Clarke, D. (1978), “Computer Aided Structural Design”, First Edition, John Wiley and Sons.

11. Clough and Penzien (1993), “Dynamics of Structures”, Second Edition, McGraw-Hill Inc, Singapore.

12. Craig, Roy R. (1981), “Structural Dynamics, an Introduction to Computer Methods”, John Wiley and Sons, USA.

13. Deitel, Deitel and Nieto (2002), “Visual Basic .Net How to Program”, Prentice Hall Inc.

14. Fleming, John F. (1989), “Computer Analysis of Structural Systems”, First Edition, McGraw-Hill Inc.

15. Hamilton, J.P. (2002), "Object Oriented Programming with Visual Basic .Net", O'Reilly.

16. Haque (2001), “Web-based Visualization Techniques for Structural Design Education”, Proceedings of the 2001 American Society for Engineering Education Annual Conference Exposition.

17. Hart and Wong (2000), “Structural Dynamics for Structural Engineers”, John Wiley and Sons, USA.

18. Hibbeler, Russell C. (2002), “Structural analysis”, Fifth Edition, Prentice-Hall, Inc.

19. Iyengar, N.G.R. and Gupta, S.K. (1980), “Programming Methods in Structural Design”, First Edition, Edward Arnold.

20. Jenkins, W. M. (1990), “Structural Analysis using Computers”, Third Edition, Wiley, New York.

21. Jiang, H., Kurama, Y. C. and Fenella, D.A. (2002), “WWW-Based.Virtual Laboratories for Reinforced Concrete Education”, University of Notr.

22. Ju, Jianing and Hosain, M. U. (1996), “Finite-Element Graphic Objects in C++”, J. Comput. Civ. Eng., 10(3), 258-260.

23. Kablawi, Hana (1997), “Formulation of Design Response Spectra for Jordan”, Master's degree thesis, University of Jordan, Jordan.

24. Kassimali, Aslam (1999), “Matrix Analysis of Structures”, Brooks/Cole Publishing Company.

25. Madan, Alok (2004), "Object-Oriented Paradigm in Programming for Computer Aided Analysis of Structures", J. Comput. Civ. Eng., 18(3), 226-236.

26. Martha, Luiz F. and Junior, Evandro P. (2002), "An Object-Oriented Framework for Finite Element Programming", WCCM V.

27. Naeim (1989), "The Seismic Design Handbook", Van Nostrand Reinhold, New York.

28. Newmark and Hall (1982), "Earthquake Spectra and Design", Earthquake Engineering Research Institute, Berkeley, California.

29. "SAP2000- Integrated Finite Element Analysis and Design of Structures-Analysis References", Computer and Structures, Inc., Berkeley, CA, USA, 2002.

30. SAP 2000 Nonlinear Version 9.03 Structural Analysis Program ,Integrated Finite Elements Analysis and Design of Structures, Inc . Berkley , California ,USA.

31. Simon, Rich, Koorhan, Leslie and Cox, Ken (2002), "Object Oriented Programming with Visual Basic.Net", Second Edition, Sams, U.S.A.

32. Wilson, Edward (1979), "CAL- A Computer Analysis Language for Teaching Structural Analysis", Pergamon Press Ltd.

33. Wilson, Edward (2000).Three Dimentional Static and Dynamic Analysis of Structures .Berkeley , CA.Computers and Structures.

Appendix A

```
*******************************************************************************************
```

University of Jordan

JU.CAL - Classes

```
*******************************************************************************************
'-------------------------------------------------------------------------------------------
```

Matrix Operations - Class

```
'-------------------------------------------------------------------------------------------
Public Class MatrixOperations_Class
Dim i, j, k As Integer
*******************************************************************************************
    'Addition of Two Matrices

 Public Function Matrix_Add(ByVal Matrix_a(,), ByVal Matrix_b(,), ByRef Matrix_Result(,)
                                          , ByVal m_Matrix_a, ByVal n_Matrix_a)
        For i = 1 To m_Matrix_a
            For j = 1 To n_Matrix_a
                Matrix_Result(i, j) = Matrix_a(i, j) + Matrix_b(i, j)
            Next j
        Next i
    End Function
*******************************************************************************************

    'Determinate , Evaluate the determinate of a matrix
    Public Function Matrix_DET(ByVal Matrix_a(,), ByRef Matrix_Result(,),
                                                    ByVal m_Matrix_a, ByVal n_Matrix_a)
        Dim temp2(100, 100), mn, d, B
        d = 0
        If m_Matrix_a = 2 Then
 Matrix_Result(1, 1) = Matrix_a(1, 1) * Matrix_a(2, 2) - Matrix_a(1, 2) * Matrix_a(2, 1)
            Exit Function
        End If
        For k = 1 To m_Matrix_a
            mn = m_Matrix_a - 1
            For i = 1 To mn
                For j = 1 To mn
```

```
                    If j < k Then temp2(i, j) = Matrix_a(i + 1, j)
                    If j >= k Then temp2(i, j) = Matrix_a(i + 1, j + 1)
                Next j
            Next i
            If mn = 2 Then
                B = temp2(1, 1) * temp2(2, 2) - temp2(1, 2) * temp2(2, 1)
            Else
                B = Matrix_DET(temp2, Matrix_Result, mn, mn)
            End If
            d = d + ((-1) ^ (1 + k)) * Matrix_a(1, k) * B
        Next k
        Matrix_Result(1, 1) = d
    End Function
*******************************************************************************************

    'Duplicate, A new Matrix which is identical to the entered Matrix

 Public Function Matrix_DUP(ByVal Matrix_a(,), ByRef Matrix_Result(,), ByVal m_Matrix_a,
                                            ByVal n_Matrix_a)
        For i = 1 To m_Matrix_a
            For j = 1 To n_Matrix_a
                Matrix_Result(i, j) = Matrix_a(i, j)
            Next j
        Next i
    End Function
*******************************************************************************************

    'DUPDG , A new Row Matrix from the Diagonal terms of the entered Matrix

Public Function Matrix_DUPDG(ByVal Matrix_a(,), ByRef Matrix_Result(,),
                      ByVal m_Matrix_a, ByVal n_Matrix_a, ByRef Numbers_diagonal)
        Numbers_diagonal = 0
        For i = 1 To m_Matrix_a
            For j = 1 To n_Matrix_a
                If i = j Then
                    Numbers_diagonal = Numbers_diagonal + 1
                    Matrix_Result(1, Numbers_diagonal) = Matrix_a(i, j)
                End If
            Next j
```

```
        Next i
    End Function
*******************************************************************************************
    'DUPSM , This operation forms a new submatrix from the Main Matrix according to
             the columns and rows ranges.

   Public Function Matrix_DUPSM(ByVal Matrix_a(,), ByRef Matrix_Result(,), ByVal m1_Row,
                                ByVal n1_Column, ByVal m2_Row, ByVal n2_Column)
        For i = 1 To m2_Row - m1_Row + 1
            For j = 1 To n2_Column - n1_Column + 1
                Matrix_Result(i, j) = Matrix_a((i + m1_Row - 1), (j + n1_Column - 1))
            Next j
        Next i
    End Function
*******************************************************************************************

    'Inverse of the matrix

    Public Function Matrix_INVERSE(ByVal Matrix_a(,), ByRef Matrix_Result(,),
                                                    ByVal m_Matrix_a, ByVal n_Matrix_a)
        Dim Matrix(100, 100)
        Dim Cofactor(100, 100)
        Dim CofactorT(100, 100)
        Dim temp(100, 100)
        Dim h, L, dit, dita
        'forming cofactor matrix
        For L = 1 To m_Matrix_a
            For k = 1 To m_Matrix_a
                h = m_Matrix_a - 1
                For i = 1 To h
                    For j = 1 To h
                        If j < k And i < l Then
                            temp(i, j) = Matrix_a(i, j)
                        End If
                        If j >= k And i < l Then
                            temp(i, j) = Matrix_a(i, j + 1)
                        End If
                        If j < k And i >= l Then
                            temp(i, j) = Matrix_a(i + 1, j)
```

```
                End If
                If j >= k And i >= l Then
                    temp(i, j) = Matrix_a(i + 1, j + 1)
                End If
            Next j
        Next i
        Call Matrix_DET(temp, Matrix_Result, h, h)
        dit = Matrix_Result(1, 1)
        Cofactor(l, k) = ((-1) ^ (l + k)) * dit
    Next k
Next l

'C Transpose
For i = 1 To m_Matrix_a
    For j = 1 To m_Matrix_a
        CofactorT(i, j) = Cofactor(j, i)
    Next j
Next i

'inverse
Call Matrix_DET(Matrix_a, Matrix_Result, m_Matrix_a, m_Matrix_a)
dita = Matrix_Result(1, 1)

If dita = 0 Then
    MsgBox("The matrix is SINGULAR", vbOKOnly, "Note...")
    dita = 1 * 10 ^ -10
End If
If m_Matrix_a = 2 Then
    Matrix_Result(1, 1) = Matrix_a(2, 2) / dita
    Matrix_Result(1, 2) = -1 * Matrix_a(1, 2) / dita
    Matrix_Result(2, 1) = -1 * Matrix_a(2, 1) / dita
    Matrix_Result(2, 2) = Matrix_a(1, 1) / dita
Else
    For i = 1 To m_Matrix_a
        For j = 1 To m_Matrix_a
            Matrix_Result(i, j) = CofactorT(i, j) / dita
        Next j
    Next i
End If
```

```
    End Function
*******************************************************************************************
    ' MAX , This operation forms a column matrix in which each row  contains
       the maximum absolute value of the corresponding  row in the entered matrix.

 Public Function Matrix_MAX(ByVal Matrix_a(,), ByRef Matrix_Result(,), ByVal m_Matrix_a,
                                              ByVal n_Matrix_a)
        Dim MaxValue
        For i = 1 To m_Matrix_a
            MaxValue = 0
            For j = 1 To n_Matrix_a
                If System.Math.Abs(Matrix_a(i, j)) > MaxValue Then
                    MaxValue = System.Math.Abs(Matrix_a(i, j))
                End If
            Next j
            Matrix_Result(i, 1) = MaxValue
        Next i
    End Function
*******************************************************************************************

    'Multiplication of Two Matrices

Public Function Matrix_Multi(ByVal Matrix_a(,), ByVal Matrix_b(,), ByRef
                             Matrix_Result(,), ByVal m_Matrix_a, ByVal n_Matrix_a,
                             ByVal m_Matrix_b, ByVal n_Matrix_b)
        Dim X
        If (n_Matrix_a = m_Matrix_b) Then
            For i = 1 To m_Matrix_a
                For j = 1 To n_Matrix_b
                    X = 0
                    For k = 1 To n_Matrix_a
                        X = X + (Matrix_a(i, k) * Matrix_b(k, j))
                    Next k
                    Matrix_Result(i, j) = X
                Next j
            Next i
        Else
            MsgBox("Wrong with the Matrices Dimensions", vbOKOnly, "Error ...")
        End If
```

```
    End Function

*****************************************************************************************

    'NORM , To form a row matrix in which the column contains the Sum of the absolute
            values or The square roots of sum of squares for the corresponding
            column of the entered matrix.

Public Function Matrix_NORM(ByVal Matrix_a(,), ByRef Matrix_Result(,), ByVal m_Matrix_a,
                                            ByVal n_Matrix_a, ByVal Norm_Value)
        If Norm_Value = 1 Then
            For i = 1 To n_Matrix_a
                Matrix_Result(1, i) = 0
                For j = 1 To m_Matrix_a
                    Matrix_Result(1, i) = Matrix_Result(1, i) +
System.Math.Abs(Matrix_a(j, i))
                Next j
            Next i
        End If
        If Norm_Value = 2 Then
            For i = 1 To n_Matrix_a
                Matrix_Result(1, i) = 0
                For j = 1 To m_Matrix_a
                    Matrix_Result(1, i) = Matrix_Result(1, i) + (Matrix_a(j, i)) ^ 2
                Next j
                Matrix_Result(1, i) = (Matrix_Result(1, i)) ^ 0.5
            Next i
        End If
    End Function

*****************************************************************************************

    'LOG, This operation replaces each element in the entered matrix with
            the logarithm of the term.

 Public Function Matrix_LOG(ByVal Matrix_a(,), ByRef Matrix_Result(,), ByVal m_Matrix_a,
                                        ByVal n_Matrix_a, ByVal Const_Value)
        For i = 1 To m_Matrix_a
            For j = 1 To n_Matrix_a
```

```
                Matrix_Result(i, j) = Matrix_a(i, j) * Const_Value
            Next j
        Next i
    End Function
*****************************************************************************************

    'SCALE, multiply with constant

Public Function Matrix_SCALE(ByVal Matrix_a(,), ByRef Matrix_Result(,),
                ByVal m_Matrix_a, ByVal n_Matrix_a, ByVal Const_Value)

        For i = 1 To m_Matrix_a
            For j = 1 To n_Matrix_a
                Matrix_Result(i, j) = Matrix_a(i, j) * Const_Value
            Next j
        Next i
    End Function
*****************************************************************************************

    'SOLVE ,

Public Function Matrix_SOLVE(ByVal A_Matrix(,), ByVal C_Matrix(,)
                            , ByRef Result_Matrix(,), ByVal mMatrix, ByVal nMatrix)
        'Decomposition
        For j = 2 To nMatrix
            If A_Matrix(1, 1) = 0 Then
                MsgBox("Matrix is singular ... Zero Diagonal,
                               DEVOT  found in [D] Matrix to solve F=K.U")
                A_Matrix(1, 1) = 1 * 10 ^ -10
            End If
            A_Matrix(j, 1) = A_Matrix(j, 1) / A_Matrix(1, 1)
            If j <> 2 Then
                For i = 2 To j - 1
                    For k = 1 To i - 1
                       A_Matrix(i, j) = A_Matrix(i, j) - A_Matrix(i, k) * A_Matrix(k, j)
                       A_Matrix(j, i) = A_Matrix(j, i) - A_Matrix(j, k) * A_Matrix(k, i)
                    Next k
                    If A_Matrix(i, i) = 0 Then
                       MsgBox("Matrix is singular ... Zero Diagonal,
```

```
                                    DEVOT  found in [D] Matrix to solve F=K.U")
                        Exit Funcation
                    End If
                    A_Matrix(j, i) = A_Matrix(j, i) / A_Matrix(i, i)
                Next i
            End If
            For k = 1 To j - 1
                A_Matrix(j, j) = A_Matrix(j, j) - A_Matrix(j, k) * A_Matrix(k, j)
            Next k
        Next j
        'Forward Reduction
        For i = 2 To nMatrix
            For k = 1 To i - 1
                C_Matrix(i, 1) = C_Matrix(i, 1) - A_Matrix(i, k) * C_Matrix(k, 1)
            Next k
        Next i
        'Back substitution
        For i = nMatrix To 1 Step -1
            If i <> nMatrix Then
                For k = i + 1 To nMatrix
                    C_Matrix(i, 1) = C_Matrix(i, 1) - A_Matrix(i, k) * C_Matrix(k, 1)
                Next k
            End If
            C_Matrix(i, 1) = C_Matrix(i, 1) / A_Matrix(i, i)
        Next i

        For i = 1 To nMatrix
            Result_Matrix(i, 1) = System.Math.Round(C_Matrix(i, 1), 5)
        Next i
    End Function
*****************************************************************************************
```

