Stability of Discrete Time Transfer Matrix Method (DT-TMM)

by Vahid Alizadehyazdi, Bachelor of Science

A Thesis Submitted in Partial
Fulfillment of the Requirements
for the Degree of
Master of Science
in the field of Mechanical Engineering

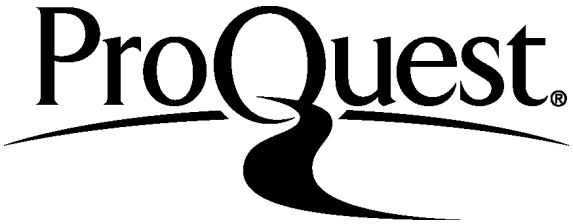
Advisory Committee:

Ryan Krauss, Chair

Keqin Gu

Fengxia Wang

Graduate School
Southern Illinois University Edwardsville
May, 2016



ABSTRACT

STABILITY OF DISCRETE TIME TRANSFER MATRIX METHOD (DT-TMM)

by

VAHID ALIZADEHYAZDI

Chairperson: Professor Ryan Krauss

Large dynamic systems and flexible structures like long robot links with many degree of freedoms are always challenging issues for engineers to model and control. These structures can be modeled with some methods like modal superposition and numerical integration.

The Transfer Matrix Method (TMM) is another method that can be used to model large systems with a huge number of subsystems and flexible structures. The size of matrices in transfer matrix models remain small regardless of the number of elements in model. Having smaller matrix sizes helps us to have less computational expense leading to a faster answer. Also, this method is very flexible, because it is possible for us to add or eliminate one subsystem easily. The transfer matrix method like other methods has its drawbacks. The TMM is limited to linear systems and can not be used for non-linear ones. Moreover, this method just gives frequency-domain output and can not perform time-domain simulation.

By combining TMM and numerical integration methods, we have a new method which is called the Discrete-Time Transfer Matrix Method (DT-TMM). The DT-TMM can model non-linear systems too. Time-domain output is another advantage of this method. Two approaches are considered in this research to combine the TMM and

numerical integration. The first approach is describing acceleration and velocity based on the displacement. Another approach is using acceleration to calculate the velocity and displacement. Also, different methods of numerical integration like Fox-Euler, Houbolt, Park Stiffly Stable, Newmark Beta and Wilson θ are studied in this research.

ACKNOWLEDGEMENTS

I would like to express my special thanks and gratitude to my adviser Dr. Ryan Krauss who gave me this opportunity to do my research under his guidance. His patience and support helped me overcome barriers. I want to thank him for his financial support through the mechanical engineering department. I learned a lot of things from him, not only in our research but also about responsibility, kindness and teaching. It has been my pleasure to work for him, and I hope in the future I can be someone like him in my professional and personal life.

I would like to thank Dr. Fengxia Wang,who gave me the Chance of working with her for about one year. I appreciate her support and help and would like to thank her for being part of the committee.

Also, I want to thank Dr. Majid Molki,the department chair for the financial support through the mechanical engineering department.

Thanks to Dr.Keqin Gu, who helped me a lot during my admission process and thanks for being part of the committee.

TABLE OF CONTENTS

ABSTRACT ii

ACKNOWLEDGEMENTS iv

LIST OF FIGURES vii

LIST OF TABLES vii

Chapter

1. INTRODUCTION 1

1.1 Problem Statement 1
1.2 Software Design 2

2. LITERATURE REVIEW 3

3. Introduction to State Vector and Transfer Matrix Method 8

3.1 State Vector 8
3.2 Transfer Matrix Method 8

4. Introduction to Discrete Time Transfer Matrix Method(DT-TMM) . . . 22

4.1 Why DT-TMM 22
4.2 DT-TMM for One Degree of Freedom System 22
4.3 Pyhton Code for One Degree of Freedom DT-TMM 31

5. ODE Integration of n Degree of Freedom System 37

6. Numerical Integration Methods in DT-TMM 46

6.1 Introduction 46
6.2 Explicit Method 46
6.3 Implicit Method 47
6.4 Numerical Integration Methods of DT-TMM 48
6.4.1 Fox-Euler 48
6.4.2 Newmark Beta 50
6.4.3 Wilson Theta 56
6.4.4 Houbolt 58
6.4.5 Park Stiffly Stable 61

7. New Approach Based on the Acceleration for DT-TMM 66

7.1 One Degree of Freedom System 66
7.2 Numerical Integration of New Method 72
7.2.1 Fox-Euler 72
7.2.2 Newmark Beta 74
7.2.3 Wilson θ 76
7.3 Pyhton Code for n Degree of Freedom with New Approach of DT-TMM 79

8. CONCLUSION 81

8.1 Conclusion 81
8.2 Future Work 82

REFERENCES 84

APPENDICES 87

A. DT-TMM (One Degree of Freedom) 87

B. ODE Integration (n Degree of Freedom) 89

C. DT-TMM n Degree of Freedom 93

D. DT-TMM n Degree of Freedom Based on the Acceleration 99

LIST OF FIGURES

Figure		Page
3.1	is a simple one degree of freedom mass-damper system which is exploded in	9
3.2	and as can be seen all internal forces and displacements described in details.	9
3.3	shows a two-degree of freedom mass-spring-damper system with internal forces	13
3.4	Exploded view of the two degree of freedom mass-spring-damper system	14
4.1	One degree of freedom mass-spring-damper system	25
4.2	Exploded view of one degree of freedom mass-spring-damper system	25
4.3	Step response of one degree of freedom mass-spring-damper system (DT-TMM)	36
5.1	$n+1$ degree of freedom mass-spring-damper system	37
5.2	Two degree of freedom mass-spring-damper system	38
5.3	Step response of two degree of freedom system (odeint code)	44
5.4	Comparison of DT-TMM and odeint output for one degree of freedom system	45
6.1	Comparison of DT-TMM Fox-Euler and odeint fot the two DOF system	51
6.2	Comparison of DT-TMM Newmark Beta and odeint for the two DOF system	54
6.3	Comparison of DT-TMM Newmark Beta for different β	55
6.4	Wilson θ	56
6.5	Comparison of Wilson θ with different θ with *odeint*	58
6.6	Comparison of Houbolt and *odeint* for the two degree of freedom system	60
6.7	Comparison of Park Stiffly Stable and *odeint* for the two degree of freedom system	62
6.8	Step response of 15 DOF system with different DT-TMM method	64
6.9	Step response of the 15 DOF system based on Newmark Beta for different β	65
7.1	One degree of freedom mass-spring-damper system	67
7.2	Exploded view of One degree of freedom mass-spring-damper system	67
7.3	Comparison of Fox-Euler DT-TMM based on acceleration and displacement with *odeint*	73
7.4	Comparison of Newmark Beta DT-TMM based on acceleration and displacement with *odeint*	75
7.5	Comparison of Wilson θ DT-TMM based on acceleration and displacement with *odeint* when $\theta = 0.9$	77
7.6	Comparison of Wilson θ DT-TMM based on acceleration and displacement with *odeint* when $\theta = 1.03$	78

LIST OF TABLES

Table Page

6.1 Parameters of the system (DOF=15). 65

CHAPTER 1

INTRODUCTION

These days, large systems with many subsystems and flexible structures (due to fast response and lower weight) can be seen in many systems in industry. Also, in some cases simultaneously we have flexible and rigid structures in a system. Modeling of rigid elements in a system is not a challenging problem. By using flexible structures, engineers are faced with new issues in modeling and controlling, such as the distributed nature of these structures.

There are some ways that flexible structures and large systems can be modeled for control design, such as modal superposition, numerical integration and the transfer matrix method. Each of these methods has its benefits and drawbacks. In modal superposition, eigenvectors and eigenvalues of the system matrix should be computed which requires a lot of work. Also, modal superposition is based on the assumption of having proportional damping.

Numerical integration is another way that can be used to model a system. Wilson θ, Newmark Beta, Fox-Euler and other numerical methods are the basis of this method. Choosing among these methods depends on the accuracy, time steps and application. The challenging problem in numerical integration method happens when we have a large system with many degrees of freedom. It increases the size of matrices, so we need to do a lot of computation.

Another way to model these structures is the transfer matrix method.

1.1 Problem Statement

The transfer matrix method is one of the method that can model flexible structures and large systems easily. The size of matrices in the TMM is much lower than other methods and depends on the number of states ,but does not depend on the number of

elements. This method has some limitations. The transfer matrix method can model linear systems using a Laplace transform. Also, we just have frequency domain output in this method. By combining TMM and numerical integration method, we can have advantages of these two methods in one package. This method is called discrete time transfer matrix method (DT-TMM). DT-TMM has the benefits of numerical integration method which are ability to model nonlinear system with time-domain output and advantages of TMM which are low size of matrices and flexibility to add or eliminate subsystems easily.

The DT-TMM, like the numerical integration method, is sensitive to time steps and based on choosing an integration method to use and accuracy of the response changes.This research focus on the stability of the DT-TMM and how to increase the accuracy.

To reach this goal, two ways are investigated to add integration method to the TMM. In the first way, acceleration and velocity are rewritten based on the displacement. In the other step, displacement and velocity are rewritten based on the acceleration. For both of mentioned approaches, different numerical integration methods were studied.

1.2 Software Design

This research is done using a programming language called Python. Python is a very powerful and open source programming language which is simple and easy to learn. In this research, four Python codes have been written. First, an *ode* code for a system with arbitrary degrees of freedom was written. This code is used to compare with the output responses of the DT-TMM method. Second code was written for the DT-TMM method for one degree of freedom based on the displacement. The DT-TMM code based on displacement for arbitrary degrees of freedom was written in third step.Finally, by rewriting third code, new DT-TMM code for arbitrary degrees of freedom based on acceleration was achieved.

CHAPTER 2

LITERATURE REVIEW

Research on the transfer matrix method for flexible structures and large systems covers a lot of manipulators and robotic systems. Due to the benefits of the transfer matrix method (lower size of matrices, lower computation and flexibility to add or eliminate subsystems), researchers have been trying to find some ways that can overcome its disadvantages.

In Matrix Methods in Elastomechanics book [1], Pestel explains different matrix methods in elastomechanics. He introduces how to break up a large system into subsystems with simple properties. Then, he discusses different methods to extract a matrix from each subsystem. Pestel, tries to show the application of this method in real world and industry, where a huge system is made from a lot of small subsystems.

In modeling, design, and control of flexible manipulator arms Book [2] explains how to use transfer matrix method to model rigid robot links. Book considers four variables (displacement, shearing force, moment and angle) to describe his system. Book et al. [3, 4] show how to use the transfer matrix method to model and control a space shuttle manipulator.

Krauss [5] uses the TMM to model the closed-loop response of a system. In his work, a flexible robot with about five meters of arm is modeled based on the TMM. Also, a software is designed to do the TMM analysis and use the output to control the system. Actuator dynamics and its interaction were two challenging parts of modeling in his research. Lateral displacement, rotation, bending moment and shearing force were chosen to model the actuator. In designing a controller, he consider approaches like using Bode plots and pole-placement.

The TMM can be used accurately where actuators and sensors have exactly the same locations. However, in most of the cases, actuators and sensors do not have exactly the

same position. This situation can affect the stability of the system. Krauss[6] study how to use the TMM to model non-collocated systems.Kruass[7] explains how he uses Python as a programming language to model a flexible robot based on TMM.

Another method to model flexible and large systems is transfer dynamic stiffness matrix(TDSM). TDSM is a combination of dynamic stiffness and transfer matrix method. TDSM is frequency dependent. Yu et al.[8] use TDSM to analyse a space structure with Timoshenko beam theory and compared the results with FEM.

Tail shafts of ships which carry the propeller are flexible structures. Xiao-jun[9] could model this important part of a ship with the transfer matrix method. Genetic algorithm is the method she used to calculate natural frequency of the system.

Bin He et al[10] studied the natural vibration of a tree structure with TMM. A tree structure is exactly like a real tree in the environment. It has some nodes and branches. This structure is divided into two subsystems, chain subsystem, and branch subsystem. Chain subsystem has one input and one output, however branch subsystem has one output with one or more inputs. Transfer matrix for branch and chain subsystems are calculated separately and after that these two matrices combine together to reach the final transfer matrix.

Xiaoting Rui et al. [11] applied the TMM to model a hybrid linear multi-body system. This method is called Multi-Body System Transfer Matrix Method (MSTMM).Their method can be used to calculate eigenvalues,orthogonality of eigenvectors, dynamic response and connected parameters of multi-body systems. Because of lower order of the matrices in this method, the speed of computation is much faster than regular methods. Also, to confirm the application of this method, they used MSTMM in a practical case.

Krauss et al.[12] modeled a robot with flexible and rigid links by using the finite element method(FEM) and transfer matrix method(TMM). Flexibility of joints is a key factor that they considered in their research and by this consideration the output of the

TMM and FEM is fairly close to the experimental outputs.Guo and Wu [13] present TMM and FEM modeling of wave penetrating catamaran (WPC) to see which one gives better results on mode shape and natural frequency. They conclude that FEM is a better way for modeling of WPC compared with TMM.

Mihail et al.[14] use the TMM and Bessel's functions to model a variable cross-section beam(conical shape). Results show that combination of Bessel's functions and transfer matrix method gives more accurate answer compare with piecemeal TMM. Dokanish[15] applies transfer matrix method with FEM to study the vibration of plates. To reach this point, he divvied a plate to many strips with related mass and stiffness. State variables in his research were displacement and internal force.

Zu and Ji [16] tried to to improve the TMM for a nonlinear system(Rotor-Bearing System). To consider the effects of rotary inertia and shear deformation, Timoshenko beam theory is used to model a shaft. Bearings are modeled by linear damping and nonlinear springs. To solve non linearity of the bearing force, it is split in terms of space and time.

Ellakany et al.[17] combine the TMM and analogue beam method to study the vibration of a composite beam. In this study, the beam is divided into some subsystems. Each sub-beam is supposed to behave based on the Bernouli-Euler beam theory. Also, shear deformation is neglected in this research.

A lot of researchers try to improve the FEM method by incorporating the TMM. To analyze displacement of large systems Ohga et al.[18] use tangent stiffness to apply TMM. They show that this method works for long and thin structures. Bao Rong et al[19] expand TMM and FE to compute the eigenvalues of flexible structures. To confirm their method, they give some numerical examples.

Lee[20] introduce spectral TMM. Matrices in this method are derived from motion equations.Coupling lateral and torsional vibrations studied by Hsieha et al.[21] with the

TMM. Euler's angles show the orientation of the shaft and disk. Equations of motions are derived from Hamilton's Principle and Newton's second law. In this research, transfer matrices are calculated by the harmonic balance method.Horner and Pilkey[22] use Ricatti in the TMM to speed up the calculations to half of the TMM. Rotating shafts are analyzed by this method to confirm the applicability.

The Newton-Raphson method and the TMM were used by Huang and Horng [23] to study torsional vibration of branched structures.They demonstrate this method for a three free-end boundaries system.

Due to limitations of the TMM which are restriction on modeling non-linear systems and frequency domain outputs, researchers use the TMM method and numerical integration method at the same time and called it DT-TMM which means Discrete Time Transfer Matrix Method. Kumar and Sankar[24]use the DT-TMM to have the advantages of numerical integration and the TMM methods together. In this way, they reduced the matrix sizes and could model non-linear systems. Also, they could show their results in a time-domain system. In this research, acceleration and velocity were expressed based on displacement. It is mentioned in their research that this method is very sensitive to time steps, because of the sensitivity of the numerical integration method.