'STODG, This operation stores a row or column matrix at the diagonal location of the main matrix in a new matrix.

```
    Public Function Matrix_STODG(ByVal A_Matrix(,), ByVal B_Matrix(,),
                        ByRef Result_Matrix(,), ByVal ma_Matrix, ByVal na_Matrix, ByVal
                       mB_Matrix, ByVal nB_Matrix)
        For i = 1 To ma_Matrix
```

```
            For j = 1 To na_Matrix
                Result_Matrix(i, j) = A_Matrix(i, j)
            Next j
        Next i
        'For row matrix
        If mB_Matrix = 1 Then
            For i = 1 To ma_Matrix
                For j = 1 To na_Matrix
                    If i = j Then
                        Result_Matrix(i, j) = B_Matrix(1, i)
                    End If
                Next j
            Next i
            'For column matrix
        ElseIf nB_Matrix = 1 Then
            For i = 1 To ma_Matrix
                For j = 1 To na_Matrix
                    If i = j Then
                        Result_Matrix(i, j) = B_Matrix(i, 1)
                    End If
                Next j
            Next i
        End If
    End Function
*****************************************************************************************

    'STOSM, This operation stores the sub matrix in  the main matrix  and save
                        the output in a new matrix.

    Public Function Matrix_STOSM(ByVal A_Matrix(,), ByVal B_Matrix(,),
                ByRef Result_Matrix(,), ByVal ma_Matrix, ByVal na_Matrix, ByVal
               mb_Matrix, ByVal nb_Matrix, ByVal m_Result, ByVal n_Result)
        For i = 1 To ma_Matrix
            For j = 1 To na_Matrix
                Result_Matrix(i, j) = A_Matrix(i, j)
            Next j
        Next i
        'To replace the submatrix values into the main matrix
        For i = 1 To mb_Matrix
```

```
            For j = 1 To nb_Matrix
                Result_Matrix(i + m_Result - 1, j + n_Result - 1) = B_Matrix(i, j)
            Next j
        Next i
    End Function
*****************************************************************************************

    'Subtraction of two matrices.

    Public Function Matrix_Subtraction(ByVal A_Matrix(,), ByVal B_Matrix(,), ByRef
Result_Matrix(,)
                                                , ByVal ma_Matrix, ByVal na_Matrix)
        For i = 1 To ma_Matrix
            For j = 1 To na_Matrix
                Result_Matrix(i, j) = A_Matrix(i, j) - B_Matrix(i, j)
            Next j
        Next i
    End Function
*****************************************************************************************

    'Transpose of the matrix.

    Public Function Matrix_Transpose(ByVal A_Matrix(,), ByRef Result_Matrix(,),
                                                ByVal ma_Matrix, ByVal na_Matrix)
        For i = 1 To ma_Matrix
            For j = 1 To na_Matrix
                Result_Matrix(j, i) = A_Matrix(i, j)
            Next j
        Next i
    End Function
*****************************************************************************************

    'Zero, This operation can be used to form null or unit matrices.

    Public Function Matrix_Zero(ByRef Result_Matrix(,), ByVal ma_Result_Matrix,
                            ByVal na_Result_Matrix, ByVal Zero_DiagonalValues, ByVal
                                    Zero_of_DiagonalValues)
        For i = 1 To ma_Result_Matrix
            For j = 1 To na_Result_Matrix
```

```
                If i = j Then
                    Result_Matrix(i, j) = Zero_DiagonalValues
                ElseIf i <> j Then
                    Result_Matrix(i, j) = Zero_of_DiagonalValues
                End If
            Next j
        Next i
    End Function
*****************************************************************************************
'-----------------------------------------------------------------------------------------
End Class ' Matrix Operation Class
'-----------------------------------------------------------------------------------------
*****************************************************************************************


'-----------------------------------------------------------------------------------------
     ' Direct Stiffness Operations - Class
'-----------------------------------------------------------------------------------------


Public Class DirectStiffnessOperations_Class

    Dim i, j, k As Integer

    ' ADDK, Adds the element stiffness matrix named [K1] to the total stiffness
              matrix named [Kt].

    Public Function DirectStiffness_ADDK(ByVal Matrix_k1(,), ByRef Matrix_Kt(,),
                     ByVal Matrix_LM(), ByVal m_Matrix_k1, ByVal n_Matrix_k1)
        Dim II, JJ
        For i = 1 To m_Matrix_k1
            II = Matrix_LM(i)
            If II <> 0 Then
                For j = 1 To n_Matrix_k1
                    JJ = Matrix_LM(j)
                    If JJ <> 0 Then
                        Matrix_Kt(II, JJ) = Matrix_Kt(II, JJ) + Matrix_k1(i, j)
                    End If
```

```
                Next j
            End If
        Next i
    End Function
**********************************************************************************
```

```
    ' MEMFRC, This operations multiplies the element stiffness matrix named [K1]
                    by the joint displacement matrix named [U1], The results of this
                    multiplication are stored in the array named [F1].

    Public Function DirectStiffness_MEMFRC(ByVal Matrix_K(,), ByVal Matrix_U(,)
                    , ByVal Matrix_LM(), ByRef Matrix_Result(,), ByVal m_Matrix_k,
            ByVal n_Matrix_k, ByVal m_Matrix_u, ByVal n_Matrix_u)
        Dim Sum, KK
        For i = 1 To m_Matrix_k
            For j = 1 To n_Matrix_u
                Sum = 0
                For k = 1 To n_Matrix_k
                    KK = Matrix_LM(k)
                    If KK <> 0 Then
                        Sum = Sum + Matrix_K(i, k) * Matrix_U(KK, j)
                    End If
                    Matrix_Result(i, j) = Sum
                Next k
            Next j
        Next i

    End Function
**********************************************************************************
```

```
    'SLOPE, This operation forms a 4 x 4 stiffness matrix, [K1], for a beam or
                column member from the classical slope deflection equations.

Public Function DirectStiffness_SLOPE(ByVal Moment_of_Inertia,
                ByVal Modulus_of_Elasticity , ByVal Length_of_Member, ByRef
                Matrix_Result(,))
        'Dimension: Matrix_Result(4,4)
        Dim II, E, L, EIL
        II = Moment_of_Inertia
```

```
        E = Modulus_of_Elasticity
        L = Length_of_Member
        EIL = E * II / L
        Matrix_Result(1, 1) = 4 * EIL
        Matrix_Result(1, 2) = 2 * EIL
        Matrix_Result(1, 3) = -6 * EIL / L
        Matrix_Result(1, 4) = -Matrix_Result(1, 3)
        Matrix_Result(2, 2) = Matrix_Result(1, 1)
        Matrix_Result(2, 3) = Matrix_Result(1, 3)
        Matrix_Result(2, 4) = Matrix_Result(1, 4)
        Matrix_Result(3, 3) = 12 * EIL / (L ^ 2)
        Matrix_Result(3, 4) = -Matrix_Result(3, 3)
        Matrix_Result(4, 4) = Matrix_Result(3, 3)
        Matrix_Result(2, 1) = Matrix_Result(1, 2)
        Matrix_Result(3, 1) = Matrix_Result(1, 3)
        Matrix_Result(3, 2) = Matrix_Result(2, 3)
        Matrix_Result(4, 1) = Matrix_Result(1, 4)
        Matrix_Result(4, 2) = Matrix_Result(2, 4)
        Matrix_Result(4, 3) = Matrix_Result(3, 4)
    End Function
*****************************************************************************************

    'FRAME, This operation forms the 6 x 6 stiffness matrix [K1] for the two-dimensional
                frame member.

    Public Function DirectStiffness_FRAME(ByVal Axial_Area, ByVal Moment_of_Inertia,
                                ByVal Modulus_of_Elasticity, ByVal X2, ByVal X1, ByVal Y1,
                                ByVal Y2, ByVal MATRIX_A(,), ByRef Matrix_Result(,))
        ' X1,X2,Y1,Y2 coordinate
        'Dimension: Matrix_Result(6,6), Matrix_A(3,6)
        Dim A, II, E, L, EIL, dx, dy, Sin, Cos, S12, S11, S33, T1, T2, T3
        II = Moment_of_Inertia
        E = Modulus_of_Elasticity
        A = Axial_Area
        dx = X2 - X1
        dy = Y2 - Y1
        L = (dx ^ 2 + dy ^ 2) ^ 0.5
        Sin = dy / L
        Cos = dx / L
```

```
        MATRIX_A(1, 1) = Sin / L
        MATRIX_A(1, 2) = -Cos / L
        MATRIX_A(1, 3) = 1
        MATRIX_A(1, 4) = -MATRIX_A(1, 1)
        MATRIX_A(1, 5) = -MATRIX_A(1, 2)
        MATRIX_A(1, 6) = 0
        MATRIX_A(2, 1) = MATRIX_A(1, 1)
        MATRIX_A(2, 2) = MATRIX_A(1, 2)
        MATRIX_A(2, 3) = 0
        MATRIX_A(2, 4) = MATRIX_A(1, 4)
        MATRIX_A(2, 5) = MATRIX_A(1, 5)
        MATRIX_A(2, 6) = 1
        MATRIX_A(3, 1) = -Cos
        MATRIX_A(3, 2) = -Sin
        MATRIX_A(3, 3) = 0
        MATRIX_A(3, 4) = Cos
        MATRIX_A(3, 5) = Sin
        MATRIX_A(3, 6) = 0
        S12 = 2 * E * II / L
        S11 = S12 + S12
        S33 = A * E / L
        For i = 1 To 6
            T1 = S11 * MATRIX_A(1, i) + S12 * MATRIX_A(2, i)
            T2 = S12 * MATRIX_A(1, i) + S11 * MATRIX_A(2, i)
            T3 = S33 * MATRIX_A(3, i)
            For j = 1 To 6
                Matrix_Result(j, i) = MATRIX_A(1, j) * T1 + MATRIX_A(2, j) * T2 +
MATRIX_A(3, j) * T3
                Matrix_Result(i, j) = Matrix_Result(j, i)
            Next j
            MATRIX_A(1, i) = T1
            MATRIX_A(2, i) = T2
            MATRIX_A(3, i) = T3
        Next i
    End Function
*******************************************************************************
'------------------------------------------------------------------------------
End Class ' DSM Operation Class
'------------------------------------------------------------------------------
```

```
'*****************************************************************************************
'-----------------------------------------------------------------------------------------
'  ' Dynamic Operations - Class
'-----------------------------------------------------------------------------------------
'-----------------------------------------------------------------------------------------
Public Class DynamicOperations_Class
'-----------------------------------------------------------------------------------------

    Dim i, N, N1, j, k As Integer