Xiaoting Rui et al[25] applied discrete time transfer matrix method to multi-body system(MS-DT-TMM) to model multi rigid and flexible systems. This method is a combination of TMM, DT-TMM, MS-TMM and numerical integration method. Also, X. Rui et al[26] extended MS-DT-TMM for multi-flexible systems to model multi-rigid ones. Moreover, FEM and MS-DT-TMM applied together to study multi-flexible systems.In addition, they use MS-DT-TMM, multi-body systems method and Ricattit TMM to extend modeling of multi-systems.

B. Rong et al.[27] expand the discrete time transfer matrix method to control a controlled multi-body system. Flexibility and lower size of matrices are two parameters

that speed up the calculations compared with other conventional methods. They design state vectors and transfer matrices for an actuator, controlled elements, and feedback elements.They compared the result of their method with conventional dynamics method and the result was the same. In designing the controller, velocity was considered to use in the feedback controller.

He,Wang and Rui[28] studied how using Ricatti in the DT-TMM can increase the stability of the system. The DT-TMM and rotational springs helps to have dynamic equations in this research. A beam considered as a lot of rigid bodies which are connected together by rotational springs. k for the spring is calculated by FE. Due to numerical error, one corrector method was used in this study. This corrector computes the transfer matrix of elements and related Ricatti matrices and compares all the state vectors of each node. Corrector continues this operation until requested accuracy is reached.

CHAPTER 3

Introduction to State Vector and Transfer Matrix Method

Since the TMM method is the base of Discrete Time Transfer Matrix Method, in this chapter TMM is explained in details for a simple two degree of freedom system.

3.1 State Vector

Sate vector at a point of an elastic system is a column vector which indicates the displacement of the point and its internal force. The displacement in the upper and the force in lower half of the column should be written.

For example in a simple mass and spring system the state vector is $\begin{bmatrix} x \\ f \end{bmatrix}$ where

x: shows displacement

f: shows internal force

In a mass less shaft with disks displacement represented as the angle of twist (ϕ) and force is the related torque (T). State vector for this system is $\begin{bmatrix} \phi \\ T \end{bmatrix}$.

To have more accurate answer in a straight beam, displacement's factors are deflection (w) and slope (ψ). Also, force's factors are moment (M) and shear force (V). This state vector can be given by $\begin{bmatrix} w \\ \psi \\ M \\ V \end{bmatrix}$.

3.2 Transfer Matrix Method

After knowing state vectors for some usual systems in mechanics, let transfer matrix method be introduced. The transfer matrix method which is usually called TMM is a method which breaks a system into the number of components.

Figure 3.1: is a simple one degree of freedom mass-damper system which is exploded in

Figure 3.2: and as can be seen all internal forces and displacements described in details.

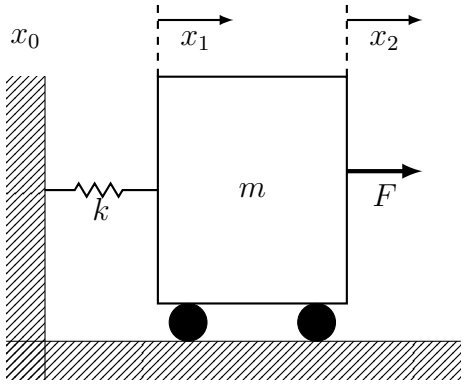


Figure 3.1: One degree of freedom mass-spring system

Exploded of above system can be seen in figure 3.2.

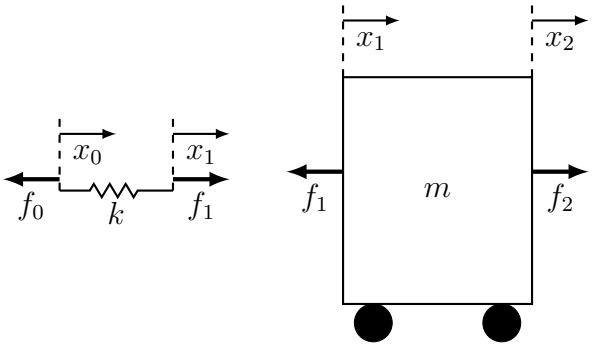


Figure 3.2 : Exploded view of the one degree of freedom mass-spring system

As the spring is mass less,

$$f_0 = f_1 \tag{3.1}$$

$$f_0 = k(x_1 - x_0) \tag{3.2}$$

$$x_1 = \frac{f_0}{k} + x_0 \tag{3.3}$$

(3.1) And (3.3) can be written in a matrix form:

$$\begin{bmatrix} x_1 \\ f_1 \end{bmatrix} = \begin{bmatrix} 1 & \frac{1}{k} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ f_0 \end{bmatrix} \tag{3.4}$$

$$U_s = \begin{bmatrix} 1 & \frac{1}{k} \\ 0 & 1 \end{bmatrix} \tag{3.5}$$

U_s is the transfer matrix of the spring which transfer state vector from point 0 to point1.

If the mass considered as a rigid body, for second part of our system can be written:

$$x_1 = x_2 \tag{3.6}$$

$$f_2 - f_1 = m\ddot{x_1} \tag{3.7}$$

By using Laplace transform

$$f_2 - f_1 = ms^2 x_1 \tag{3.8}$$

$$f_2 = f_1 + ms^2 x_1 \tag{3.9}$$

(3.5) And (3.9) can be written in a matrix form:

$$\begin{bmatrix} x_2 \\ f_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ ms^2 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ f_1 \end{bmatrix} \tag{3.10}$$

$$U_m = \begin{bmatrix} 1 & 0 \\ ms^2 & 1 \end{bmatrix} \tag{3.11}$$

U_m is transfer matrix of mass .

Plugging in x_1 and f_1 from equation 3.4 gives

$$\begin{bmatrix} x_2 \\ f_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ ms^2 & 1 \end{bmatrix} \begin{bmatrix} 1 & \frac{1}{k} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ f_0 \end{bmatrix} \tag{3.12}$$

$$\begin{bmatrix} x_2 \\ f_2 \end{bmatrix} = \begin{bmatrix} 1 & \frac{1}{k} \\ ms^2 & \frac{ms^2}{k} + 1 \end{bmatrix} \begin{bmatrix} x_0 \\ f_0 \end{bmatrix} \tag{3.13}$$

In a free response system

$$x_0 = 0 \tag{3.14}$$

$$f_2 = 0 \tag{3.15}$$

The above equation yield the results

$$\begin{bmatrix} x_2 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 & \frac{1}{k} \\ ms^2 & \frac{ms^2}{k} + 1 \end{bmatrix} \begin{bmatrix} 0 \\ f_0 \end{bmatrix} \tag{3.16}$$

The first row gives

$$x_2 = \frac{f_0}{k} \tag{3.17}$$

The second row gives

$$(\frac{ms^2}{k}+1)f_0 = 0 \tag{3.18}$$

Non-trivial solution for equation 15 gives

$$(\frac{ms^2}{k}+1) = 0 \tag{3.19}$$

$$s = \pm j\sqrt{\frac{k}{m}} \tag{3.20}$$

The imaginary part of the solution is natural frequency.

In a forced response system which f_2= F$\neq 0$ *and* $x_0 = 0$ equation 3.13 can be written

$$\begin{bmatrix} x_2 \\ F \end{bmatrix} = \begin{bmatrix} 1 & \frac{1}{k} \\ ms^2 & \frac{ms^2}{k}+1 \end{bmatrix} \begin{bmatrix} 0 \\ f_0 \end{bmatrix} \tag{3.21}$$

The first row gives

$$x_2 = \frac{f_0}{k} \tag{3.22}$$

The second row gives

$$F = \frac{ms^2}{k} f_0 \tag{3.23}$$

If Equation 23 plugs in equation 22, transfer function for the system can be written

$$\frac{x_2}{F} = \frac{1}{ms^2+k} \tag{3.24}$$

Figure 3.3: shows a two-degree of freedom mass-spring-damper system with internal forces

However, Laplace transform is much easier way to reach the transfer function for this system, but in complicated systems this method will help a lot.

To be more familiar with TMM, a forced two degree of freedom system will be presented in below.

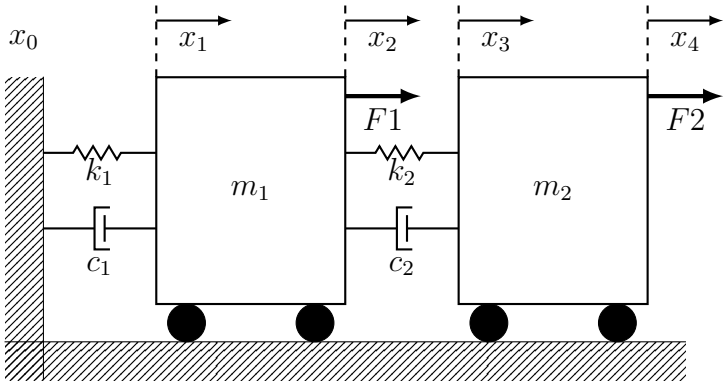


Figure 3.3: Two degree of freedom mass-spring-damper system

Exploded of the above system can be seen in figure 3.4.

As the spring s_1 and the damper d_1 are mass less

$$f_0 = f_1 \tag{3.25}$$

$$f_1 = k_1(x_1 - x_0) + c_1(\dot{x_1} - \dot{x_0}) \tag{3.26}$$

By using Laplace transform

$$f_1 = k_1(x_1 - x_0) + c_1 s(x_1 - x_0) \tag{3.27}$$

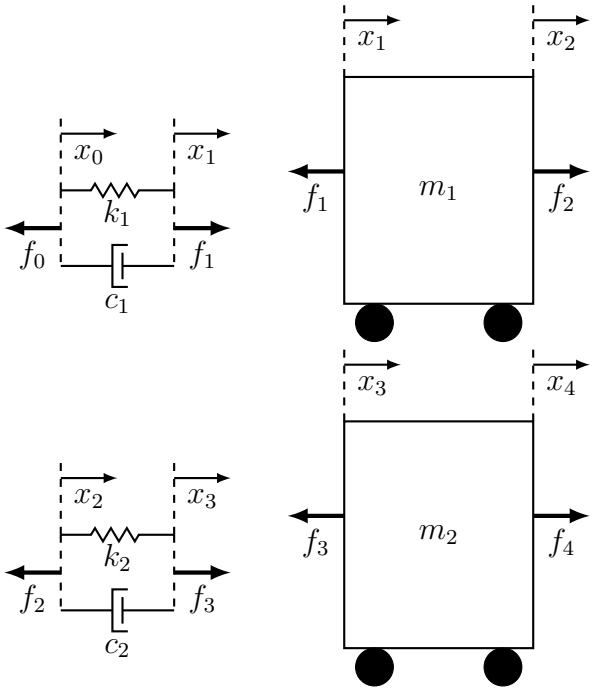


Figure 3.4: Exploded view of the two degree of freedom mass-spring-damper system

If equation 3.27 solves for x_1

$$x_1 = x_0 + \frac{f_1}{k_1 + c_1 s} \tag{3.28}$$

Equation 3.25 and 3.28 gives

$$\begin{bmatrix} x_1 \\ f_1 \end{bmatrix} = \begin{bmatrix} 1 & \frac{1}{k_1+c_1 s} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ f_0 \end{bmatrix} \tag{3.29}$$

$$U_{s1}d_1 = \begin{bmatrix} 1 & \frac{1}{k_1+c_1 s} \\ 0 & 1 \end{bmatrix} \tag{3.30}$$

$U_{s1}d_1$ is the transfer matrix of spring s_1 and damper d_1 . For the next component since the mass m_1 is rigid

$$x_1 = x_2 \tag{3.31}$$

$$f_2 - f_1 = m_1 \ddot{x}_1 \tag{3.32}$$

By using Laplace transform

$$f_2 - f_1 = m_1 s^2 x_1 \tag{3.33}$$

$$f_2 = f_1 + m_1 s^2 x_1 \tag{3.34}$$

Equation 3.31 and 3.34 can be written

$$\begin{bmatrix} x_2 \\ f_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ m_1 s^2 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ f_1 \end{bmatrix} \tag{3.35}$$

$$U_{m1} = \begin{bmatrix} 1 & 0 \\ m_1 s^2 & 1 \end{bmatrix} \tag{3.36}$$

U_{m1} is transfer matrix of mass m_1 .

When external force does not apply on the last element, augmented transfer matrices can play the role of injecting forces, moments, or displacements into the TMM matrices. It means one extra column and row should be added to the transfer matrix. All added elements are zero except the lower one on the diagonal which is 1.

An augmented mass transfer matrix will take the form

$$U_m = \begin{bmatrix} 1 & 0 & 0 \\ ms^2 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.37}$$

An augmented spring/damper transfer matrix will be

$$U_s d = \begin{bmatrix} 1 & \frac{1}{k+cs} & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.38}$$

An augmented forcing transfer matrix will be

$$U_F = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -F \\ 0 & 0 & 1 \end{bmatrix} \tag{3.39}$$

When using augmented matrices, state vector should be in below form

$$Z = \begin{bmatrix} x \\ f \\ 1 \end{bmatrix} \tag{3.40}$$

Now augmented matrices can be written for all components. Augmented spring/damper transfer matrix for $spring/damper1$

$$U_{s1} d_1 = \begin{bmatrix} 1 & \frac{1}{k_1+c_1 s} & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.41}$$

Augmented mass transfer matrix for m_1

$$U_{m1} = \begin{bmatrix} 1 & 0 & 0 \\ m_1 s^2 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.42}$$

Augmented forcing transfer matrix for F_1

$$U_{F1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -F_1 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.43}$$

Augmented spring/damper transfer matrix for $spring/damper2$

$$U_{s2d_2} = \begin{bmatrix} 1 & \frac{1}{k_2+c_2 s} & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.44}$$

Augmented mass transfer matrix for m_2

$$U_{m2} = \begin{bmatrix} 1 & 0 & 0 \\ m_2 s^2 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.45}$$

Now, transfer matrix of the system can be achieved by multiplying all transfer matrices. The order of multiplication is from end to first. If F_2 =0 then

$$U_{sys} = U_{m2} U_{s2d2} U_{F1} U_{m1} U_{s1d1} \tag{3.46}$$

State vector of base is

$$Z_{base} = \begin{bmatrix} x_{base} \\ f_{base} \\ 1 \end{bmatrix} \tag{3.47}$$

State vector of end is

$$Z_{end} = \begin{bmatrix} x_{end} \\ f_{end} \\ 1 \end{bmatrix} \tag{3.48}$$

By applying boundary conditions which are

$$F_end = 0 \tag{3.49}$$

$$x_{base} = 0 \tag{3.50}$$

Base and end state vectors are

$$Z_{base} = \begin{bmatrix} 0 \\ f_{base} \\ 1 \end{bmatrix} \tag{3.51}$$

$$Z_{end} = \begin{bmatrix} x_{end} \\ 0 \\ 1 \end{bmatrix} \tag{3.52}$$

By having U_{sys}

$$Z_{end} = U_{sys} Z_{base} \tag{3.53}$$

U_{sys} Can be written in this form

$$U_{sys} = \begin{bmatrix} U_{11} & U_{12} & U_{13} \\ U_{21} & U_{22} & U_{23} \\ U_{31} & U_{32} & U_{33} \end{bmatrix} \tag{3.54}$$

Plugging in Z_{end} ,Z_{base} and U_{sys} in equation 3.53

$$\begin{bmatrix} x_{end} \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} U_{11} & U_{12} & U_{13} \\ U_{21} & U_{22} & U_{23} \\ U_{31} & U_{32} & U_{33} \end{bmatrix} \begin{bmatrix} 0 \\ f_{base} \\ 1 \end{bmatrix} \tag{3.55}$$

x_{end} And f_{base} are unknown From line 2 of equation 3.55

$$U_{22} f_{base} + U_{23} = 0 \tag{3.56}$$

$$f_{base} = \frac{-U_{23}}{U_{22}} \tag{3.57}$$

State vector of base is

$$Z_{base} = \begin{bmatrix} 0 \\ \frac{-U_{23}}{U_{22}} \\ 1 \end{bmatrix} \tag{3.58}$$

By having Z_{base} , state vector of every point in the system can be obtained. For instance, state vector for point 2 is

$$Z_2 = Z_{m1} = U_{m1}U_{s1d1}Z_{base} \tag{3.59}$$

Also, state vector of point 3 is

$$Z_3 = U_{s2d2}U_{m1}U_{s1d1}Z_{base} \tag{3.60}$$

Equations of 3.59 and 3.53 can be used to obtain transfer function x_1/F and x_2/F. From equation 3.59

$$\begin{bmatrix} x_1 \\ f_1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ m_1 s^2 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & \frac{1}{k_1+c_1 s} & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ \frac{-U_{23}}{U_{22}} \\ 1 \end{bmatrix} \tag{3.61}$$

From equation 3.61,$\frac{x_1}{F}$ is equal

$$\frac{x_1}{F} = \frac{m_2 s^2 + c_2 s + k_2}{P} \tag{3.62}$$

From Equation 3.53, $\frac{x_2}{F}$ is equal

$$\frac{x_2}{F} = \frac{c_2 F s + F k_2}{P} \tag{3.63}$$

where P is

$$(m_1 m_2)s^4 + (c_2 m_1 + c_2 m_2 + c_1 m_2)s^3 + (k_2 m_2 + k_1 m_2 + k_2 m_1 + c_1 c_2)s^2 + (k_2 c_1 + k_1 c_2)s + k_1 k_2 \tag{3.64}$$

For systems with low degree of freedom using TMM method is not reasonable, however when the degree of freedom increases TMM can be much easier to model a system and reach to transfer matrices.

As mentioned before, TMM has some limitations. TMM just can model linear systems and its output is in frequency domain. Numerical integration comes to help us to overcome these limitations. Combination of TMM and numerical integration method can solve these issues. Thus, Modeling non-linear systems and time domain output are reachable by applying DT-TMM method.DT-TMM will be introduce in the next chapter and its output will be compared with newton method.

CHAPTER 4

Introduction to Discrete Time Transfer Matrix Method(DT-TMM)

4.1 Why DT-TMM

As explained before in introduction and literature review, Because of the Transfer Matrix Method limitations, researchers have been trying to modify TMM. TMM just can model linear systems due to using Laplace transform. Also, its output is just frequency domain. One the most effective method that engineers work on that is combining TMM and numerical integration methods which is called discrete time transfer matrix method(DT-TMM). This combination has the benefits of both methods. The order of matrices in DT-TMM is like TMM which lead to lower computation and faster response. Moreover, DT-TMM can model non-linear systems and has the ability to have time-domain output due to using numerical integration method. Unfortunately, DT-TMM inherits the drawback of numerical engineering which is sensitivity to time steps. Time steps should be chosen in a way that have the lowest effects on the output. In this chapter one degree of freedom and two degree of freedom systems will be explained in details by this method.

4.2 DT-TMM for One Degree of Freedom System

State vector in DT-TMM method is exactly like TMM. In the following examples the state vector is based on the displacement and internal force. One degree of freedom system which contain mass, spring and damper can be seen in figure 4.1. It is divided to 3 parts which are

1-spring-damper

2-mass

3-force

also, there are 3 nodes of x_0, x_1, x_2

Kumar and Sankar [24] to combine the TMM method with numerical integration method considered the velocity and acceleration in terms of displacement. In this way there is no need to use Laplace transform which limit the TMM to linear systems.

Equations that Kumar and Sankar [24] used are in below

$$\ddot{x_n}(t_i) = A_n(t_i)x_n(t_i) + B_n(t_i) \tag{4.1}$$

$$\dot{x_n}(t_i) = D_n(t_i)x_n(t_i) + E_n(t_i) \tag{4.2}$$

These equations are linear based on the displacement.

As mentioned before in TMM, a system divided to some subsystems. Each subsystem can be modeled by a mass-spring-damper system. Each subsystem is called station. Subscript n is related to the location of the part and denote the number of subsystem. For example, In figure 4.1, we just have one station (one degree of freedom), so in our calculation n is equal one. To be more clear, in a two degree of freedom system n will be equal one and two. Subscript i show the related time step. For example, if we want to show the response of the system during 1 second and time step is 0.1 second. Then, i can be from one to ten. Moreover, n_{th} station is composed of m_n, k_n and c_n. $\dot{x_n}$ and $\ddot{x_n}$ show the velocity and acceleration of m_n .

A_n, B_n, D_n and E_n are the coefficients of equations 4.1 and 4.2 which should be calculated in each time step of DT-TMM. Depending on the numerical integration method, these coefficients are different.

A_n is the coefficient of x_n in equation 4.1 which is proportional to the square of the time step inversely and is constant in each time step. For example A_n for Newmark β method is

$$A_n = \frac{1}{\beta \Delta T^2} \tag{4.3}$$

Depending on the numerical integration method B_n in equation 4.1 is a function of the different parameters like displacement, velocity and acceleration in the same step time or step times before. Also, it can be proportional to the square of the time step inversely. For example B_n for Newmark β method is

$$B_n = \frac{-1}{\beta \Delta T^2}[x(t_{i-1}) + \Delta T \dot{x}(t_{i-1}) + (0.5 - \beta)\Delta T^2 \ddot{x}(t_{i-1})] \tag{4.4}$$

D_n is the coefficient of x_n in equation 4.2 which is proportional to the time step inversely and is constant in each time step. For example D_n for Newmark β method is

$$A_n = \frac{\gamma}{\beta \Delta T} \tag{4.5}$$

Depending on the numerical integration method E_n in equation 4.2 is a function of the different parameters like displacement, velocity and acceleration in the same step time or step times before.For example E_n for Newmark β method is

$$E_n = \dot{x}(t_{i-1}) + \Delta T[(1 - \gamma)\ddot{x}(t_{i-1}) + (\gamma \beta_n)] \tag{4.6}$$

β and γ are constant numbers and depend on the accuracy and change of the acceleration in each time step. Foe example, if the acceleration change linearly in each time step, $\beta = \frac{1}{6}$ and $\gamma = \frac{1}{2}$ They will be explained latter in details.

DT-TMM for one degree of freedom system will be explained in details. Complete and exploded one degree of freedom mass-spring-damper system can be seen in next page figures.

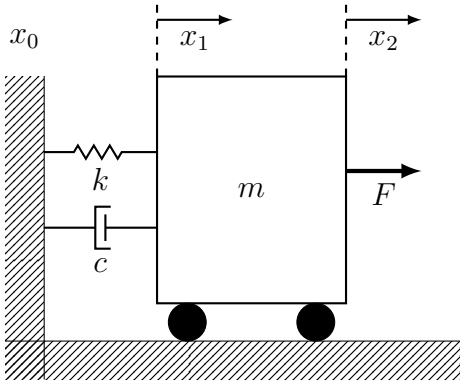


Figure 4.1: One degree of freedom mass-spring-damper system

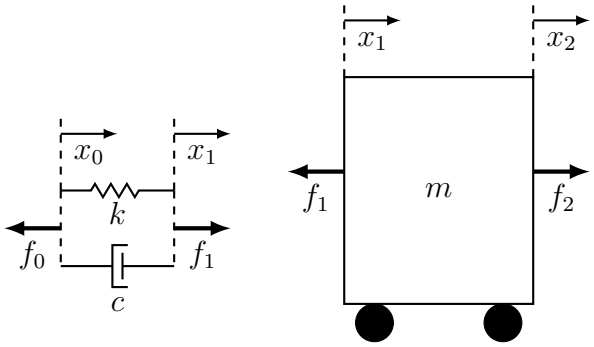


Figure 4.2: Exploded view of one degree of freedom mass-spring-damper system

As the spring-damper is mass less

$$f_0 = f1 \tag{4.7}$$

force on the mass-spring is

$$f_0 = k(x_1 - x_0) + c(\dot{x_1} - \dot{x_0}) \tag{4.8}$$

And force equation for the mass is

$$f_2 - f_1 = m\ddot{x_1} \tag{4.9}$$

Since, the mass is rigid

$$x_2 = x_1 \tag{4.10}$$

To combine numerical integration and TMM methods, equation 4.1 and 4.2 need to be plugged in equations 4.8 and 4.9.

By plugging in 4.2 in 4.8

$$f_0 = f_1 = k(x_1(t_i) - x_0(t_i)) + c[(D_1(t_i)x_1(t_i) + E_1(t_i)) - (D_0(t_i)x_0(t_i) + E_0(t_i))] \tag{4.11}$$

Equation 4.11 can be written for x_1

$$x_1 = \frac{(cD_0 + k)x_0}{(cD_1 + k)} + \frac{f_0}{(cD_1 + k)} + \frac{c(E_0 - E_1)}{(cD_1 + k)} \tag{4.12}$$

Equations of 4.12 and 4.7 can be written in matrix form

$$\begin{bmatrix} x_1 \\ f_1 \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{(cD_0+k)}{(cD_1+k)} & \frac{1}{(cD_1+k)} & \frac{c(E_0-E_1)}{(cD_1+k)} \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ f_0 \\ 1 \end{bmatrix} \tag{4.13}$$

U_{sd} is DT-TMM transfer matrix for spring-damper which transfer state vectors from point 0 to point 1.

$$U_{sd} = \begin{bmatrix} \frac{(cD_0+k)}{(cD_1+k)} & \frac{1}{(cD_1+k)} & \frac{c(E_0-E_1)}{(cD_1+k)} \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.14}$$

In the mentioned one degree of freedom system, one station just exist, so D_0 and E_0 are equal zero.

Equation 4.1 can be substitute in equation 4.9 to achieve DT-TMM transfer matrix for mass m

$$f_2 - f_1 = m[A_1(t_i)x_n(t_i) + B_1(t_i)] \tag{4.15}$$

This equation can be rewrite for f_2

$$f_2 = f_1 + mA_1(t_i)x_n(t_i) + mB_1(t_i) \tag{4.16}$$

Equations of 4.16 and 4.10 can be written in matrix form

$$\begin{bmatrix} x_2 \\ f_2 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ mA_1 & 1 & mB_1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ f_1 \\ 1 \end{bmatrix} \tag{4.17}$$

U_m is DT-TMM transfer matrix for mass which transfer state vectors from point 1 to point 2.

$$U_m = \begin{bmatrix} 1 & 0 & 0 \\ mA_1 & 1 & mB_1 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.18}$$

Matrix U_f for DT-TMM is same as TMM.

$$U_f = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -F \\ 0 & 0 & 1 \end{bmatrix} \tag{4.19}$$

Now by having transfer matrices for spring-damper, mass and force system transfer matrix can be achieved. The method of multiplication of the matrices is exactly same as TMM method and it starts from end point to start point.

$$U_{sys} = U_f U_m U_{sd} \tag{4.20}$$

State vector of base is

$$Z_{base} = \begin{bmatrix} x_{base} \\ f_{base} \\ 1 \end{bmatrix} \tag{4.21}$$

State vector of end is

$$Z_{end} = \begin{bmatrix} x_{end} \\ f_{end} \\ 1 \end{bmatrix} \tag{4.22}$$

By applying boundary conditions which are

$$F_{end} = 0 \tag{4.23}$$

$$x_{base} = 0 \tag{4.24}$$

Base and end state vectors are

$$Z_{base} = \begin{bmatrix} 0 \\ f_{base} \\ 1 \end{bmatrix} \tag{4.25}$$

$$Z_{end} = \begin{bmatrix} x_{end} \\ 0 \\ 1 \end{bmatrix} \tag{4.26}$$

By having U_{sys}

$$Z_{end} = U_{sys} Z_{base} \tag{4.27}$$

U_{sys} Can be written in this form

$$U_{sys} = \begin{bmatrix} U_{11} & U_{12} & U_{13} \\ U_{21} & U_{22} & U_{23} \\ U_{31} & U_{32} & U_{33} \end{bmatrix} \tag{4.28}$$

Plugging in Z_{end} ,Z_{base} and U_{sys} in equation 4.27

$$\begin{bmatrix} x_{end} \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} U_{11} & U_{12} & U_{13} \\ U_{21} & U_{22} & U_{23} \\ U_{31} & U_{32} & U_{33} \end{bmatrix} \begin{bmatrix} 0 \\ f_{base} \\ 1 \end{bmatrix} \tag{4.29}$$

x_{end} And f_{base} are unknown From line 2 of equation 4.29

$$U_{22} f_{base} + U_{23} = 0 \tag{4.30}$$

$$f_{base} = \frac{-U_{23}}{U_{22}} \tag{4.31}$$

State vector of base is

$$Z_{base} = \begin{bmatrix} 0 \\ \frac{-U_{23}}{U_{22}} \\ 1 \end{bmatrix} \tag{4.32}$$

By having Z_{base} , state vector of every point in the system can be obtained. For instance, state vector for point 2 is

$$Z_2 = Z_m = U_f U_m U_{sd} Z_{base} \tag{4.33}$$

Also, state vector of point 1 is

$$Z_1 = U_{sd} Z_{base} \tag{4.34}$$

Now by having the initial conditions (x_0 and $\dot{x_0}$), we can have the displacement of the system during the time and have a time-domain output.

4.3 Pyhton Code for One Degree of Freedom DT-TMM

There are many programming languages that can be used to write DT-TMM code. Python is the chosen language to write this code due its powerful packages and being open-source. Python code for one degree of freedom system with mass, spring and damper is explained in details below.

```
from scipy import *
import numpy

"""""DOF: Degree of Freedom(one mass, one damper, one spring
   Usd: transfer matrix of spring / damper
   Um: Transfer matrix of mass
   Uf: Transfer matrix of applied force
   Usys: Transfer matrix of the system   """""
Usd=numpy.zeros((3,3))
Um=numpy.zeros((3,3))
Uf=numpy.zeros((3,3))
Usys=numpy.zeros((3,3))
```

Numpy is a powerful package for scientific calculation. It can prepare multi-dimensional arrays for different purposes. To use this ability of Numpy, it is imported at the begining of the program. Numpy.zeros creates 3 by 3 matrices for transfer matrices of mass, spring-damper, force and system. It should be mentioned that location of items in python matrices start from zero. It means, for example $U_m(0,0)$ is the content of the cell which is located in first row and column.

```
"""""""""Constatnt coefficients of Newmark Beta method"""""""""
beta = 1.0 / 6.0
gamma = 0.5

"""""""""m: mass,  k: spring constant,  c: damping coefficient, f: force"""""""""

m=array([2.0])
k=array([12.0])
c=array([3.0])
f=array([1.0])

"""""""""T: time, dt: time step , N: numebr of steps """""""""
T=10
dt=.02
N = int(T/dt)

"""""""""defining arrays of displacement, velocity and acceleration   """""""""
x=zeros(N)
xdot=zeros(N)
xddot=zeros(N)
```

New-mark Beta is the integration method which is used here. Based on the explanation which will be discussed later, β=1/6 and $\gamma = 0.5$ considered. In the next step, mass(kg), spring constant(N/m), damping coefficient ($N - Sec/m$) and force(N) as the parameters of the system imported. T is the duration that needs to be studied. dt is time step and is one of the most important parameters that should be selected carefully to have the stable response. By dividing T by dt, the number of steps achieved that is demonstrated by N.