    'FUNG, This operation generates the matrix name [M2 ] which contains values at equal
                intervals, of the function specified in the array named [ M1]. The array
                [M1] must be a [ 2 x K] 'The time interval [dt] is specified in the
                [1x1] matrix named [M3].

    Public Function Dynamic_FUNG(ByVal Matrix_M1(,), ByVal Matrix_dt(,),
                        ByRef Matrix_Result(,), ByVal m_Matrix_M1, ByVal n_Matrix_M1)
        Dim II, JJ, T, NCC, S
        T = Matrix_M1(1, 1)
        JJ = 1
        NCC = n_Matrix_M1 - 1
        For II = 1 To NCC
S=(Matrix_M1(2, (II + 1))-(Matrix_M1(2, II)))/ (Matrix_M1(1, II + 1) - Matrix_M1(1, II))
100:        Matrix_Result(1, JJ) = T
            Matrix_Result(2, JJ) = Matrix_M1(2, II) + S * (T - Matrix_M1(1, II))
            If JJ = n_Matrix_M1 Then Exit Function
            JJ = JJ + 1
            T = T + Matrix_dt(1, 1)
            If T < Matrix_M1(1, (II + 1)) Then GoTo 100
        Next II
    End Function

'*****************************************************************************************
```

```
'STEP, This operations calculates the dynamic response of a structural system using
        direct step-by-step integration of the following linear matrix equations
       of motion.  MU''+CU'+KU = R(t)=PF(t)

Public Function Dynamic_STEP(ByVal Matrix_K(,), ByVal Matrix_M(,),
              ByVal Matrix_C(,),ByRef Matrix_Ui(,), ByVal Matrix_U(,), ByVal
       Matrix_D(),ByVal Matrix_F(), ByVal DTT, ByVal DELTA, ByVal ALFA, ByVal
       Theta, ByVal m_Matrix_K,ByVal n_Matrix_K, ByVal m_Matrix_Ui, ByVal
       m_Matrix_F, ByVal m_Matrix_D)

    'Compute Integration Constants
    Dim DT, A, A0, A1, A2, A3, A4, A5, A6, A7, A8, A9, FF, L, M, X, Y, DA, V
    If Theta = 0 Then Theta = 1
    DT = Theta * DTT
    A0 = 1 / (ALFA * DT ^ 2)
    A1 = DELTA / (ALFA * DT)
    A2 = 1 / (ALFA * DT)
    A3 = (0.5 / ALFA) - 1
    A4 = (DELTA / ALFA) - 1
    A5 = 0.5 * DT * ((DELTA / ALFA) - 2)
    A6 = DTT * (1 - DELTA)
    A7 = DTT * DELTA
    A8 = (0.5 - ALFA) * DTT ^ 2
    A9 = ALFA * DTT ^ 2
    'Form and Triangularize Effective Stiffness Matrix
    For i = 1 To m_Matrix_K
        For j = 1 To m_Matrix_K
         Matrix_K(i, j) = Matrix_K(i, j) + A0 * Matrix_M(i, j) + A1 * Matrix_C(i, j)
        Next j
    Next i
    Call Matrix_SymSol(Matrix_K, Matrix_Ui, m_Matrix_K, 1, 1)
    ' For Each Time Step
    k = 1
    For i = 1 To N1
        For j = 1 To N1
            ' 1. CALCULATE EFFECTIVE LOAD AT TIME T+DT
            k = k + 1
            FF = Matrix_F(k - 1) + Theta * (Matrix_F(k) - Matrix_F(k - 1))
            For L = 1 To m_Matrix_K
```

```
                    Matrix_U(L, i) = Matrix_D(L) * FF
                Next L
                For L = 1 To m_Matrix_K
                  X = A0 * Matrix_Ui(L, 1) + A2 * Matrix_Ui(L, 2) + A3 * Matrix_Ui(L, 3)
                  Y = A1 * Matrix_Ui(L, 1) + A4 * Matrix_Ui(L, 2) + A5 * Matrix_Ui(L, 3)
                    For M = 1 To m_Matrix_K
               Matrix_U(M, i) = Matrix_U(M, i) + X * Matrix_M(M, L) + Y * Matrix_C(M, L)
                    Next M
                Next L
                '2. SOLVE FOR DISPLACEMENT AT T+DT
                Call Matrix_SymSol(Matrix_K, Matrix_U, m_Matrix_K, 1, 2)
                '3. CALCULATE ACCELERATIONS AND VELOCITIES AT TIME T+DT
                For L = 1 To m_Matrix_K
A = A0 *(Matrix_U(L, i) - Matrix_Ui(L, 1)) - A2 * Matrix_Ui(L, 2) - A3 * Matrix_Ui(L, 3)
                    DA = (A - Matrix_Ui(L, 3)) / Theta
                    A = Matrix_Ui(L, 3) + DA
                    V = Matrix_Ui(L, 2) + A6 * Matrix_Ui(L, 3) + A7 * A Matrix_U(L, i) =
Matrix_Ui(L, 1) + DTT * Matrix_Ui(L, 2) + A8 * Matrix_Ui(L, 3) + A9 * A
                    Matrix_Ui(L, 3) = A
                    Matrix_Ui(L, 2) = V
                    Matrix_Ui(L, 1) = Matrix_U(L, i)
                Next L
            Next j
        Next i
    End Function
*******************************************************************************

    'EIGEN, This operation solves the following eigenvalue problem :
                [k].[phi] = [M].[phi].[Lamda]

    Public Function Dynamic_EIGEN(ByVal Matrix_K(,), ByRef M(), ByRef Matrix_U(,),
                                          ByRef m_Matrix_K, ByRef n_Matrix_K)

        Dim TEST, NRLM, TOLER, NN, N, NR, XMAX, II, JL, JJ, Kii, Kij, Kjj, D, H2, HT,
TN, CS, SN, Sin2, Cos2, IM
        TEST = 1 / (10 ^ 8)
        N = m_Matrix_K
        NN = N - 1
        NR = 0
```

```
        NRLM = 5 * N ^ 2
        TOLER = 0.1
        'Normalize to Unit Matrix
        For i = 1 To N
            M(i) = 1 / (Math.Sqrt(M(i)))
        Next i
        For i = 1 To N
            For j = 1 To N
                Matrix_K(i, j) = M(i) * Matrix_K(i, j) * M(j)
                Matrix_U(i, j) = 0
            Next j
            Matrix_U(i, i) = 1
        Next i
        'Reduce Matrix to Diagonal
50:     XMAX = 0
        For II = 1 To NN
            JL = II + 1
            For JJ = JL To N
                ' Check if Rotation is Required
                Kii = Matrix_K(II, II)
                Kij = Matrix_K(II, JJ)
                Kjj = Matrix_K(JJ, JJ)
                D = Math.Abs(Kii * Kjj)
                H2 = Kij * Kij
                If H2 > (XMAX * D) Then XMAX = (H2 / D)
                If H2 < (TOLER * D) Then GoTo 600
                ' Compute TAN , SIN  and COS
                NR = NR + 1
                HT = 0.5 * (Kii - Kjj) / Kij
                TN = -1.0 * HT - (Math.Sign(Math.Sqrt(HT * HT + 1)))
                CS = 1 / (Math.Sqrt(1 + TN ^ 2))
                SN = CS * TN
                Cos2 = CS ^ 2
                Sin2 = SN ^ 2
                'Reduce II , JJ Element to Zero
                HT = 2 * Kij * CS * SN
                Matrix_K(II, JJ) = 0
                Matrix_K(II, II) = Kii * Cos2 + HT + Kjj * Sin2
                Matrix_K(JJ, JJ) = Kii * Sin2 - HT + Kjj * Cos2
```

```
                For i = 1 To N
                    If (i - II) < 0 Then
                        HT = Matrix_K(i, II)
                        Matrix_K(i, II) = CS * HT + SN * Matrix_K(i, JJ)
                        Matrix_K(i, JJ) = -1.0 * SN * HT + CS * Matrix_K(i, JJ)
                    ElseIf (i - II) > 0 Then
                        If (i - JJ) < 0 Then
                            HT = Matrix_K(II, i)
                            Matrix_K(II, i) = CS * HT + SN * Matrix_K(i, JJ)
                            Matrix_K(i, JJ) = -1.0 * SN * HT + CS * Matrix_K(i, JJ)
                        ElseIf (i - JJ) > 0 Then
                            HT = Matrix_K(II, i)
                            Matrix_K(II, i) = CS * HT + SN * Matrix_K(JJ, i)
                            Matrix_K(JJ, i) = -1.0 * SN * HT + CS * Matrix_K(JJ, i)
                        End If
                    End If
530:            Next i
                'Operate on Eigenvectors
                For i = 1 To N
                    HT = Matrix_U(i, II)
                    Matrix_U(i, II) = CS * HT + SN * Matrix_U(i, JJ)
                    Matrix_U(i, JJ) = -1.0 * SN * HT + CS * Matrix_U(i, JJ)
                Next i
600:        Next JJ
700:    Next II
        ' Test for end of iteration and set new tolerance
        If NRLM < NR Then GoTo 1000
        If XMAX < TEST Then GoTo 710
        TOLER = 0.1 * XMAX
        GoTo 50
        'Normalize and order Eigenvectors
710:    For i = 1 To N
            For j = 1 To N
                Matrix_U(i, j) = Matrix_U(i, j) * M(i)
            Next j
            M(i) = Matrix_K(i, i)
        Next i
        ' Order Eigenvalues and Eigenvectors
        For i = 1 To NN
```

```
            JL = i + 1
            HT = M(i)
            IM = i
            For j = JL To N
                If (HT < M(j)) Then GoTo 850
                HT = M(j)
                IM = j
850:        Next j
            M(IM) = M(i)
            M(i) = HT
            For j = 1 To N
                HT = Matrix_U(j, i)
                Matrix_U(j, i) = Matrix_U(j, IM)
                Matrix_U(j, IM) = HT
            Next j
        Next i
1000:
    End Function
*******************************************************************************

    ' Symmetric equation solver

    Public Function Matrix_SymSol(ByRef A_Matrix(,), ByRef B_Matrix(,), ByVal ma_Matrix,
                                            ByVal nB_Matrix, ByVal M)

        'M=0 Triangularize and solve
        'M=1 Triangularize only
        'M=2 Forward reduction and backsubstitution only
        Dim D, L, LL
        LL = nB_Matrix
        If M = 2 Then GoTo 500
        For N = 1 To ma_Matrix
            If N = ma_Matrix Then GoTo 500
            D = A_Matrix(N, N)
            If D = 0 Then
 MsgBox("Matrix is singular ... Zero Diagonal,
DEVOT  found in [D] Matrix to solve F=K.U")
                D = 1 * 10 ^ -10
            End If
```

```
            N1 = N + 1
            For j = N1 To ma_Matrix
                If A_Matrix(N, j) = 0 Then GoTo 300
                A_Matrix(N, j) = (A_Matrix(N, j)) / D
200:            For k = j To ma_Matrix
                    A_Matrix(k, j) = A_Matrix(k, j) - A_Matrix(k, N) * A_Matrix(N, j)
                    A_Matrix(j, k) = A_Matrix(k, j)
                Next k
300:        Next j
400:    Next N
        '  FORWARD REDUCTION AND BACKSUBSTITUTION
500:    If M = 1 Then Exit Function
        For N = 1 To ma_Matrix
            For L = 1 To LL
                B_Matrix(N, L) = (B_Matrix(N, L)) / A_Matrix(N, N)
            Next L
            If N = ma_Matrix Then GoTo 800
            N1 = N + 1
            For L = 1 To LL
                For k = N1 To ma_Matrix
                    B_Matrix(k, L) = B_Matrix(k, L) - A_Matrix(k, N) * B_Matrix(N, L)
                Next k
            Next L
        Next N
800:    N1 = N
        N = N - 1
        If N = 0 Then Exit Function
        For L = 1 To LL
            For j = N1 To ma_Matrix
                B_Matrix(N, L) = B_Matrix(N, L) - A_Matrix(N, j) * B_Matrix(j, L)
            Next j
        Next L
        GoTo 800
    End Function
'*****************************************************************************************
'-----------------------------------------------------------------------------------------
End Class  ' Dynamic Operations - Class
'-----------------------------------------------------------------------------------------
```