As there are N steps to have response of the system during T, Zeros command creates three N zero arrays function for displacement, velocity and acceleration. Python start from zero to count items in arrays and matrices. Initial condition to investigate the system is initial displacement and velocity. In this example, initial conditions are zero. It means

$$x_0 = 0 \tag{4.35}$$

$$\dot{x_0} = 0 \tag{4.36}$$

As zeros command used in this code, it means x_0 and $\dot{x}_0$ are equal zero and there is no need to enter initial conditions. However, if initial conditions were not zero, after definition of displacement and velocity by using zeros command, initial displacement $x(0,0)$ and velocity $\dot{x}(0,0)$ should be defined separately.

```
""""""""" DT-TMM  """""""""
for p in range(1,N):
```

In this step, *for* command is used to have N steps. For each step below stages need to be done receptively.

1- A, B, D and E coefficients should be calculated.

2-Then, transfer matrices (U_f, U_{sd}, U_m) can be achieved by using coefficients of the first step.

3- Transfer matrix for the whole system (U_{sys}) computed by multiplying mentioned transfer matrices of step2.

4- Internal force of the base (f_{base}) can be reached by dividing $-U_{sys}(1,2)$ by $U_{sys}(1,1)$.

5- Displacement of the mass (x_{end}) should be computed.

6- By having displacement, velocity and acceleration of the mass (x_{end}) can be calculated.

When displacement, Velocity and acceleration of the (x_{end}) achieved for the first step time, following mentioned step leads to have displacement, velocity and acceleration for the next step time, and this loop can be continued till reaching step time N

```
""" Calculation of A,B,D and E coefficients for each time step"""""
A = 1.0/(beta*dt**2)
B = -1.0/(beta*dt**2)*(x[p-1] + dt*xdot[p-1] + (0.5-beta)*dt**2*xddot[p-1])
E = xdot[p-1] + dt*((1.0-gamma)*xddot[p-1]+gamma*B)
D = gamma/(beta*dt)
```

Above code shows how to calculate A, B, D and E coefficients. In this code, Newmark Beta method used. To have these coefficients, displacement, velocity and acceleration of

the step before are needed.

```
"""Calculation of Um for each time step"""""

 for i in range(3):
    for j in range(3):
       if i==j:
        Um[i][j]=1
    Um[1][0]=m[0]*A
    Um[1][2]=m[0]*B
```

Here, two for loops and one if command are used to calculate transfer matrix of mass (U_m).

```
"""""Calciulation of Usd for each time step"""""
for i in range(1,3):
   for j in range(1,3):
      if i==j:
        Usd[i][j]=1
   Usd[0][0]=(k[0]/(k[0]+c[0]*D))
   Usd[0][1]=(1.0/(k[0]+c[0]*D))
   Usd[0][2]=(-c[0]*(E)/(k[0]+c[0]*D))
```

In this step, D and E helped to have transfer matrix for spring-damper (U_{sd}). Compared with 4.13 ,D_0 and E_0 are zero, because there is a one degree of freedom system which means we just have one station.

```
"""Calculation of Uf for each time step"""""
for i in range(3):
    for j in range(3):
        if i==j :
           Uf[i][j]=1.0
    Uf[1][2]=-f[0]
```

Mentioned code in above, creates transfer matrix for force (U_f).

```
"""""""Calculation of Usys and Fbase for each time step"""
v=dot(Uf,Um)
Usys=dot(v,Usd)
Fbase= - Usys[1][2]/Usys[1][1]
```

As mentioned before, U_{sys} can be computed by multiplying transfer matrices from end of the system to starting point. Matrix V keep the values of multiplication of U_f by U_m, and then U_{sys} computed by multiplying V by U_{sd}. Then, f_{base} calculated by dividing $-U_{sys}(1,2)$ by $U_{sys}(1,1)$.

```
""" Calculation of x, xdot and xddot for each time step"""""
x[p] =  (Usys[0][1]*Fbase) + Usys[0][2]
xdot[p] = D*x[p]+E
xddot[p]=A*x[p]+B
```

Final step in loop *for* is computation of displacement, velocity and acceleration. Displacement x_{end} can be achieved by using the first row of matrix 4.29. Velocity and acceleration can be calculated by using equations 4.2 and 4.1 respectively.

```
""" Plotting the graph"""""
figure(1)
clf()
t = arange(0.0, T, dt)
plot(t,x)
plt.ylabel('x')
plt.xlabel('time')
show()
```

Final part of the code is drawing the displacement versus time graph.

In below, the step response of the system by DT-TMM method which is shown in figure 4.1 can be seen in figure 4.3. Mass is two (kg), spring constant is twelve (N/m) and damping coefficient is three $(N - Sec/m)$.

To compare the DT-TMM method with real output, another code is written which show the response of the n degree of freedom system based on the newton law. This code will be explained in the next chapter briefly.

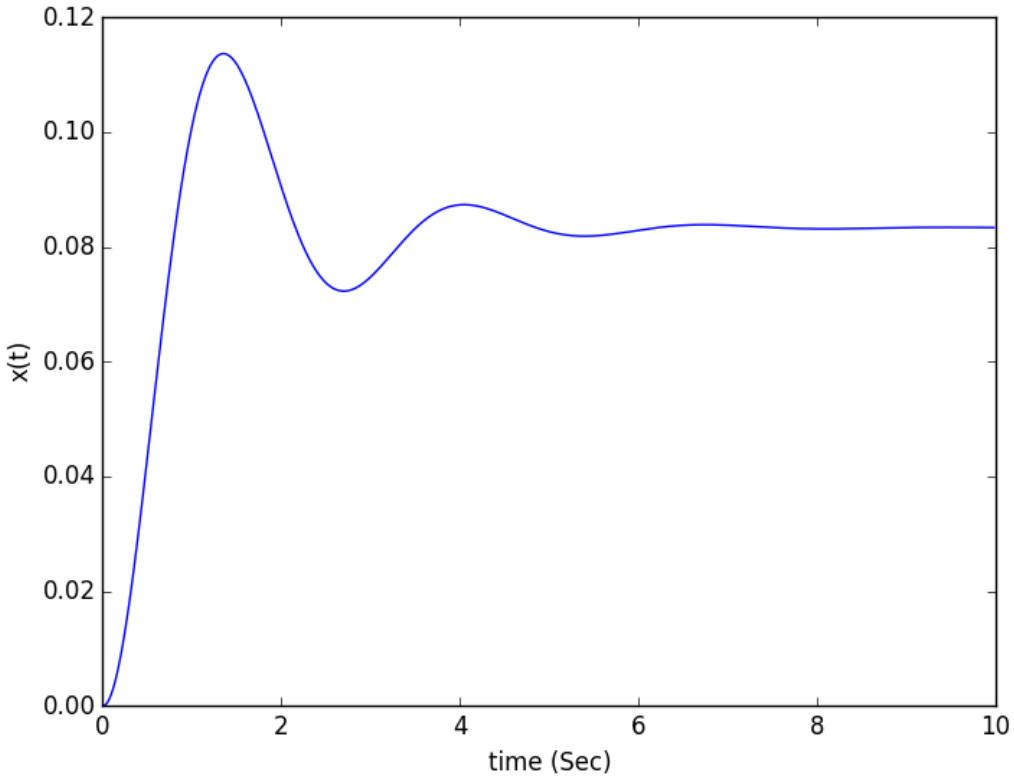


Figure 4.3: Step response of one degree of freedom mass-spring-damper system (DT-TMM)

CHAPTER 5

ODE Integration of n Degree of Freedom System

To compare the result of the DT-TMM method, a Pyhton code is written for a system with n degree of freedom (figure5.1) based on the ODE integration which will be explained in details below. For this issue, *odeint* command used which a powerful command to solve ordinary differential equation. Time step in *odeint* can be adjusted as the program go along, which lead to much more accurate answer.

Inputs of this code are:

1- Degree of freedom of the system

2-Initial conditions (velocity $\dot{x}$ and displacement x of the all masses)

3- Parameters of the system (masses, spring constants, damping coefficients and forces)

Output is response of the elements versus time.

Just to make the used procedure here clear, a two degree of freedom (figure5.2) modeling is explaining.

Using the second newton's law for the m_0 gives:

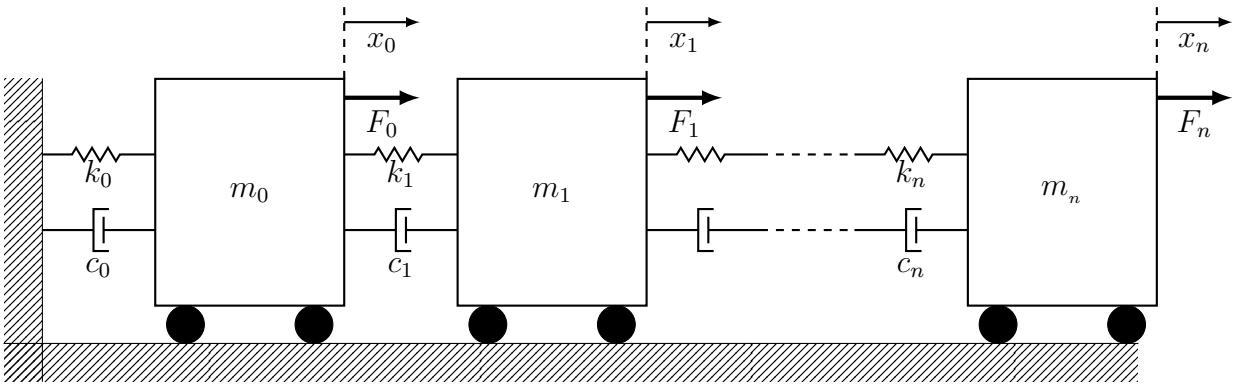


Figure 5.1: $n+1$ degree of freedom mass-spring-damper system

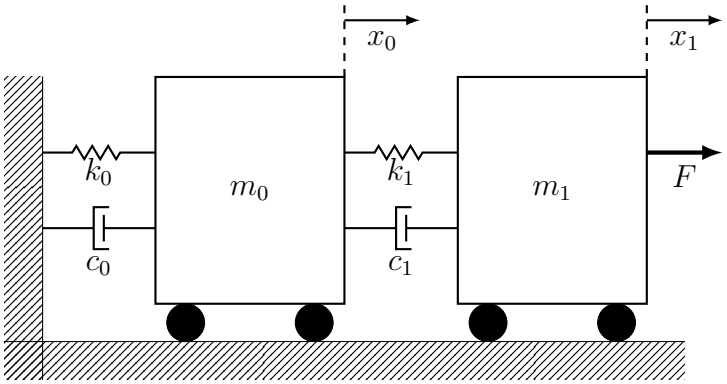


Figure 5.2: Two degree of freedom mass-spring-damper system

$$m_0\ddot{x_0} + c_0\dot{x_0} + k_0x_0 + c_1(\dot{x_0} - \dot{x_1}) + k_1(x_0 - x_1) = 0 \tag{5.1}$$

For the second mass (m_1) can be written

$$m_1\ddot{x_1} + c_1(\dot{x_1} - \dot{x_0}) + k_1(x_1 - x_0) = F \tag{5.2}$$

Equations 5.1 and 5.2 can be rewrite in matrix form and is called matrix equation of motion.

$$\begin{bmatrix} m_0 & 0 \\ 0 & m_1 \end{bmatrix} \begin{bmatrix} \ddot{x_0} \\ \ddot{x_1} \end{bmatrix} + \begin{bmatrix} c_0 + c_1 & -c_1 \\ -c_1 & c_1 \end{bmatrix} \begin{bmatrix} \dot{x_0} \\ \dot{x_1} \end{bmatrix} + \begin{bmatrix} k_0 + k_1 & -k_1 \\ -k_1 & k_1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} = \begin{bmatrix} 0 \\ F \end{bmatrix} \tag{5.3}$$

To use *odeint* command in Python, we should have a linear format like $\dot{X} = AX + B$ which can be achieved By changing the variables

$$\dot{x_0} = x_2 \tag{5.4}$$

$$\dot{x_1} = x_3 \tag{5.5}$$

New state spaces form are

$$\dot{X} = \begin{bmatrix} \dot{x_0} \\ \dot{x_1} \\ \dot{x_2} \\ \dot{x_3} \end{bmatrix} \tag{5.6}$$

$$X = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \tag{5.7}$$

Matrix equation 5.3 and equations 5.4 and 5.5 can be rewrite in below form

$$\begin{bmatrix} \dot{x_0} \\ \dot{x_1} \\ \dot{x_2} \\ \dot{x_3} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ \frac{k_0+k_1}{m_0} & \frac{-k_1}{m_0} & \frac{c_0+c_1}{m_0} & \frac{-c_1}{m_0} \\ \frac{-k_1}{m_1} & k_1 m_1 & -c_1 m_1 & c_1 m_1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ \frac{F}{m_1} \end{bmatrix} \tag{5.8}$$

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ \frac{k_0+k_1}{m_0} & \frac{-k_1}{m_0} & \frac{c_0+c_1}{m_0} & \frac{-c_1}{m_0} \\ \frac{-k_1}{m_1} & k_1 m_1 & -c_1 m_1 & c_1 m_1 \end{bmatrix} \tag{5.9}$$

$$B = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \frac{F}{m_1} \end{bmatrix} \tag{5.10}$$

A is 4×4 matrix that consist of four 2×2 matrices which are called A_1, A_2, A_3 and A_4. The location these matrices can be seen in below

$$A = \begin{bmatrix} A_1 & A_2 \\ A_3 & A_4 \end{bmatrix} \tag{5.11}$$

A_1 is a zero matrix. A_2 is an identity matrix. A_3 and A_4 are spring constant and damping coefficient matrices respectively.A_3 and A_4 have same pattern. Now related code will be explained briefly in below.

```
from scipy.integrate import odeint
import numpy

"""Input of this program:
1:Degrre of freedom,
2:initial conditions,
3:array of m, array of k, array of c ,and array of f
output of this program: response of last element versus time
(also can have response of each element versus time)"""


""" DOF = Degree of freedom"""
DOF=2

""" n=number of states"""
n= (2*DOF)
```

First, *odeint* command should be imported from scipy.integrate. Numpy is another package that is needed in this code. We should have the number of degree of freedom, initial conditions (displacement and velocity of all elements)and system parameters such as masses, spring constants, damping coefficients and forces as inputs. n is the number of States which is two times more than degree of freedom. For example, for a two degree of freedom system, number of states (n) is equal four.

```
""" x-dot = Ax+B
 Matrix B is a n*1 matrix which shows applid force to elements
 Matrix A is a n*n matrix which includes 4 matices(A1,A2,A3,A4) of DOF*DOF
 A=[[A1, A2],[A3,A4]],
 A1: a zero DOF*DOF matrix
 A2: a I DOF*DOF matrix
 A3: a K(Spring Constant) DOF*DOF matrix
 A4: a C(Damper Constant) DOF*DOF matrix
"""
A=numpy.zeros((n,n))
B=numpy.zeros((n,1))
"""  m is the mass of the elements (kg)"""
m=array([2.0,3.0])

"""  k is the stiffness coefficients (N/m)"""

k=array([20,15])

"""  c is the damping coefficients (N*s/m)"""

c=array([5,6])
"""f is the force applied to the element (N)"""

f=array([0,1])
```

As explained A and B are n by n and n by 1 matrices respectively and Numpy.zeros define these matrices. In above, system parameters which include masses, spring constants, damping coefficients and forces defined.

```
""" Calcaulation of matrix B """

for i in range(DOF):
    B[i][0]=0.0

for i in range(DOF,n):
    B[i][0]=f[i-DOF]/m[i-DOF]
```

This part of code, describe matrix B. First half of this matrix is zero and the next half present the applied forces on elements over related masses.

```
""" Calculation of matrix A1"""

for i in range(DOF):
      for j in range(DOF):
         A[i][j]= 0

""" Calculation of matrix A2"""

for i in range (DOF):
      for j in range(DOF,n):
          if j-i==DOF :
              A[i][j]=1
```

This section shows the calculation of A_1 and A_2 matrices which shaped matrix A. A_1 is a zero matrix and A_2 is an identity matrix.

```
""" Calculation of matrix A3"""

for i in range (DOF,n):
      for j in range(DOF-1):
          if i-j==DOF :
              A[i][j]=-(k[j]+k[j+1])/m[i-DOF]
A[(n-1)][(DOF-1)]=-k[(DOF-1)]/m[DOF-1]

p=DOF
q=1
for i in range (DOF-1):
       A[p][q]=k[q]/m[p-DOF]
       p=p+1
       q=q+1

p=DOF+1
q=0
for j in range (DOF-1):
       A[p][q]=k[q+1]/m[p-DOF]
       p=p+1
       q=q+1
```

Next step is calculation of matrix A_3 which related code is above. This matrix shows the spring constant part of the matrix A.

```
""" Calculation of matrix A4"""

for i in range (DOF,n-1):
      for j in range(DOF,n-1):
          if i==j :
              A[i][j]=-(c[j-DOF]+c[j+1-DOF])/m[i-DOF]
A[n-1][n-1]=-c[DOF-1]/m[DOF-1]

p=DOF
q=DOF+1.0
for i in range (DOF-1):
       A[p][q]=c[q-DOF]/m[p-DOF]
       p=p+1
       q=q+1

p=DOF+1
q=DOF


for j in range (DOF-1):
       A[p][q]=c[q-DOF+1]/m[p-DOF]
       p=p+1
       q=q+1
```

The procedure to have matrix A_4 is exactly same as matrix A_3. However, this matrix show the damping coefficient part of the matrix A. Complete code can be seen in the second appendix.