Appendix B

****

Output Sap2000

****

SAP2000 v9.0.1 14/04/06 21:44:42

Table: Active Degrees of Freedom

UX	UY	UZ	RX	RY	RZ
Yes/No	Yes/No	Yes/No	Yes/No	Yes/No	Yes/No
Yes	No	No	No	No	No

Table: Analysis Case Definitions

Case	Type	InitialCond	ModalCase
Text	Text	Text	Text
LOAD1	LinStatic	Zero	
EIGENMODES	LinModal	Zero	
HIST1	LinModHist	Zero	EIGENMODES

Table: Assembled Joint Masses

Joint	U1	U2	U3	R1	R2	R3
Text	KN-s2/m	KN-s2/m	KN-s2/m	KN-m-s2	KN-m-s2	KN-m-s2
1	0.00	0.00	0.00	0.0000	0.0000	0.0000
2	10.00	0.00	0.00	0.0000	0.0000	0.0000
3	10.00	0.00	0.00	0.0000	0.0000	0.0000
4	10.00	0.00	0.00	0.0000	0.0000	0.0000
5	0.00	0.00	0.00	0.0000	0.0000	0.0000
6	10.00	0.00	0.00	0.0000	0.0000	0.0000
7	10.00	0.00	0.00	0.0000	0.0000	0.0000
8	10.00	0.00	0.00	0.0000	0.0000	0.0000
9	0.00	0.00	0.00	0.0000	0.0000	0.0000
10	10.00	0.00	0.00	0.0000	0.0000	0.0000
11	10.00	0.00	0.00	0.0000	0.0000	0.0000
12	10.00	0.00	0.00	0.0000	0.0000	0.0000

Table: Base Reactions, Part 1 of 1

OutputCase	CaseType	StepType	StepNum	GlobalFX	GlobalFY	GlobalFZ	GlobalMX	GlobalMY
Text	Text	Text	Unitless	KN	KN	KN	KN-m	KN-m
EIGENMODES	LinModal	Mode	1.000000	-863.802	0.000	0.000	0.0000	-1295.7026
EIGENMODES	LinModal	Mode	2.000000	-2210.997	0.000	0.000	0.0000	-3316.4958
EIGENMODES	LinModal	Mode	3.000000	-4497.165	0.000	0.000	0.0000	-6745.7474
HIST1	LinModHist	Max		133.696	0.000	0.000	0.0000	200.5447
HIST1	LinModHist	Min		-95.112	0.000	0.000	0.0000	-142.6678

Table: Case - Modal 1 - General

Case	ModeType	MaxNumModes	MinNumModes	EigenShift	EigenCutoff	EigenTol
Text	Text	Unitless	Unitless	Cyc/sec	Cyc/sec	Unitless
EIGENMODES	Eigen	3	1	0.0000E+00	0.0000E+00	1.0000E-05

Table: Case - Modal History 1 - General

Case	HistoryType	OutSteps	StepSize	DampingType	ConstDamp
Text	Text	Unitless	Unitless	Text	Unitless

HIST1	Transient	100	0.050000	Constant	0.0300

Table: Case - Modal History 2 - Load Assignments

Case	LoadType	LoadName	Function	LoadSF	TimeFactor	ArrivalTime
Text	Text	Text	Text	Unitless	Sec	Sec
HIST1	LoadPattern	LOAD1	FUNC1	1.000000	1.0000	0.0000

Table: Case - Static 1 - Load Assignments

Case	LoadType	LoadName	LoadSF
Text	Text	Text	Unitless
LOAD1	Load case	LOAD1	1.000000

Table: Connectivity - Frame

Frame	JointI	JointJ	IsCurved	Length	CentroidX	CentroidY	CentroidZ
Text	Text	Text	Yes/No	m	m	m	m
1	1	2	No	3.00000	-6.00000	0.00000	1.50000
2	2	3	No	3.00000	-6.00000	0.00000	4.50000
3	3	4	No	3.00000	-6.00000	0.00000	7.50000
4	5	6	No	3.00000	0.00000	0.00000	1.50000
5	6	7	No	3.00000	0.00000	0.00000	4.50000
6	7	8	No	3.00000	0.00000	0.00000	7.50000
7	9	10	No	3.00000	6.00000	0.00000	1.50000
8	10	11	No	3.00000	6.00000	0.00000	4.50000
9	11	12	No	3.00000	6.00000	0.00000	7.50000
10	2	6	No	6.00000	-3.00000	0.00000	3.00000
11	3	7	No	6.00000	-3.00000	0.00000	6.00000
12	4	8	No	6.00000	-3.00000	0.00000	9.00000
13	6	10	No	6.00000	3.00000	0.00000	3.00000
14	7	11	No	6.00000	3.00000	0.00000	6.00000
15	8	12	No	6.00000	3.00000	0.00000	9.00000