The Out put of this code for mentioned two degree of freedom system in figure 5.3 with below parameters is: ($m_0 = 2, m_1 = 3, k_0 = 20, k_1 = 15, c_0 = 5, c_1 = 6, F = 1$ and with zero initial conditions)

To compare the response of the DT-TMM with *odeint*, the out put of two method for one degree of freedom system mentioned in section 4 is shown in figure 5.4. The results match each other which shows the accuracy of the DT-TMM method.

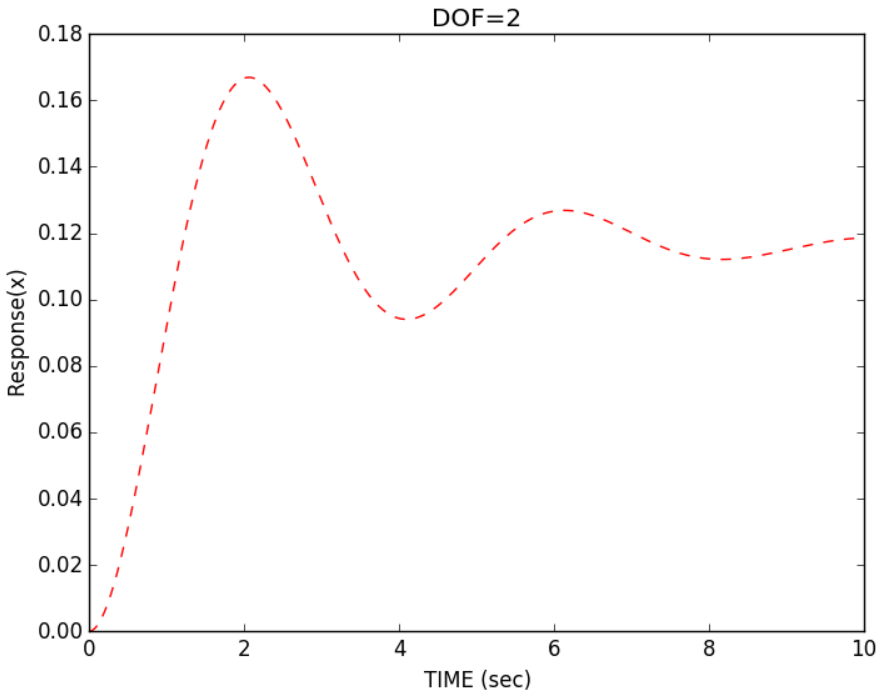


Figure 5.3: Step response of two degree of freedom system (odeint code)

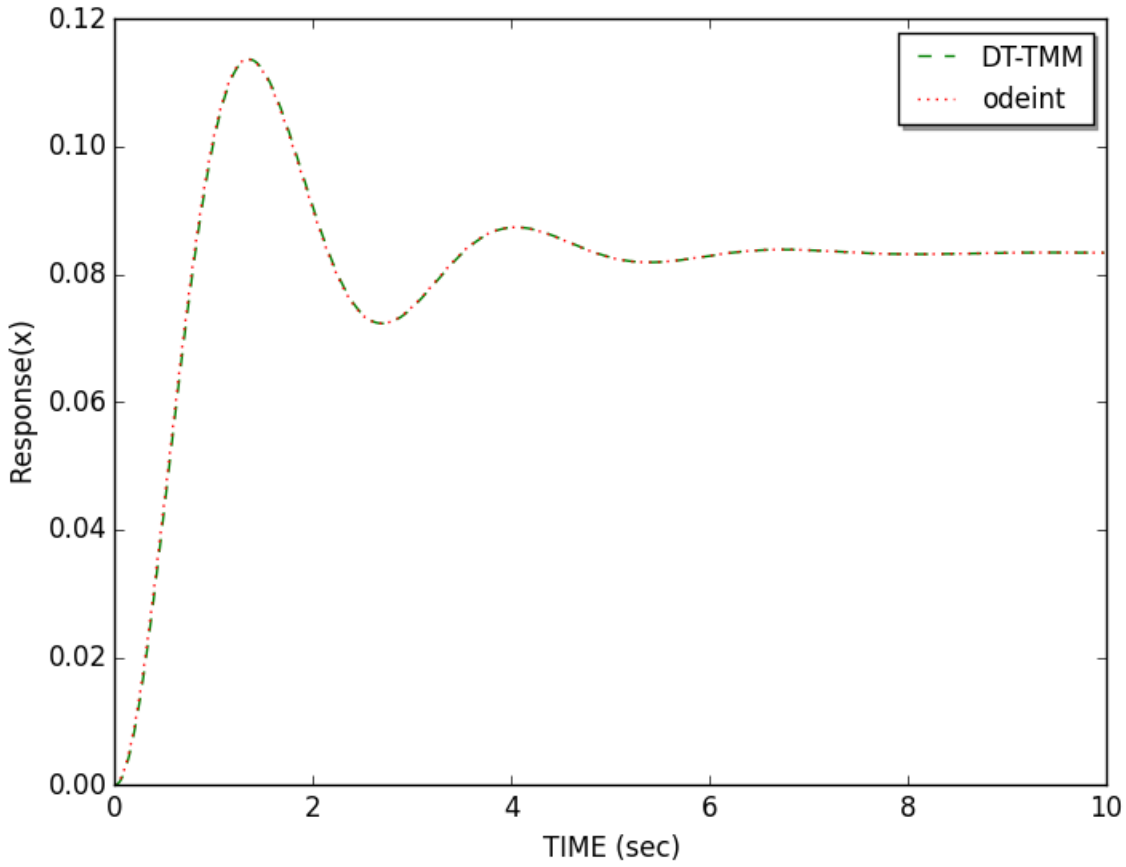


Figure 5.4: Comparison of DT-TMM and odeint output for one degree of freedom system

CHAPTER 6

Numerical Integration Methods in DT-TMM

6.1 Introduction

Most of the times differential equations do not have exact solution, so numerical integration method is one way that a good approximation of the solution can be achieved. There are two kind of numerical integration method first is explicit and another is implicit.

6.2 Explicit Method

Explicit is a straightforward method which means x_{n+1} can be reached by having x_n. To meke it claer let 's have a simple example for below equation.

$$\dot{x} = f(x, t) \tag{6.1}$$

By dividing time t to N steps, Δt is time step and equation 6.1 can be rewrite into discrete time format like below.

$$\dot{x_n} = f(x_n, t_n) \tag{6.2}$$

Suppose that the derivative approximation of step n is

$$\dot{x_n} = \frac{x_{n+1} - x_n}{\Delta t} \tag{6.3}$$

By plugging equation 6.3 in equation 6.2, we have

$$\frac{x_{n+1} - x_n}{\Delta t} = f(x_n, t_n) \tag{6.4}$$

$$x_{n+1} = x_n + \Delta t f(x_n, t_n) \tag{6.5}$$

x_{n+1} can be reached by having x_n. It means by having initial conditions(x and $\dot{x}$), an approximation solution can be calculated at time t.

6.3 Implicit Method

Implicit is a backward method which is a little different from explicit method. The difference is just in derivative approximation.

Suppose that the derivative approximation of step n is

$$\dot{x_n} = \frac{x_n - x_{n-1}}{\Delta t} \tag{6.6}$$

By plugging equation 6.6 in equation 6.2, we have

$$\frac{x_n - x_{n-1}}{\Delta t} = f(x_n, t_n) \tag{6.7}$$

$$x_n = x_{n-1} + \Delta t f(x_n, t_n) \tag{6.8}$$

By re-indexing equation 6.8, we have following equation for x_{n+1}

$$x_{n+1} = x_n + \Delta t f(x_{n+1}, t_{n+1}) \tag{6.9}$$

To have x_{n+1}, equation 6.9 should be solved for each step.

6.4 Numerical Integration Methods of DT-TMM

To combine DT-TMM and numerical integration methods, five below numerical integration methods are chosen to compare the results and choose the best.

1- Fox-Euler

2- Wilson Theat

3- Newmark Beta Method

4- Houbolt

5- Park Stiffly

These methods will be explained in details in following sections. In most of the these methods, using Taylor series is the point that calculation started. Taylor series is in below form.

$$x(t_i) = x(t_{i-1}) + \Delta T\dot{x}(t_{i-1}) + (\Delta T^2/2)\ddot{x}(t_{i-1}) \tag{6.10}$$

6.4.1 Fox-Euler

In this method, acceleration $\ddot{x_n}$ supposed to be constant during each time step and is equal acceleration at end point of related time step. This assumption leads to change in equation 6.10 which is shown in following.

$$x(t_i) = x(t_{i-1}) + \Delta T\dot{x}(t_{i-1}) + \frac{\Delta T^2}{2}\ddot{x}(t_i) \tag{6.11}$$

Equation 6.11 can be rewrite in below form

$$\ddot{x}(t_i) = \frac{2}{\Delta T^2}[x(t_i) - x(t_{i-1}) - \Delta T\dot{x}(t_{i-1})] \tag{6.12}$$

To use this method in DT-TMM, equation 6.12 should be be in the equation 4.1 (Linear based on the displacement) format which is written again in below.

$$\ddot{x_n}(t_i) = A_n(t_i)x_n(t_i) + B_n(t_i) \tag{6.13}$$

Equation 6.12 is

$$\ddot{x}(t_i) = \frac{2}{\Delta T^2}x(t_i) - \frac{2}{\Delta T^2}[x(t_{i-1}) + \Delta T\dot{x}(t_{i-1})] \tag{6.14}$$

A and B can be achieved by comparing 6.14 and 6.13

$$A = \frac{2}{\Delta T^2} \tag{6.15}$$

$$B = \frac{-2}{\Delta T^2}[x(t_{i-1}) + \Delta T\dot{x}(t_{i-1})] \tag{6.16}$$

There two left coefficients which should be computed for DT-TMM method. They are D and E which express equation of velocity. We have below equation which gives us the velocity when the acceleration is constant.

$$\dot{x}(t_i) = \ddot{x}(t_i)\Delta T + \dot{x}(t_{i-1}) \tag{6.17}$$

By substituting equation 6.12 in 6.17 we have.

$$\dot{x}(t_i) = \frac{2}{\Delta T}x(t_i) - \frac{2}{\Delta T}[x(t_{i-1}) + \Delta T\dot{x}(t_{i-1})] + \dot{x}(t_{i-1}) \tag{6.18}$$

Equation 6.18 should be rewrite in equation 4.2 format which is shown below

$$\dot{x_n}(t_i) = D_n(t_i)x_n(t_i) + E_n(t_i) \tag{6.19}$$

$$\dot{x}(t_i) = \frac{2}{\Delta T}x(t_i) - [\frac{2}{\Delta T}x(t_{i-1}) + \dot{x}(t_{i-1})] \tag{6.20}$$

Which means

$$D = \frac{2}{\Delta T} \tag{6.21}$$

$$E = -[\frac{2}{\Delta T}x(t_{i-1}) + \dot{x}(t_{i-1})] \tag{6.22}$$

The step response of two degree of freedom system in section 4 with $m_0 = 2, m_1 = 3, k_0 = 20, k_1 = 15, c_0 = 5, c_1 = 6, F = 1$ and zero initial conditions is compared with DT-TMM method by using Fox-Euler and *odeint* code in figure 6.1. As can be seen, they pretty match each other, however there are some area which responses are not same exactly.