Table: Frame Section Assignments

Frame	SectionType	AutoSelect	AnalSect	DesignSect	MatProp
Text	Text	Text	Text	Text	Text
1	I/Wide Flange	N.A.	HE260-A	HE260-A	Default
2	I/Wide Flange	N.A.	HE220-A	HE220-A	Default
3	I/Wide Flange	N.A.	HE180-A	HE180-A	Default
4	I/Wide Flange	N.A.	HE260-A	HE260-A	Default
5	I/Wide Flange	N.A.	HE220-A	HE220-A	Default
6	I/Wide Flange	N.A.	HE180-A	HE180-A	Default
7	I/Wide Flange	N.A.	HE260-A	HE260-A	Default
8	I/Wide Flange	N.A.	HE220-A	HE220-A	Default
9	I/Wide Flange	N.A.	HE180-A	HE180-A	Default
10	I/Wide Flange	N.A.	HE180-A	HE180-A	Default
11	I/Wide Flange	N.A.	HE180-A	HE180-A	Default
12	I/Wide Flange	N.A.	HE180-A	HE180-A	Default
13	I/Wide Flange	N.A.	HE180-A	HE180-A	Default
14	I/Wide Flange	N.A.	HE180-A	HE180-A	Default
15	I/Wide Flange	N.A.	HE180-A	HE180-A	Default

Table: Frame Section Properties 01 - General, Part 1 of 6

SectionName	Material	Shape	t3	t2	tf	tw
Text	Text	Text	m	m	m	m
HE180-A	STEEL	I/Wide Flange	0.171000	0.180000	0.009500	0.006000
HE220-A	STEEL	I/Wide Flange	0.210000	0.220000	0.011000	0.007000
HE260-A	STEEL	I/Wide Flange	0.250000	0.260000	0.012500	0.007500

Table: Frame Section Properties 01 - General, Part 2 of 6

SectionName	t2b	tfb	Area	TorsConst	I33	I22	AS2
Text	m	m	m2	m4	m4	m4	m2
HE180-A	0.180000	0.009500	0.004530	1.490E-07	0.000025	9.250E-06	0.001026
HE220-A	0.220000	0.011000	0.006430	2.860E-07	0.000054	0.000020	0.001470
HE260-A	0.260000	0.012500	0.008680	5.420E-07	0.000104	0.000037	0.001875

Table: Frame Section Properties 01 - General, Part 3 of 6

SectionName	AS3	S33	S22	Z33	Z22	R33	R22
Text	m2	m3	m3	m3	m3	m	m
HE180-A	0.002850	0.000294	0.000103	0.000325	0.000156	0.074437	0.045188
HE220-A	0.004033	0.000515	0.000178	0.000568	0.000271	0.091726	0.055140
HE260-A	0.005417	0.000836	0.000282	0.000920	0.000430	0.109723	0.065006

Table: Frame Section Properties 01 - General, Part 4 of 6

SectionName	ConcCol	ConcBeam	Color	TotalWt	TotalMass	FromFile	AMod
Text	Yes/No	Yes/No	Text	KN	KN-s2/m	Yes/No	Unitless
HE180-A	No	No	Black	0.000	0.00	Yes	0.000000
HE220-A	No	No	Black	0.000	0.00	Yes	0.000000
HE260-A	No	No	Black	0.000	0.00	Yes	0.000000

Table: Frame Section Properties 01 - General, Part 5 of 6

SectionName	A2Mod	A3Mod	JMod	I2Mod	I3Mod	MMod	WMod
Text	Unitless	Unitless	Unitless	Unitless	Unitless	Unitless	Unitless
HE180-A	0.000000	0.000000	0.000000	0.000000	1.000000	0.000000	0.000000
HE220-A	0.000000	0.000000	0.000000	0.000000	1.000000	0.000000	0.000000
HE260-A	0.000000	0.000000	0.000000	0.000000	1.000000	0.000000	0.000000

Table: Frame Section Properties 01 - General, Part 6 of 6

SectionName	SectInFile	FileName
Text	Text	Text
HE180-A	HE180-A	c:\sap2000n\euro.pro
HE220-A	HE220-A	c:\sap2000n\euro.pro
HE260-A	HE260-A	c:\sap2000n\euro.pro

Table: Function - Time History - User

Name	Time	Value
Text	Sec	Unitless
FUNC1	0.0000	0.000000
FUNC1	0.0500	0.310000
FUNC1	0.1000	0.590000
FUNC1	0.1500	0.810000
FUNC1	0.2000	0.950000
FUNC1	0.2500	1.000000
FUNC1	0.3000	0.950000
FUNC1	0.3500	0.810000
FUNC1	0.4000	0.590000
FUNC1	0.4500	0.310000
FUNC1	0.5000	0.000000
FUNC1	0.5500	-0.310000
FUNC1	0.6000	-0.590000
FUNC1	0.6500	-0.810000
FUNC1	0.7000	-0.950000
FUNC1	0.7500	-1.000000
FUNC1	0.8000	-0.950000
FUNC1	0.8500	-0.810000
FUNC1	0.9000	-0.590000
FUNC1	0.9500	-0.310000
FUNC1	1.0000	0.000000

Table: Grid Lines

CoordSys	AxisDir	XRYZCoord	Visible
Text	Text	m	Yes/No
GLOBAL	X	-6.00000	Yes
GLOBAL	X	0.00000	Yes
GLOBAL	X	6.00000	Yes
GLOBAL	Y	0.00000	Yes
GLOBAL	Z	0.00000	Yes
GLOBAL	Z	3.00000	Yes
GLOBAL	Z	6.00000	Yes
GLOBAL	Z	9.00000	Yes

Table: Joint Accelerations - Absolute, Part 1 of 1

Joint	OutputCase	CaseType	StepType	StepNum	U1	U2	U3	R1
Text	Text	Text	Text	Unitless	m/sec2	m/sec2	m/sec2	rad/sec2
1	EIGENMODES	LinModal	Mode	1.000000	0.00000	0.00000	0.00000	0.000
1	EIGENMODES	LinModal	Mode	2.000000	0.00000	0.00000	0.00000	0.000
1	EIGENMODES	LinModal	Mode	3.000000	0.00000	0.00000	0.00000	0.000
1	HIST1	LinModHist	Max		0.00000	0.00000	0.00000	0.000
1	HIST1	LinModHist	Min		0.00000	0.00000	0.00000	0.000
2	EIGENMODES	LinModal	Mode	1.000000	-3.25679	0.00000	0.00000	0.000
2	EIGENMODES	LinModal	Mode	2.000000	-45.32179	0.00000	0.00000	0.000
2	EIGENMODES	LinModal	Mode	3.000000	-267.99062	0.00000	0.00000	0.000
2	HIST1	LinModHist	Max		0.29464	0.00000	0.00000	0.000
2	HIST1	LinModHist	Min		-0.30384	0.00000	0.00000	0.000
3	EIGENMODES	LinModal	Mode	1.000000	-8.83607	0.00000	0.00000	0.000
3	EIGENMODES	LinModal	Mode	2.000000	-79.03054	0.00000	0.00000	0.000
3	EIGENMODES	LinModal	Mode	3.000000	139.78021	0.00000	0.00000	0.000
3	HIST1	LinModHist	Max		0.77563	0.00000	0.00000	0.000
3	HIST1	LinModHist	Min		-0.84420	0.00000	0.00000	0.000
4	EIGENMODES	LinModal	Mode	1.000000	-16.70054	0.00000	0.00000	0.000
4	EIGENMODES	LinModal	Mode	2.000000	50.65243	0.00000	0.00000	0.000
4	EIGENMODES	LinModal	Mode	3.000000	-21.69510	0.00000	0.00000	0.000
4	HIST1	LinModHist	Max		1.45182	0.00000	0.00000	0.000
4	HIST1	LinModHist	Min		-1.56710	0.00000	0.00000	0.000
5	EIGENMODES	LinModal	Mode	1.000000	0.00000	0.00000	0.00000	0.000
5	EIGENMODES	LinModal	Mode	2.000000	0.00000	0.00000	0.00000	0.000
5	EIGENMODES	LinModal	Mode	3.000000	0.00000	0.00000	0.00000	0.000
5	HIST1	LinModHist	Max		0.00000	0.00000	0.00000	0.000
5	HIST1	LinModHist	Min		0.00000	0.00000	0.00000	0.000
6	EIGENMODES	LinModal	Mode	1.000000	-3.25679	0.00000	0.00000	0.000
6	EIGENMODES	LinModal	Mode	2.000000	-45.32179	0.00000	0.00000	0.000
6	EIGENMODES	LinModal	Mode	3.000000	-267.99062	0.00000	0.00000	0.000
6	HIST1	LinModHist	Max		0.29464	0.00000	0.00000	0.000
6	HIST1	LinModHist	Min		-0.30384	0.00000	0.00000	0.000
7	EIGENMODES	LinModal	Mode	1.000000	-8.83607	0.00000	0.00000	0.000
7	EIGENMODES	LinModal	Mode	2.000000	-79.03054	0.00000	0.00000	0.000
7	EIGENMODES	LinModal	Mode	3.000000	139.78021	0.00000	0.00000	0.000
7	HIST1	LinModHist	Max		0.77563	0.00000	0.00000	0.000
7	HIST1	LinModHist	Min		-0.84420	0.00000	0.00000	0.000
8	EIGENMODES	LinModal	Mode	1.000000	-16.70054	0.00000	0.00000	0.000
8	EIGENMODES	LinModal	Mode	2.000000	50.65243	0.00000	0.00000	0.000
8	EIGENMODES	LinModal	Mode	3.000000	-21.69510	0.00000	0.00000	0.000
8	HIST1	LinModHist	Max		1.45182	0.00000	0.00000	0.000
8	HIST1	LinModHist	Min		-1.56710	0.00000	0.00000	0.000
9	EIGENMODES	LinModal	Mode	1.000000	0.00000	0.00000	0.00000	0.000
9	EIGENMODES	LinModal	Mode	2.000000	0.00000	0.00000	0.00000	0.000
9	EIGENMODES	LinModal	Mode	3.000000	0.00000	0.00000	0.00000	0.000
9	HIST1	LinModHist	Max		0.00000	0.00000	0.00000	0.000
9	HIST1	LinModHist	Min		0.00000	0.00000	0.00000	0.000
10	EIGENMODES	LinModal	Mode	1.000000	-3.25679	0.00000	0.00000	0.000
10	EIGENMODES	LinModal	Mode	2.000000	-45.32179	0.00000	0.00000	0.000
10	EIGENMODES	LinModal	Mode	3.000000	-267.99062	0.00000	0.00000	0.000
10	HIST1	LinModHist	Max		0.29464	0.00000	0.00000	0.000
10	HIST1	LinModHist	Min		-0.30384	0.00000	0.00000	0.000
11	EIGENMODES	LinModal	Mode	1.000000	-8.83607	0.00000	0.00000	0.000
11	EIGENMODES	LinModal	Mode	2.000000	-79.03054	0.00000	0.00000	0.000
11	EIGENMODES	LinModal	Mode	3.000000	139.78021	0.00000	0.00000	0.000
11	HIST1	LinModHist	Max		0.77563	0.00000	0.00000	0.000
11	HIST1	LinModHist	Min		-0.84420	0.00000	0.00000	0.000
12	EIGENMODES	LinModal	Mode	1.000000	-16.70054	0.00000	0.00000	0.000
12	EIGENMODES	LinModal	Mode	2.000000	50.65243	0.00000	0.00000	0.000

12	EIGENMODES	LinModal	Mode	3.000000	-21.69510	0.00000	0.00000	0.000
12	HIST1	LinModHist	Max		1.45182	0.00000	0.00000	0.000
12	HIST1	LinModHist	Min		-1.56710	0.00000	0.00000	0.000

Table: Joint Accelerations - Relative, Part 1 of 1

Joint	OutputCase	CaseType	StepType	StepNum	U1	U2	U3	R1
Text	Text	Text	Text	Unitless	m/sec2	m/sec2	m/sec2	rad/sec2
1	EIGENMODES	LinModal	Mode	1.000000	0.00000	0.00000	0.00000	0.000
1	EIGENMODES	LinModal	Mode	2.000000	0.00000	0.00000	0.00000	0.000
1	EIGENMODES	LinModal	Mode	3.000000	0.00000	0.00000	0.00000	0.000
1	HIST1	LinModHist	Max		0.00000	0.00000	0.00000	0.000
1	HIST1	LinModHist	Min		0.00000	0.00000	0.00000	0.000
2	EIGENMODES	LinModal	Mode	1.000000	-3.25679	0.00000	0.00000	0.000
2	EIGENMODES	LinModal	Mode	2.000000	-45.32179	0.00000	0.00000	0.000
2	EIGENMODES	LinModal	Mode	3.000000	-267.99062	0.00000	0.00000	0.000
2	HIST1	LinModHist	Max		0.29464	0.00000	0.00000	0.000
2	HIST1	LinModHist	Min		-0.30384	0.00000	0.00000	0.000
3	EIGENMODES	LinModal	Mode	1.000000	-8.83607	0.00000	0.00000	0.000
3	EIGENMODES	LinModal	Mode	2.000000	-79.03054	0.00000	0.00000	0.000
3	EIGENMODES	LinModal	Mode	3.000000	139.78021	0.00000	0.00000	0.000
3	HIST1	LinModHist	Max		0.77563	0.00000	0.00000	0.000
3	HIST1	LinModHist	Min		-0.84420	0.00000	0.00000	0.000
4	EIGENMODES	LinModal	Mode	1.000000	-16.70054	0.00000	0.00000	0.000
4	EIGENMODES	LinModal	Mode	2.000000	50.65243	0.00000	0.00000	0.000
4	EIGENMODES	LinModal	Mode	3.000000	-21.69510	0.00000	0.00000	0.000
4	HIST1	LinModHist	Max		1.45182	0.00000	0.00000	0.000
4	HIST1	LinModHist	Min		-1.56710	0.00000	0.00000	0.000
5	EIGENMODES	LinModal	Mode	1.000000	0.00000	0.00000	0.00000	0.000
5	EIGENMODES	LinModal	Mode	2.000000	0.00000	0.00000	0.00000	0.000
5	EIGENMODES	LinModal	Mode	3.000000	0.00000	0.00000	0.00000	0.000
5	HIST1	LinModHist	Max		0.00000	0.00000	0.00000	0.000
5	HIST1	LinModHist	Min		0.00000	0.00000	0.00000	0.000
6	EIGENMODES	LinModal	Mode	1.000000	-3.25679	0.00000	0.00000	0.000
6	EIGENMODES	LinModal	Mode	2.000000	-45.32179	0.00000	0.00000	0.000
6	EIGENMODES	LinModal	Mode	3.000000	-267.99062	0.00000	0.00000	0.000
6	HIST1	LinModHist	Max		0.29464	0.00000	0.00000	0.000
6	HIST1	LinModHist	Min		-0.30384	0.00000	0.00000	0.000
7	EIGENMODES	LinModal	Mode	1.000000	-8.83607	0.00000	0.00000	0.000
7	EIGENMODES	LinModal	Mode	2.000000	-79.03054	0.00000	0.00000	0.000
7	EIGENMODES	LinModal	Mode	3.000000	139.78021	0.00000	0.00000	0.000
7	HIST1	LinModHist	Max		0.77563	0.00000	0.00000	0.000
7	HIST1	LinModHist	Min		-0.84420	0.00000	0.00000	0.000
8	EIGENMODES	LinModal	Mode	1.000000	-16.70054	0.00000	0.00000	0.000
8	EIGENMODES	LinModal	Mode	2.000000	50.65243	0.00000	0.00000	0.000
8	EIGENMODES	LinModal	Mode	3.000000	-21.69510	0.00000	0.00000	0.000
8	HIST1	LinModHist	Max		1.45182	0.00000	0.00000	0.000
8	HIST1	LinModHist	Min		-1.56710	0.00000	0.00000	0.000
9	EIGENMODES	LinModal	Mode	1.000000	0.00000	0.00000	0.00000	0.000
9	EIGENMODES	LinModal	Mode	2.000000	0.00000	0.00000	0.00000	0.000
9	EIGENMODES	LinModal	Mode	3.000000	0.00000	0.00000	0.00000	0.000
9	HIST1	LinModHist	Max		0.00000	0.00000	0.00000	0.000
9	HIST1	LinModHist	Min		0.00000	0.00000	0.00000	0.000
10	EIGENMODES	LinModal	Mode	1.000000	-3.25679	0.00000	0.00000	0.000
10	EIGENMODES	LinModal	Mode	2.000000	-45.32179	0.00000	0.00000	0.000
10	EIGENMODES	LinModal	Mode	3.000000	-267.99062	0.00000	0.00000	0.000
10	HIST1	LinModHist	Max		0.29464	0.00000	0.00000	0.000
10	HIST1	LinModHist	Min		-0.30384	0.00000	0.00000	0.000
11	EIGENMODES	LinModal	Mode	1.000000	-8.83607	0.00000	0.00000	0.000
11	EIGENMODES	LinModal	Mode	2.000000	-79.03054	0.00000	0.00000	0.000
11	EIGENMODES	LinModal	Mode	3.000000	139.78021	0.00000	0.00000	0.000
11	HIST1	LinModHist	Max		0.77563	0.00000	0.00000	0.000
11	HIST1	LinModHist	Min		-0.84420	0.00000	0.00000	0.000
12	EIGENMODES	LinModal	Mode	1.000000	-16.70054	0.00000	0.00000	0.000
12	EIGENMODES	LinModal	Mode	2.000000	50.65243	0.00000	0.00000	0.000
12	EIGENMODES	LinModal	Mode	3.000000	-21.69510	0.00000	0.00000	0.000
12	HIST1	LinModHist	Max		1.45182	0.00000	0.00000	0.000
12	HIST1	LinModHist	Min		-1.56710	0.00000	0.00000	0.000

Table: Joint Added Mass Assignments

Joint	CoordSys	Mass1	Mass2	Mass3	MMI1	MMI2	MMI3

Text	Text	KN-s2/m	KN-s2/m	KN-s2/m	KN-m-s2	KN-m-s2	KN-m-s2
2	Joint Local	10.00	0.00	0.00	0.0000	0.0000	0.0000
3	Joint Local	10.00	0.00	0.00	0.0000	0.0000	0.0000
4	Joint Local	10.00	0.00	0.00	0.0000	0.0000	0.0000
6	Joint Local	10.00	0.00	0.00	0.0000	0.0000	0.0000
7	Joint Local	10.00	0.00	0.00	0.0000	0.0000	0.0000
8	Joint Local	10.00	0.00	0.00	0.0000	0.0000	0.0000
10	Joint Local	10.00	0.00	0.00	0.0000	0.0000	0.0000
11	Joint Local	10.00	0.00	0.00	0.0000	0.0000	0.0000
12	Joint Local	10.00	0.00	0.00	0.0000	0.0000	0.0000

Table: Joint Constraint Assignments

Joint Text	Constraint Text	Type Text
2	DIAPH1	Diaphragm
3	DIAPH2	Diaphragm
4	DIAPH3	Diaphragm
6	DIAPH1	Diaphragm
7	DIAPH2	Diaphragm
8	DIAPH3	Diaphragm
10	DIAPH1	Diaphragm
11	DIAPH2	Diaphragm
12	DIAPH3	Diaphragm

Table: Joint Coordinates, Part 1 of 2

Joint Text	CoordSys Text	CoordType Text	XorR m	Y m	Z m	SpecialJt Yes/No	GlobalX m
1	GLOBAL	Cartesian	-6.00000	0.00000	0.00000	No	-6.00000
2	GLOBAL	Cartesian	-6.00000	0.00000	3.00000	No	-6.00000
3	GLOBAL	Cartesian	-6.00000	0.00000	6.00000	No	-6.00000
4	GLOBAL	Cartesian	-6.00000	0.00000	9.00000	No	-6.00000
5	GLOBAL	Cartesian	0.00000	0.00000	0.00000	No	0.00000
6	GLOBAL	Cartesian	0.00000	0.00000	3.00000	No	0.00000
7	GLOBAL	Cartesian	0.00000	0.00000	6.00000	No	0.00000
8	GLOBAL	Cartesian	0.00000	0.00000	9.00000	No	0.00000
9	GLOBAL	Cartesian	6.00000	0.00000	0.00000	No	6.00000
10	GLOBAL	Cartesian	6.00000	0.00000	3.00000	No	6.00000
11	GLOBAL	Cartesian	6.00000	0.00000	6.00000	No	6.00000
12	GLOBAL	Cartesian	6.00000	0.00000	9.00000	No	6.00000

Table: Joint Coordinates, Part 2 of 2

Joint Text	GlobalY m	GlobalZ m
1	0.00000	0.00000
2	0.00000	3.00000
3	0.00000	6.00000
4	0.00000	9.00000
5	0.00000	0.00000
6	0.00000	3.00000
7	0.00000	6.00000
8	0.00000	9.00000
9	0.00000	0.00000
10	0.00000	3.00000
11	0.00000	6.00000
12	0.00000	9.00000

Table: Joint Displacements, Part 1 of 1

Joint Text	OutputCase Text	CaseType Text	StepType Text	StepNum Unitless	U1 m	U2 m	U3 m	R1 Radians
1	EIGENMODES	LinModal	Mode	1.000000	0.000000	0.000000	0.000000	0.000000
1	EIGENMODES	LinModal	Mode	2.000000	0.000000	0.000000	0.000000	0.000000
1	EIGENMODES	LinModal	Mode	3.000000	0.000000	0.000000	0.000000	0.000000
1	HIST1	LinModHist	Max		0.000000	0.000000	0.000000	0.000000
1	HIST1	LinModHist	Min		0.000000	0.000000	0.000000	0.000000