6.4.2 Newmark Beta

In Fox-Euler method we supposed that acceleration is constant. To have more accurate answer, Newmark Beta method consider that acceleration change linearly in each time step. Parameters β and γ can be changed to have better answer for particular systems. This method is based on Taylor series too. Taylor series for $x_i{+}_1$ and is using.

$$x(t_{i+1}) = x(t_i) + \Delta T\dot{x}(t_i) + \frac{\Delta T^2}{2}\ddot{x}(t_i) + \frac{\Delta T^3}{6}\dddot{x}(t_i) \tag{6.23}$$

$$\dot{x}(t_{i+1}) = \dot{x}(t_i) + \Delta T\ddot{x}(t_i) + \frac{\Delta T^2}{2}\dddot{x}(t_i) \tag{6.24}$$

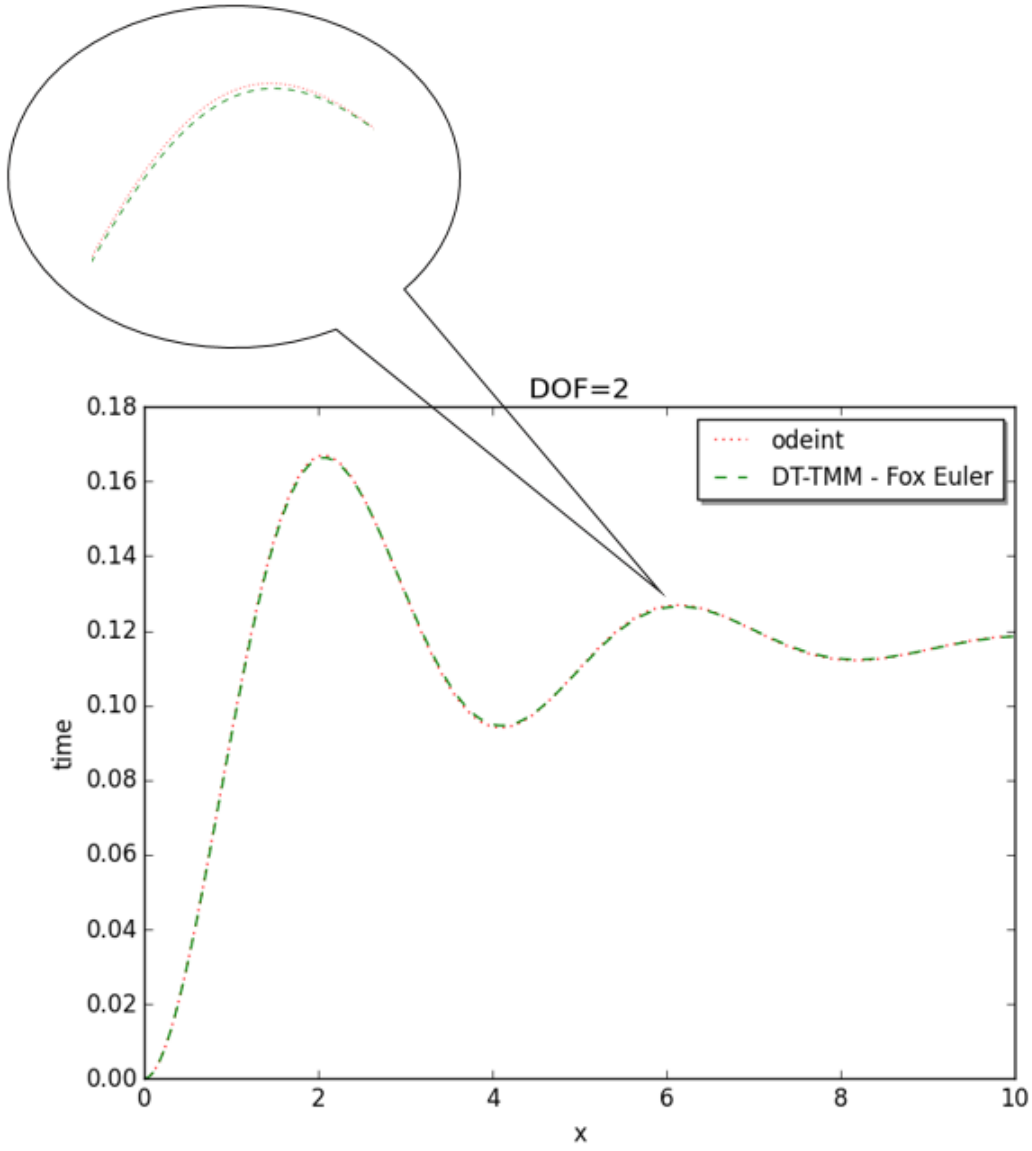


Figure 6.1: Comparison of DT-TMM Fox-Euler and odeint fot the two DOF system

Coefficients of third derivative of above equations can be considered as β and γ

$$x(t_{i+1}) = x(t_i) + \Delta T\dot{x}(t_i) + \frac{\Delta T^2}{2}\ddot{x}(t_i) + (\beta\Delta T^3)\dddot{x}(t_i) \tag{6.25}$$

$$\dot{x}(t_{i+1}) = \dot{x}(t_i) + \Delta T\ddot{x}(t_i) + (\gamma\Delta T^2)\dddot{x}(t_i) \tag{6.26}$$

If we assume that the acceleration change linearly between time steps we have

$$\dddot{x}(t_i) = \frac{\ddot{x}(t_{i+1}) - \ddot{x}(t_i)}{\Delta T} \tag{6.27}$$

Equation 6.27 can be substituted in equations 6.26 and 6.25

$$x(t_{i+1}) = x(t_i) + \Delta T\dot{x}(t_i) + (0.5 - \beta)\Delta T^2\ddot{x}(t_i) + \beta\Delta T^2\ddot{x}(t_{i+1}) \tag{6.28}$$

$$\dot{x}(t_{i+1}) = \dot{x}(t_i) + (1 - \gamma)\Delta T\ddot{x}(t_i) + \gamma\Delta T\ddot{x}(t_{i+1}) \tag{6.29}$$

6.28 and 6.29 equations can be write in below form to have acceleration and velocity.

$$\ddot{x}(t_{i+1}) = (\frac{1}{\beta\Delta T^2})[x(t_{i+1}) - x(t_i)] - (\frac{1}{\beta\Delta T})\dot{x}(t_i) - (\frac{1}{2\beta} - 1)\ddot{x}(t_i) \tag{6.30}$$

$$\dot{x}(t_{i+1}) = (\frac{1}{\beta\Delta T})[x(t_{i+1}) - x(t_i)] - (\frac{\gamma}{\beta} - 1)\dot{x}(t_i) - \Delta T(\frac{\gamma}{2\beta} - 1)\ddot{x}(t_i) \tag{6.31}$$

To use Newmark Beta method in DT-TMM, 6.30 and 6.31 equations should be rewrite in 4.1 and 4.2 equations format.

$$\ddot{x}(t_{i+1}) = (\frac{1}{\beta\Delta T^2})x(t_{i+1}) - (\frac{1}{\beta\Delta T^2})(x(t_i) + \Delta T\dot{x}(t_i) + (0.5-\beta)\Delta T^2\ddot{x}(t_i)) \tag{6.32}$$

By comparing equation 6.32 and 4.1 we have

$$A = (\frac{1}{\beta\Delta T^2}) \tag{6.33}$$

$$B = -(\frac{1}{\beta\Delta T^2})(x(t_i) + \Delta T\dot{x}(t_i) + (0.5-\beta)\Delta T^2\ddot{x}(t_i)) \tag{6.34}$$

Equation 6.31 is

$$\dot{x}(t_{i+1}) = (\frac{\gamma}{\beta\Delta T})x(t_{i+1}) + \dot{x}(t_i) + \Delta T[(1-\gamma)\ddot{x}(t_i) + \gamma B] \tag{6.35}$$

Comparison of 6.35 and 4.2 gives

$$D = \frac{\gamma}{\beta\Delta T} \tag{6.36}$$

$$E = \dot{x}(t_i) + \Delta T[(1-\gamma)\ddot{x}(t_i) + \gamma B] \tag{6.37}$$

Two degree of freedom system in section 4 is modeled again with Newmark Beta method and the result can be comprised with odeint (figure 6.2). It is clear that the result match each other ($\beta = \frac{1}{6}$, $\gamma = 0.5$)

β and γ can be changed based on the requested accuracy and stability of the system. Some of these parameters can be seen in below.

1- $\beta = \frac{1}{6}$, $\gamma = 0.5$: Acceleration change linearly at each time step

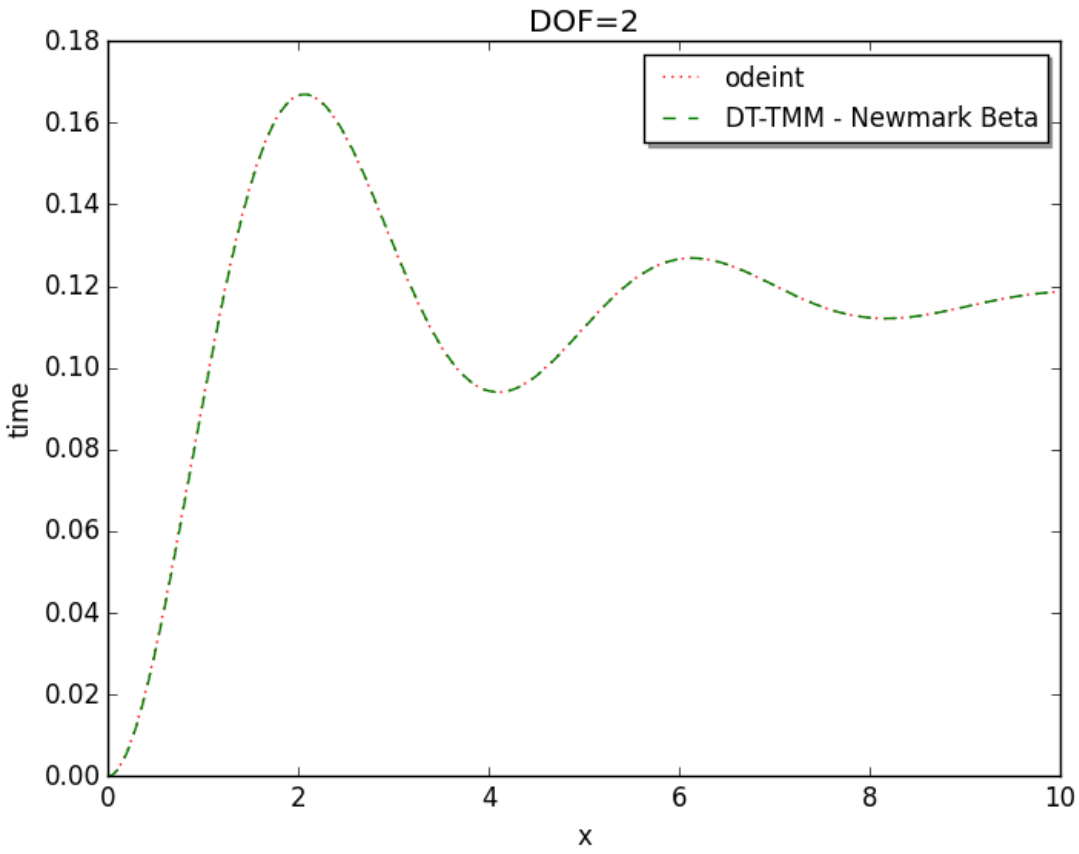


Figure 6.2: Comparison of DT-TMM Newmark Beta and odeint for the two DOF system

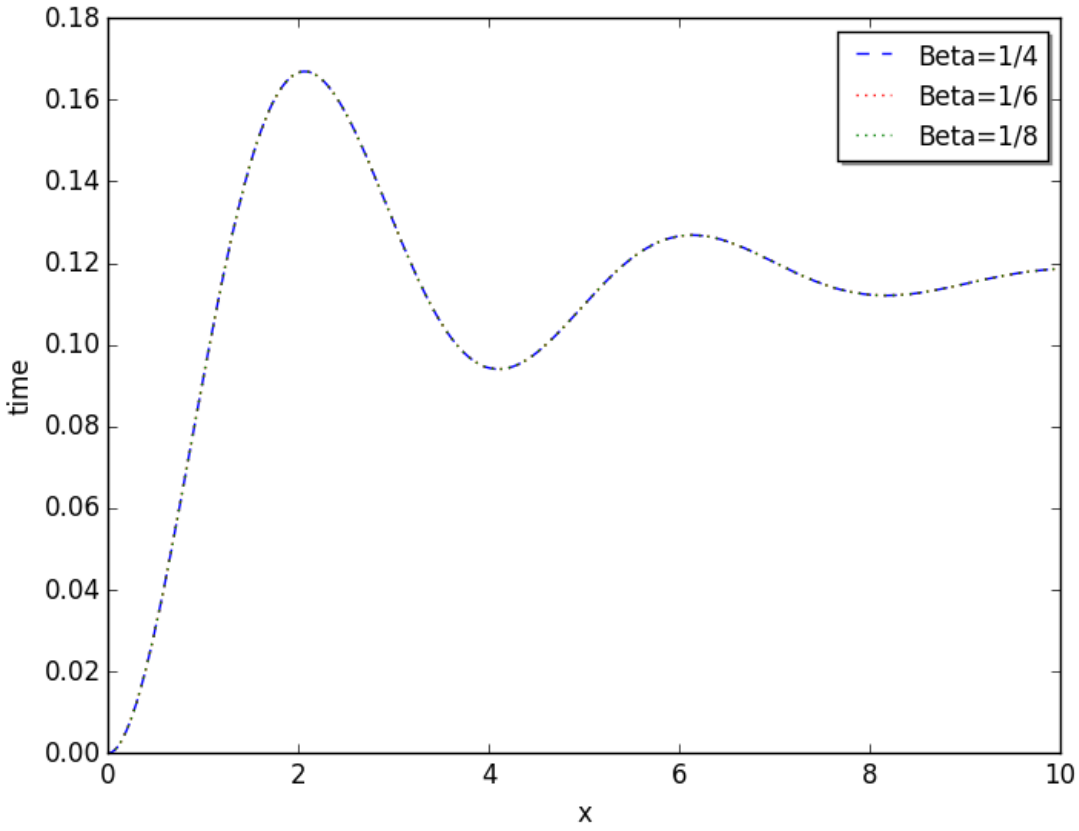


Figure 6.3: Comparison of DT-TMM Newmark Beta for different β

2- $\beta = 0$, $\gamma = 0.5$: Acceleration is constant and is equal starting point at each time step

3-$\beta = \frac{1}{8}$, $\gamma = 0.5$: Acceleration is constant and is equal starting point till middle of time step ($\ddot{x}(t_i) : t_i - t_i + \frac{\Delta T}{2}$) and then change to acceleration of end point ($\ddot{x}(t_{i+1}) : t_i + \frac{\Delta T}{2} - t_{i+1}$).

4- $\beta = \frac{1}{4}$, $\gamma = 0.5$: Acceleration is constant and equal the average of acceleration at start and end point of time step ($\frac{\ddot{x}(t_{i+1})+\ddot{x}(t_i)}{2}$).

Step response of two degree of freedom system based on DT-TMM Newmark Beat for different β is shown in figure 6.3. The results exactly match each other. However, in more complicated systems the result will not be same exactly.

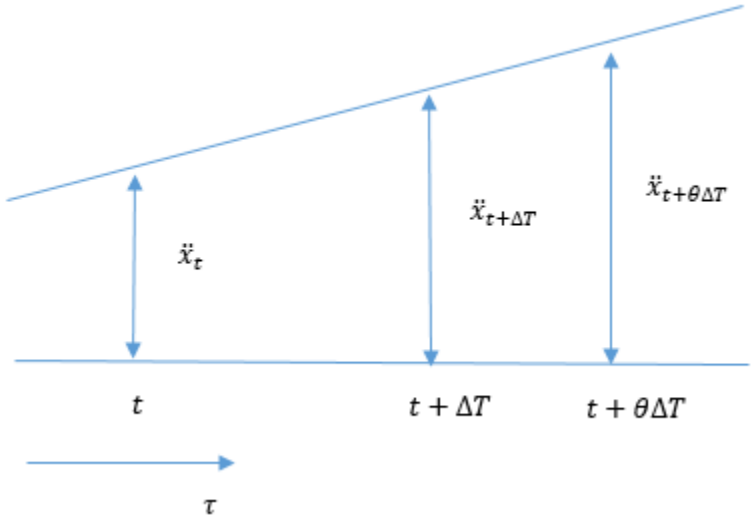


Figure 6.4: Wilson θ

6.4.3 Wilson Theta

Wilson Θ method is based on the assumption that acceleration change linearly between to time. Figure 6.4 presents this description that acceleration change from time t to $t+\theta\Delta T$ linearly when $\theta \geq 1$.