2	EIGENMODES	LinModal	Mode	1.000000	0.031013	0.000000	0.000000	0.000000
2	EIGENMODES	LinModal	Mode	2.000000	0.079382	0.000000	0.000000	0.000000
2	EIGENMODES	LinModal	Mode	3.000000	0.161462	0.000000	0.000000	0.000000
2	HIST1	LinModHist	Max		0.003415	0.000000	0.000000	0.000000
2	HIST1	LinModHist	Min		-0.004800	0.000000	0.000000	0.000000
3	EIGENMODES	LinModal	Mode	1.000000	0.084143	0.000000	0.000000	0.000000
3	EIGENMODES	LinModal	Mode	2.000000	0.138423	0.000000	0.000000	0.000000
3	EIGENMODES	LinModal	Mode	3.000000	-0.084216	0.000000	0.000000	0.000000
3	HIST1	LinModHist	Max		0.009119	0.000000	0.000000	0.000000
3	HIST1	LinModHist	Min		-0.012814	0.000000	0.000000	0.000000
4	EIGENMODES	LinModal	Mode	1.000000	0.159033	0.000000	0.000000	0.000000
4	EIGENMODES	LinModal	Mode	2.000000	-0.088718	0.000000	0.000000	0.000000
4	EIGENMODES	LinModal	Mode	3.000000	0.013071	0.000000	0.000000	0.000000
4	HIST1	LinModHist	Max		0.016893	0.000000	0.000000	0.000000
4	HIST1	LinModHist	Min		-0.023742	0.000000	0.000000	0.000000
5	EIGENMODES	LinModal	Mode	1.000000	0.000000	0.000000	0.000000	0.000000
5	EIGENMODES	LinModal	Mode	2.000000	0.000000	0.000000	0.000000	0.000000
5	EIGENMODES	LinModal	Mode	3.000000	0.000000	0.000000	0.000000	0.000000
5	HIST1	LinModHist	Max		0.000000	0.000000	0.000000	0.000000
5	HIST1	LinModHist	Min		0.000000	0.000000	0.000000	0.000000
6	EIGENMODES	LinModal	Mode	1.000000	0.031013	0.000000	0.000000	0.000000
6	EIGENMODES	LinModal	Mode	2.000000	0.079382	0.000000	0.000000	0.000000
6	EIGENMODES	LinModal	Mode	3.000000	0.161462	0.000000	0.000000	0.000000
6	HIST1	LinModHist	Max		0.003415	0.000000	0.000000	0.000000
6	HIST1	LinModHist	Min		-0.004800	0.000000	0.000000	0.000000
7	EIGENMODES	LinModal	Mode	1.000000	0.084143	0.000000	0.000000	0.000000
7	EIGENMODES	LinModal	Mode	2.000000	0.138423	0.000000	0.000000	0.000000
7	EIGENMODES	LinModal	Mode	3.000000	-0.084216	0.000000	0.000000	0.000000
7	HIST1	LinModHist	Max		0.009119	0.000000	0.000000	0.000000
7	HIST1	LinModHist	Min		-0.012814	0.000000	0.000000	0.000000
8	EIGENMODES	LinModal	Mode	1.000000	0.159033	0.000000	0.000000	0.000000
8	EIGENMODES	LinModal	Mode	2.000000	-0.088718	0.000000	0.000000	0.000000
8	EIGENMODES	LinModal	Mode	3.000000	0.013071	0.000000	0.000000	0.000000
8	HIST1	LinModHist	Max		0.016893	0.000000	0.000000	0.000000
8	HIST1	LinModHist	Min		-0.023742	0.000000	0.000000	0.000000
9	EIGENMODES	LinModal	Mode	1.000000	0.000000	0.000000	0.000000	0.000000
9	EIGENMODES	LinModal	Mode	2.000000	0.000000	0.000000	0.000000	0.000000
9	EIGENMODES	LinModal	Mode	3.000000	0.000000	0.000000	0.000000	0.000000
9	HIST1	LinModHist	Max		0.000000	0.000000	0.000000	0.000000
9	HIST1	LinModHist	Min		0.000000	0.000000	0.000000	0.000000
10	EIGENMODES	LinModal	Mode	1.000000	0.031013	0.000000	0.000000	0.000000
10	EIGENMODES	LinModal	Mode	2.000000	0.079382	0.000000	0.000000	0.000000
10	EIGENMODES	LinModal	Mode	3.000000	0.161462	0.000000	0.000000	0.000000
10	HIST1	LinModHist	Max		0.003415	0.000000	0.000000	0.000000
10	HIST1	LinModHist	Min		-0.004800	0.000000	0.000000	0.000000
11	EIGENMODES	LinModal	Mode	1.000000	0.084143	0.000000	0.000000	0.000000
11	EIGENMODES	LinModal	Mode	2.000000	0.138423	0.000000	0.000000	0.000000
11	EIGENMODES	LinModal	Mode	3.000000	-0.084216	0.000000	0.000000	0.000000
11	HIST1	LinModHist	Max		0.009119	0.000000	0.000000	0.000000
11	HIST1	LinModHist	Min		-0.012814	0.000000	0.000000	0.000000
12	EIGENMODES	LinModal	Mode	1.000000	0.159033	0.000000	0.000000	0.000000
12	EIGENMODES	LinModal	Mode	2.000000	-0.088718	0.000000	0.000000	0.000000
12	EIGENMODES	LinModal	Mode	3.000000	0.013071	0.000000	0.000000	0.000000
12	HIST1	LinModHist	Max		0.016893	0.000000	0.000000	0.000000
12	HIST1	LinModHist	Min		-0.023742	0.000000	0.000000	0.000000

Table: Joint Loads - Force, Part 1 of 1

Joint	LoadCase	CoordSys	F1	F2	F3	M1	M2
Text	Text	Text	KN	KN	KN	KN-m	KN-m
2	LOAD1	GLOBAL	10.000	0.000	0.000	0.0000	0.0000
3	LOAD1	GLOBAL	20.000	0.000	0.000	0.0000	0.0000
4	LOAD1	GLOBAL	30.000	0.000	0.000	0.0000	0.0000

Table: Joint Reactions, Part 1 of 1

Joint	OutputCase	CaseType	StepType	StepNum	U1	U2	U3	R1
Text	Text	Text	Text	Unitless	KN	KN	KN	KN-m
1	EIGENMODES	LinModal	Mode	1.000000	-287.934	0.000	0.000	0.0000
1	EIGENMODES	LinModal	Mode	2.000000	-736.999	0.000	0.000	0.0000
1	EIGENMODES	LinModal	Mode	3.000000	-1499.055	0.000	0.000	0.0000
1	HIST1	LinModHist	Max		44.565	0.000	0.000	0.0000
1	HIST1	LinModHist	Min		-31.704	0.000	0.000	0.0000

5	EIGENMODES	LinModal	Mode	1.000000	-287.934	0.000	0.000	0.0000
5	EIGENMODES	LinModal	Mode	2.000000	-736.999	0.000	0.000	0.0000
5	EIGENMODES	LinModal	Mode	3.000000	-1499.055	0.000	0.000	0.0000
5	HIST1	LinModHist	Max		44.565	0.000	0.000	0.0000
5	HIST1	LinModHist	Min		-31.704	0.000	0.000	0.0000
9	EIGENMODES	LinModal	Mode	1.000000	-287.934	0.000	0.000	0.0000
9	EIGENMODES	LinModal	Mode	2.000000	-736.999	0.000	0.000	0.0000
9	EIGENMODES	LinModal	Mode	3.000000	-1499.055	0.000	0.000	0.0000
9	HIST1	LinModHist	Max		44.565	0.000	0.000	0.0000
9	HIST1	LinModHist	Min		-31.704	0.000	0.000	0.0000

Table: Joint Reactions, Part 2 of 2

Joint	OutputCase	StepType	StepNum	R2	R3
Text	Text	Text	Unitless	KN-m	KN-m
1	EIGENMODES	Mode	1.000000	-431.9009	0.0000
1	EIGENMODES	Mode	2.000000	-1105.4986	0.0000
1	EIGENMODES	Mode	3.000000	-2248.5825	0.0000
1	HIST1	Max		66.8482	0.0000
1	HIST1	Min		-47.5559	0.0000
5	EIGENMODES	Mode	1.000000	-431.9009	0.0000
5	EIGENMODES	Mode	2.000000	-1105.4986	0.0000
5	EIGENMODES	Mode	3.000000	-2248.5825	0.0000
5	HIST1	Max		66.8482	0.0000
5	HIST1	Min		-47.5559	0.0000
9	EIGENMODES	Mode	1.000000	-431.9009	0.0000
9	EIGENMODES	Mode	2.000000	-1105.4986	0.0000
9	EIGENMODES	Mode	3.000000	-2248.5825	0.0000
9	HIST1	Max		66.8482	0.0000
9	HIST1	Min		-47.5559	0.0000

Table: Joint Restraint Assignments

Joint	U1	U2	U3	R1	R2	R3
Text	Yes/No	Yes/No	Yes/No	Yes/No	Yes/No	Yes/No
1	Yes	Yes	Yes	Yes	Yes	Yes
5	Yes	Yes	Yes	Yes	Yes	Yes
9	Yes	Yes	Yes	Yes	Yes	Yes

Table: Material List 1 - By Object Type

ObjectType	Material	TotalWeight	NumPieces
Text	Text	KN	Unitless
Frame	STEEL	0.000	15

Table: Material List 2 - By Section Property

Section	ObjectType	NumPieces	TotalLength	TotalWeight
Text	Text	Unitless	m	KN
HE180-A	Frame	9	45.00000	0.000
HE220-A	Frame	3	9.00000	0.000
HE260-A	Frame	3	9.00000	0.000

Table: Material Properties 01 - General, Part 1 of 2

Material	Type	DesignType	UnitMass	UnitWeight	E	U	A
Text	Text	Text	KN-s2/m4	KN/m3	KN/m2	Unitless	1/C
CONC	Isotropic	Concrete	2.4007E+00	2.3562E+01	24821128.40	0.200000	9.9000E-06
OTHER	Isotropic	None	2.4007E+00	2.3562E+01	24821128.40	0.200000	9.9000E-06
STEEL	Isotropic	Steel	7.8271E+00	7.6820E+01	199900000.0	0.300000	1.1700E-05

Table: Material Properties 01 - General, Part 2 of 2

Material	MDampRatio	VDampMass	VDampStiff	HDampMass	HDampStiff	NumAdvance	Color
Text	Unitless	1/Sec	Sec	1/Sec2	Unitless	Unitless	Text

CONC	0.0000	0.0000	0.0000	0.0000	0.000000	0	Black
OTHER	0.0000	0.0000	0.0000	0.0000	0.000000	0	Black
STEEL	0.0000	0.0000	0.0000	0.0000	0.000000	0	Black