From time t to $t+\theta\Delta T$ we can have

$$\ddot{x}_{t+\tau} = \ddot{x}_t + \frac{\tau}{\theta\Delta T}(\ddot{x}_{t+\theta\Delta T} - \ddot{x}_t) \tag{6.38}$$

Taking the integration for equation 3.8 gives the velocity and displacement of the system.

$$\dot{x}_{t+\tau} = \dot{x}_t + \ddot{x}_t\tau + \frac{\tau^2}{2\theta\Delta T}(\ddot{x}_{t+\theta\Delta T} - \ddot{x}_t) \tag{6.39}$$

$$x_{t+\tau} = x_t + \dot{x}_t\tau + \frac{1}{2}\ddot{x}_t\tau^2 + \frac{\tau^3}{6\theta\Delta T}(\ddot{x}_{t+\theta\Delta T} - \ddot{x}_t) \tag{6.40}$$

By plugging $\tau = \theta\Delta T$ in above equations we have

$$\dot{x}_{t+\theta\Delta T} = \dot{x}_t + \frac{\theta\Delta T}{2}(\ddot{x}_t + \ddot{x}_{t+\theta\Delta T}) \tag{6.41}$$

$$x_{t+\theta\Delta T} = x_t + \theta\Delta T\dot{x}_t + \frac{\theta^2\Delta T^2}{6}(2\ddot{x}_t + \ddot{x}_{t+\theta\Delta T}) \tag{6.42}$$

Acceleration and velocity can be achieved from equations 6.42 and 6.41, also to use Wison θ in DT-TMM method they should be written in equation 4.1 and 4.2 format.

$$\ddot{x}_{t+\theta\Delta T} = \frac{6}{\theta^2\Delta T^2}x_{t+\theta\Delta T} - \frac{6}{\theta^2\Delta T^2}x_t - \frac{6}{\theta\Delta T}\dot{x}_t - 2\ddot{x}_t \tag{6.43}$$

$$\dot{x}_{t+\theta\Delta T} = \frac{3}{\theta\Delta T}x_{t+\theta\Delta T} - \frac{3}{\theta\Delta T}x_t - \frac{\theta\Delta T}{2}\ddot{x}_t - 2\dot{x}_t \tag{6.44}$$

By comparing equations 6.43 and 6.44 coefficients A, B, D and E can be reached.

$$A = \frac{6}{\theta^2\Delta T^2} \tag{6.45}$$

$$B = -\frac{6}{\theta^2\Delta T^2}x_t - \frac{6}{\theta\Delta T}\dot{x}_t - 2\ddot{x}_t \tag{6.46}$$

$$D = \frac{3}{\theta\Delta T} \tag{6.47}$$

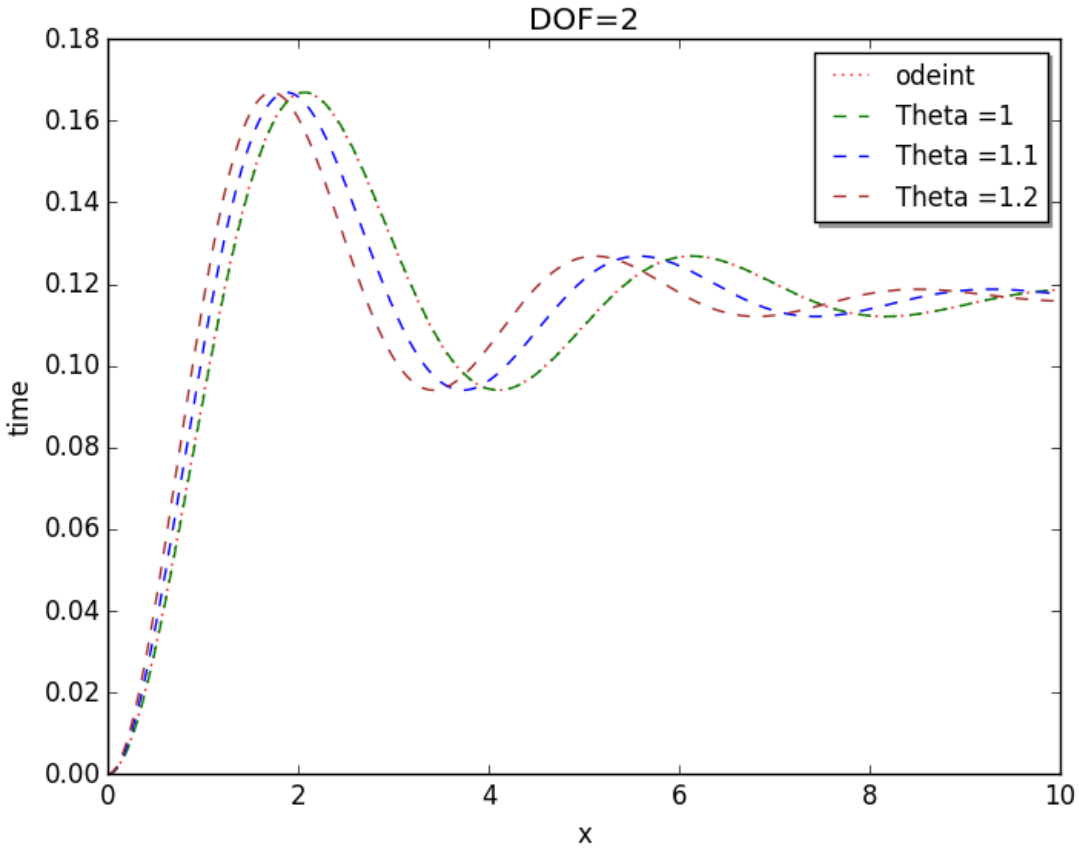


Figure 6.5: Comparison of Wilson θ with different θ with *odeint*

$$E = -\frac{3}{\theta\Delta T}x_t - \frac{\theta\Delta T}{2}\ddot{x}_t - 2\dot{x}_t \tag{6.48}$$

When $\theta = 1$, Wison θ method is same as Newmark Beta method when $\beta = \frac{1}{6}$ and $\gamma = 0.5$.

Comparison of DT-TMM Wilson θ with different and *odeint* for two degree of freedom system can be seen in figure 6.5.

6.4.4 Houbolt

Houbolt method is a implicit method which displacement of two step before is necessary. three below equation can be written based on the Taylor series

$$x_t = x_{t+\Delta T} - \Delta T\dot{x}_{t+\Delta T} + \frac{\Delta T^2}{2}\ddot{x}_{t+\Delta T} - \frac{\Delta T^3}{6}\dddot{x}_{t+\Delta T} \tag{6.49}$$

$$x_{t-\Delta T} = x_{t+\Delta T} - 2\Delta T\dot{x}_{t+\Delta T} + \frac{(2\Delta T)^2}{2}\ddot{x}_{t+\Delta T} - \frac{(2\Delta T)^3}{6}\dddot{x}_{t+\Delta T} \tag{6.50}$$

$$x_{t-2\Delta T} = x_{t+\Delta T} - 3\Delta T\dot{x}_{t+\Delta T} + \frac{(3\Delta T)^2}{2}\ddot{x}_{t+\Delta T} - \frac{(3\Delta T)^3}{6}\dddot{x}_{t+\Delta T} \tag{6.51}$$

By solving above equations for velocity and acceleration we have

$$\dot{x}_{t+\Delta T} = \frac{1}{6\Delta T}(11x_{t+\Delta T} - 18x_t + 9x_{t-\Delta T} - 2x_{t-2\Delta T}) \tag{6.52}$$

$$\ddot{x}_{t+\Delta T} = \frac{1}{\Delta T^2}(2x_{t+\Delta T} - 5x_t + 4x_{t-\Delta T} - x_{t-2\Delta T}) \tag{6.53}$$

BY comparing equations 6.52 and 6.53 with equations 4.1 and 4.2 coefficient of DT-TMM method (A, B, D and E) can be calculated.

$$A = \frac{2}{\Delta T^2} \tag{6.54}$$

$$B = \frac{-1}{\Delta T^2}(5x_t - 4x_{t-\Delta T} - +x_{t-2\Delta T}) \tag{6.55}$$

$$D = \frac{11}{6\Delta T} \tag{6.56}$$

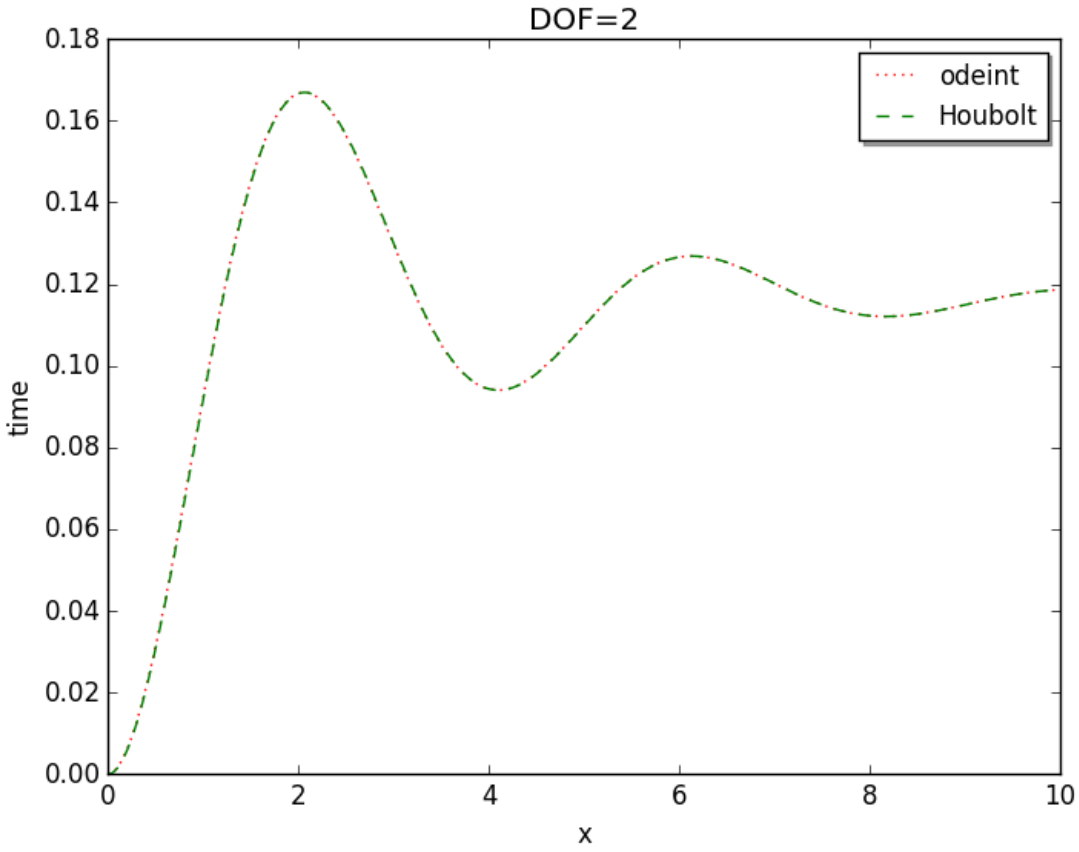


Figure 6.6: Comparison of Houbolt and *odeint* for the two degree of freedom system

$$E = \frac{-1}{6\Delta T}(18x_t - 9x_{t-\Delta T} + 2x_{t-2\Delta T}) \tag{6.57}$$

Since we need to have two steps before in Houbolt method, Other method like Newmark Beta or Fox-Euler which do not need to have steps before can be used in just first two steps. In this research, Newmark Beta is used to have the data for first two steps. In figure 6.6, step response of Houbolt and *odeint* for two degree of freedom system compared together. The result match each other.

6.4.5 Park Stiffly Stable

This method is good one for low frequency systems which will be explained in below. This method is implicit one that need to have two steps before, so Newmark Beta is used for two first steps. Park Stiffly Stable is the average of two below equations which is derived from Houbolt method.

$$\dot{x}_{t+\Delta T} = \frac{1}{6\Delta T}(11x_{t+\Delta T} - 18x_t + 9x_{t-\Delta T} - 2x_{t-2\Delta T}) \tag{6.58}$$

$$\dot{x}_{t+\Delta T} = \frac{1}{2\Delta T}(2x_{t+\Delta T} - 4x_t + x_{t-\Delta T}) \tag{6.59}$$

Average of these equations gives

$$\dot{x}_{t+\Delta T} = \frac{1}{6\Delta T}(10x_{t+\Delta T} - 15x_t + 6x_{t-\Delta T} - x_{t-2\Delta T}) \tag{6.60}$$

By taking the derivative acceleration can be reached

$$\ddot{x}_{t+\Delta T} = \frac{1}{6\Delta T}(10\dot{x}_{t+\Delta T} - 15\dot{x}_t + 6\dot{x}_{t-\Delta T} - \dot{x}_{t-2\Delta T}) \tag{6.61}$$

Equation 6.60 substitute in equation 6.61 to have

$$\ddot{x}_{t+\Delta T} = \frac{10}{36\Delta T^2}(10x_{t+\Delta T} - 15x_t + 6x_{t-\Delta T} - x_{t-2\Delta T}) + \frac{1}{6\Delta T}(-15\dot{x}_t + 6\dot{x}_{t-\Delta T} - \dot{x}_{t-2\Delta T}) \tag{6.62}$$

To use Park Stiffly stable method in the DT-TMM, by comparing equations 6.62 and 6.60 with equations 4.1 and 4.2 we have

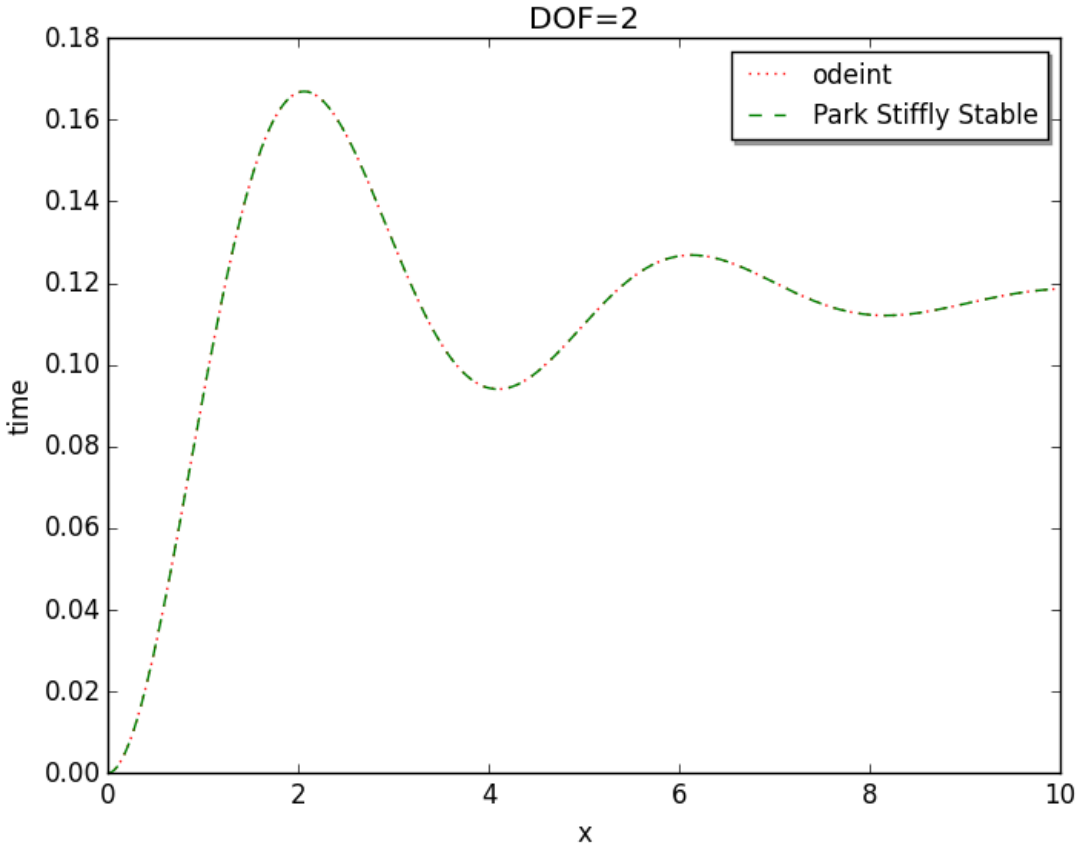


Figure 6.7: Comparison of Park Stiffly Stable and *odeint* for the two degree of freedom system

$$A = \frac{100}{36\Delta T^2} \tag{6.63}$$

$$B = \frac{10}{36\Delta T^2}(-15x_t + 6x_{t-\Delta T} - x_{t-2\Delta T}) + \frac{1}{6\Delta T}(-15\dot{x}_t + 6\dot{x}_{t-\Delta T} - \dot{x}_{t-2\Delta T}) \tag{6.64}$$

$$D = \frac{10}{6\Delta T} \tag{6.65}$$

$$E = \frac{1}{6\Delta T}(-15x_t + 6x_{t-\Delta T} - x_{t-2\Delta T}) \tag{6.66}$$

Figure 6.7 shows the accuracy of Park Stiffly stable method compared with *odeint* for two degree of freedom system. As told before, for first two steps, Newmark Beta used. Result shows that two graphs match each other accurately.