Table: Material Properties 03 - Design Steel

Material	Fy	Fu
Text	KN/m2	KN/m2
STEEL	248211.28	310264.10

Table: Material Properties 04 - Design Concrete

Material	Fc	RebarFy	RebarFys	LtWtConc	LtWtFact
Text	KN/m2	KN/m2	KN/m2	Yes/No	Unitless
CONC	27579.03	413685.47	275790.32	No	1.000000

Table: Material Properties 07 - Time Dependence For Steel

Material	Relaxation	Class
Text	Yes/No	Unitless
STEEL	No	1

Table: Material Properties 08 - Time Dependence For Concrete, Part 1 of 2

Material	E	Creep	Shrinkage	S	RelHumid	NotionSize	BetaSC	ShrinkStart
Text	Yes/No	Yes/No	Yes/No	Unitless	Percent	m	Unitless	Unitless
CONC	No	No	No	0.250000	50.0000	0.100000	5.000000	0.000000

Table: Material Properties 08 - Time Dependence For Concrete, Part 2 of 2

Material	BetaSC	CreepType
Text	Unitless	Text
CONC	5.000000	Full Integration

Table: Material Properties 09 - Stress-Strain Curves 1 - General

Material	SSCurve	HysType	Color
Text	Text	Text	Text
CONC	Default-Rebar	Kinematic	Gray8Dark
CONC	Default-Unconfined	Takeda	Red
CONC	Default-Confined	Takeda	Blue
STEEL	Default-Steel	Kinematic	Gray8Dark

Table: Material Properties 10 - Stress-Strain Curves 2 - Data

Material	SSCurve	Point	Strain	Stress	PointID
Text	Text	Text	Unitless	KN/m2	Text
CONC	Default-Rebar	1	-0.050000	0.00	-E
CONC	Default-Rebar	2	-0.035000	-206842.74	-D
CONC	Default-Rebar	3	-0.020000	-413685.47	-C
CONC	Default-Rebar	4	-0.002069	-413685.47	-B
CONC	Default-Rebar	5	0.000000	0.00	A
CONC	Default-Rebar	6	0.002069	413685.47	B
CONC	Default-Rebar	7	0.020000	413685.47	
CONC	Default-Rebar	8	0.050000	517106.84	C
CONC	Default-Rebar	9	0.080000	517106.84	D
CONC	Default-Rebar	10	0.100000	413685.47	E
CONC	Default-Unconfined	1	-0.011111	0.00	
CONC	Default-Unconfined	2	-0.002222	-27579.03	-C
CONC	Default-Unconfined	3	-0.002000	-27303.24	
CONC	Default-Unconfined	4	-0.001778	-26475.87	-B
CONC	Default-Unconfined	5	-0.001333	-23166.39	

CONC	Default-Unconfined	6	-0.000889	-17650.58	
CONC	Default-Unconfined	7	-0.000444	-9928.45	
CONC	Default-Unconfined	8	0.000000	0.00	A
CONC	Default-Unconfined	9	0.000105	2616.38	
CONC	Default-Unconfined	10	0.000106	0.00	
CONC	Default-Confined	1	-0.024444	0.00	
CONC	Default-Confined	2	-0.022222	-5515.81	-E
CONC	Default-Confined	3	-0.017778	-5515.81	-D
CONC	Default-Confined	4	-0.002222	-27579.03	-C
CONC	Default-Confined	5	-0.002000	-27303.24	
CONC	Default-Confined	6	-0.001778	-26475.87	-B
CONC	Default-Confined	7	-0.001333	-23166.39	
CONC	Default-Confined	8	-0.000889	-17650.58	
CONC	Default-Confined	9	-0.000444	-9928.45	
CONC	Default-Confined	10	0.000000	0.00	A
CONC	Default-Confined	11	0.000105	2616.38	
CONC	Default-Confined	12	0.000106	0.00	
STEEL	Default-Steel	1	-0.050000	0.00	-E
STEEL	Default-Steel	2	-0.035000	-124105.64	-D
STEEL	Default-Steel	3	-0.020000	-248211.28	-C
STEEL	Default-Steel	4	-0.001241	-248211.28	-B
STEEL	Default-Steel	5	0.000000	0.00	A
STEEL	Default-Steel	6	0.001241	248211.28	B
STEEL	Default-Steel	7	0.020000	248211.28	
STEEL	Default-Steel	8	0.050000	399895.96	C
STEEL	Default-Steel	9	0.080000	399895.96	D
STEEL	Default-Steel	10	0.100000	248211.28	E

Table: Modal Load Participation Ratios

OutputCase	ItemType	Item	Static	Dynamic
Text	Text	Text	Percent	Percent
EIGENMODES	Acceleration	UX	100.0000	100.0000
EIGENMODES	Acceleration	UY	0.0000	0.0000
EIGENMODES	Acceleration	UZ	0.0000	0.0000

Table: Modal Participating Mass Ratios, Part 1 of 3

OutputCase	StepType	StepNum	Period	UX	UY	UZ	SumUX	SumUY
Text	Text	Unitless	Sec	Unitless	Unitless	Unitless	Unitless	Unitless
EIGENMODES	Mode	1.000000	0.613138	0.75180	0.00000	0.00000	0.75180	0.00000
EIGENMODES	Mode	2.000000	0.262958	0.16663	0.00000	0.00000	0.91843	0.00000
EIGENMODES	Mode	3.000000	0.154225	0.08157	0.00000	0.00000	1.00000	0.00000

Table: Modal Participating Mass Ratios, Part 2 of 3

OutputCase	StepType	StepNum	SumUZ	RX	RY	RZ	SumRX	SumRY
Text	Text	Unitless	Unitless	Unitless	Unitless	Unitless	Unitless	Unitless
EIGENMODES	Mode	1.000000	0.00000	0.00000	0.98039	0.00000	0.00000	0.98039
EIGENMODES	Mode	2.000000	0.00000	0.00000	0.01739	0.00000	0.00000	0.99777
EIGENMODES	Mode	3.000000	0.00000	0.00000	0.00223	0.00000	0.00000	1.00000

Table: Modal Participating Mass Ratios, Part 3 of 3

OutputCase	StepType	StepNum	SumRZ
Text	Text	Unitless	Unitless
EIGENMODES	Mode	1.000000	0.00000
EIGENMODES	Mode	2.000000	0.00000
EIGENMODES	Mode	3.000000	0.00000

Table: Modal Participation Factors, Part 1 of 2

OutputCase	StepType	StepNum	Period	UX	UY	UZ	RX	RY
Text	Text	Unitless	Sec	KN-s2	KN-s2	KN-s2	KN-m-s2	KN-m-s2
EIGENMODES	Mode	1.000000	0.613138	-8.225668	0.000000	0.000000	0.000000	-60.875804
EIGENMODES	Mode	2.000000	0.262958	-3.872590	0.000000	0.000000	0.000000	-8.106533
EIGENMODES	Mode	3.000000	0.154225	-2.709507	0.000000	0.000000	0.000000	-2.901839

Table: Modal Participation Factors, Part 2 of 2

OutputCase	StepType	StepNum	RZ	ModalMass	ModalStiff
Text	Text	Unitless	KN-m-s2	KN-m-s2	KN-m
EIGENMODES	Mode	1.000000	0.000000	1.0000	105.01295
EIGENMODES	Mode	2.000000	0.000000	1.0000	570.93501
EIGENMODES	Mode	3.000000	0.000000	1.0000	1659.77248

Table: Modal Periods And Frequencies

OutputCase	StepType	StepNum	Period	Frequency	CircFreq	Eigenvalue
Text	Text	Unitless	Sec	Cyc/sec	rad/sec	rad2/sec2
EIGENMODES	Mode	1.000000	0.613138	1.6310E+00	1.0248E+01	1.0501E+02
EIGENMODES	Mode	2.000000	0.262958	3.8029E+00	2.3894E+01	5.7094E+02
EIGENMODES	Mode	3.000000	0.154225	6.4840E+00	4.0740E+01	1.6598E+03

تعلم الهندسة الإنشائية بمساعدة الحاسوب باستخدام البرمجة غرضية التوجه

إعداد
أسعد سعيد أبو خلف

المشرف
أ. د حسان سفاريني

الملخص

في هذا البحثِ، تم تطوير برنامج الحاسوبَ J U.CAL للمُسَاعَدَة في تعلم مفاهيم التحليل الإنشائي للمباني بواسطة الحاسوب بطريقة المصفوفات. إن الطرق التقليدية لتعلم هذه المفاهيم تتطلب وقتا طويلا وجهدا مضنيا من حيث القيام بعمل حسابات يدويه لهذه الطرق. لذا جاء برنامج J U.CAL والذي صُمّمَ باستخدام لغة البرمجة Visual Basic .Net - 2003 لتجسير هذه الفجوه في تعلم المفاهيم المشار إليها أعلاه. إن برنامج J U.CAL يستخدم البرمجة غرضية التوجه (OOP) بشكل أساسي. هذه النظرةِ في البرمجة حالياً هي المنهجية المستخدمة في تطوير اكثر واحدث البرامج. إن الخصائص القوية والمرنة للبرمجة غرضية التوجه تسمح للمبرمج لإنتاج برامج قابلة للدعم وللإثباتِ أكثر وضوحاً مع درجة عالية مِنْ الكفاءةِ.

J U.CAL يَشْملُ ثلاثة أجزاءِ: الجزء الأول هو عمليات المصفوفة التي تُستَعملُ لإدخال مصفوفات إلى الحاسوب وذلك بتَعريف اسمِ المصفوفة وقِيَمَ مداخلِ المصفوفة ومن ثم يتم استخدامها في العمليات الحسابية مثل عمليات الإضافة والضرب والطرح وقلب القيم وغيرها من العمليات الرياضية الخاصة بالمصفوفات. هذه العملياتِ قَدْ تُستَعملُ لوحدها أو بالارتباط مع الجزء الثاني و/ أو الجزء الثالث مِنْ البرنامج. يَستعملُ الجزءُ الثانيُ في عمليات حلول الجساءه المباشرةِ لتَشكيل جساءه كل عنصر من عناصر المبنى رياضيا ومن ثم دمجها لتشكيل الجساءه الكليه للمنشأ بالطرق الرياضية المتعارف عليها في علم الهندسة الإنشائية. يَصِفُ الجزءُ الثالثُ تحليل ديناميكي مثل طريقةِ التكامل التدريجيةِ وعمليات eigenvalues and eigenvectors. إن J U.CAL مُقَسَّمُ إلى سلسلة منطقيّة مِنْ العملياتِ المنفصلةِ.

إضافة إلى ذلك J U.CAL يحتوي على برنامجان منفصلان، تحليل ديناميكي للأبنية المتعددة الطوابق وكذلك برنامج احتساب استجابة الطيف الترددي من قراءات سيزموجرافيه.

أخيراً، لقد تمت مقارنة نَتائِج J U.CAL مع البرامج التجاريةِ المعروفة وجاءت المقارنةِ جيدةِ بين تلك النَتائِجُ.