To compare all the methods in one huge system, a 15 degree of freedom system which studied by kumar and Sankar[24] investigated. For this issue, a Python code based on the DT-TMM for n degree of freedom system is written which will be explained in next chapter briefly. The digression of each method can be seen by comparing with *odeint*. These systems is just mass- spring without damper. Related parameters is in table 6.1

Station	Mass(kg)	K (N/m)	F(N)	x_0	$\dot{x}_0$
1	10	55000	0	0	0
2	9	50000	0	0	0
3	8	45000	0	0	0
4	7	40000	0	0	0
5	6	40000	0	0	0
6	6	35000	0	0	0
7	5.5	30000	0	0	0
8	5	25000	0	0	0
9	4.5	20000	0	0	0
10	4	15000	0	0	0
11	3.5	10000	0	0	0
12	4	9000	0	0	0
13	2.5	8000	0	0	0
14	2	7000	0	0	0
15	2	6000	1	0	0

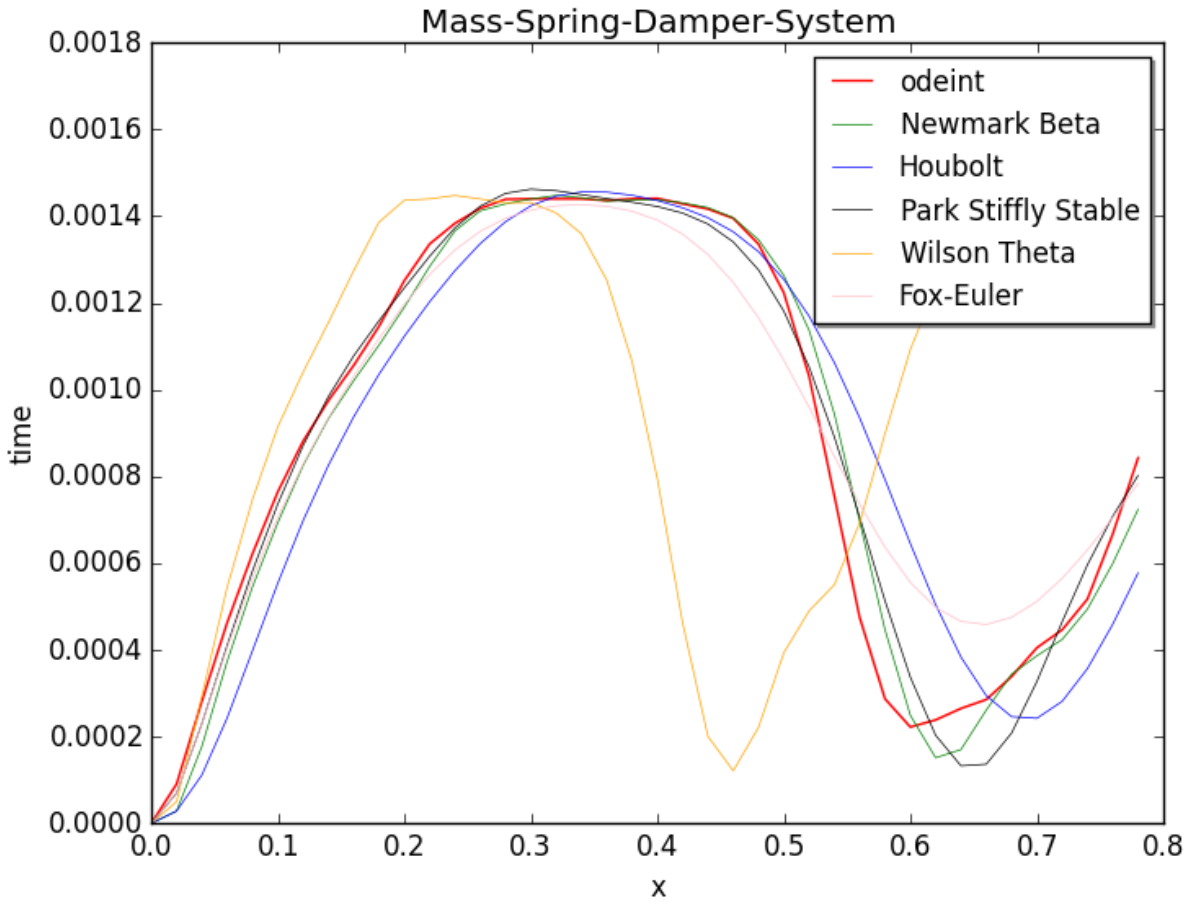


Figure 6.8: Step response of 15 DOF system with different DT-TMM method

Table 6.1: Parameters of the system (DOF=15)

The closest answer belong to Newmark Beta. Now, Trying to find Best β and γ to have more accurate answer.

When $\beta = \frac{1}{8}$, we have more accurate response which means Acceleration is constant and is equal starting point till middle of time step $(\ddot{x}(t_i) : t_i - t_i + \frac{\Delta T}{2})$ and then change to acceleration of end point $(\ddot{x}(t_{i+1}) : t_i + \frac{\Delta T}{2} - t_{i+1}\ )$.

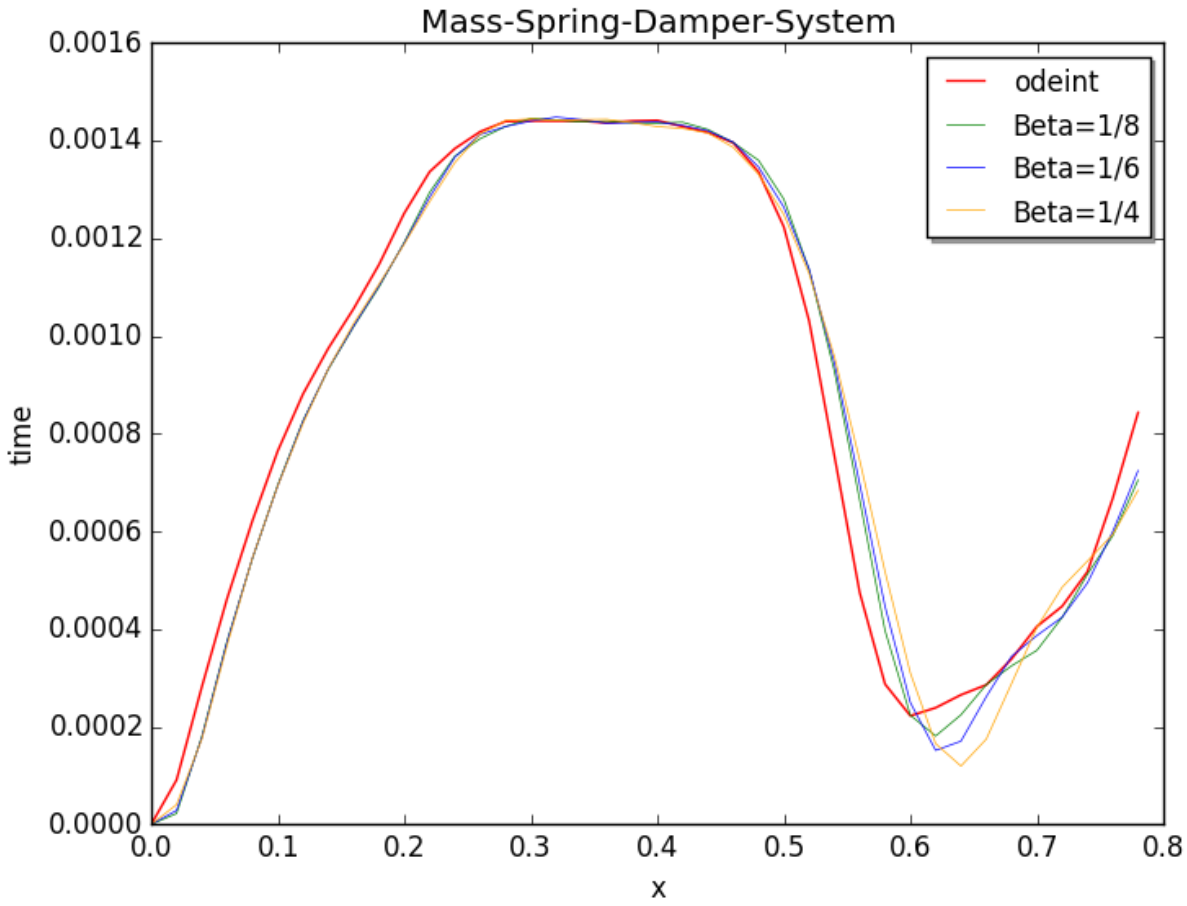


Figure 6.9: Step response of the 15 DOF system based on Newmark Beta for different β

CHAPTER 7

New Approach Based on the Acceleration for DT-TMM

7.1 One Degree of Freedom System

In this section a new approach which is based on the acceleration instead of displacement is studied. State vector in this method is based on the acceleration and internal force. New DT-TMM method for one degree of freedom system (figure 7.1) will be explained to make it clear.

Below equations show displacement and velocity in terms of acceleration

$$\dot{x_n}(t_i) = A_n(t_i)\ddot{x}_n(t_i) + B_n(t_i) \tag{7.1}$$

$$x_n(t_i) = D_n(t_i)\ddot{x}_n(t_i) + E_n(t_i) \tag{7.2}$$

These equations are linear based on the acceleration. Subscript n is related to the location of the part and denote the number of subsystem. Index i show the related time step. Moreover, n_{th} station is composed of m_n, k_n and c_n. $\dot{x_n}$ and $\ddot{x_n}$ show the velocity and acceleration of m_n .

A_n, B_n, D_n and E_n are the coefficients of equations 7.1 and 7.2 which should be calculated in each time step based on the related numerical integration method.

As the spring-damper is mass less

$$f_0 = f1 \tag{7.3}$$

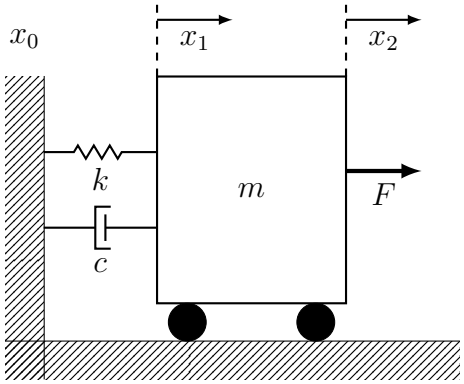


Figure 7.1: One degree of freedom mass-spring-damper system

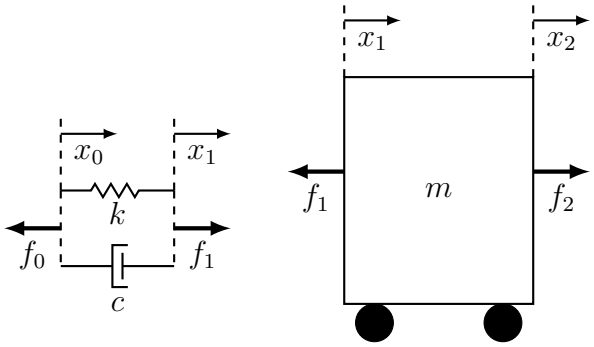


Figure 7.2: Exploded view of One degree of freedom mass-spring-damper system

force on the mass-spring is

$$f_0 = k(x_1 - x_0) + c(\dot{x_1} - \dot{x_0}) \tag{7.4}$$

And force equation for the mass is

$$f_2 - f_1 = m\ddot{x_1} \tag{7.5}$$

Since, the mass is rigid

$$x_2 = x_1 \tag{7.6}$$

To combine numerical integration and TMM methods, equation 7.1 and 7.2 need to be plugged in equations 7.4.

By plugging in 7.1 and 7.2 in 7.4

$$\begin{aligned} f_0 = f_1 = \ & k[D_1(t_i)\ddot{x}_1(t_i) + E_1(t_i) - D_0(t_i)\ddot{x}_0(t_i) - E_0(t_i)] + c[A_1(t_i)\ddot{x}_1(t_i) \\ & + B_1(t_i) - A_0(t_i)\ddot{x}_0(t_i) - B_0(t_i)] \end{aligned} \tag{7.7}$$

Equation 7.7 can be written for x_1

$$\ddot{x}_1(t_i) = \frac{f_1}{kD_1(t_i) + cA_1(t_i)} + \frac{kD_0(t_i) + cA_0(t_i)}{kD_1(t_i) + cA_1(t_i)}\ddot{x}_0(t_i) + \frac{k(E_0(t_i) - E_1(t_i)) + c(B_0(t_i) - B_1(t_i))}{kD_1(t_i) + cA_1(t_i)} \tag{7.8}$$

Equations of 7.8 and 7.3 can be written in matrix form

$$\begin{bmatrix} \ddot{x}_1(t_i) \\ f_1 \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{kD_0(t_i)+cA_0(t_i)}{kD_1(t_i)+cA_1(t_i)} & \frac{1}{kD_1(t_i)+cA_1(t_i)} & \frac{k(E_0(t_i)-E_1(t_i))+c(B_0(t_i)-B_1(t_i))}{kD_1(t_i)+cA_1(t_i)} \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \ddot{x}_0(t_i) \\ f_0 \\ 1 \end{bmatrix} \tag{7.9}$$

U_{sd} is DT-TMM transfer matrix for spring-damper which transfer state vectors from point 0 to point 1.

$$U_{sd} = \begin{bmatrix} \frac{kD_0(t_i)+cA_0(t_i)}{kD_1(t_i)+cA_1(t_i)} & \frac{1}{kD_1(t_i)+cA_1(t_i)} & \frac{k(E_0(t_i)-E_1(t_i))+c(B_0(t_i)-B_1(t_i))}{kD_1(t_i)+cA_1(t_i)} \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{7.10}$$

In the mentioned one degree of freedom system (figure 7.1), one station just exist, so D_0 and E_0 are equal zero.

From equation 7.6 we have

$$\ddot{x}_1 = \ddot{x}_0 \tag{7.11}$$

Equation 7.11 and 7.5 can be written in matrix form

$$\begin{bmatrix} \ddot{x}_2 \\ f_2 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ m & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \ddot{x}_1 \\ f_1 \\ 1 \end{bmatrix} \tag{7.12}$$

U_m is DT-TMM transfer matrix for mass which transfer state vectors from point 1 to point 2.

$$U_m = \begin{bmatrix} 1 & 0 & 0 \\ m & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{7.13}$$

Matrix U_f is same as before.

$$U_f = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -F \\ 0 & 0 & 1 \end{bmatrix} \tag{7.14}$$

Now by having transfer matrices for spring-damper, mass and force system transfer matrix can be achieved. The method of multiplication of the matrices is exactly same as what we had before and it starts from end point to start point.

$$U_{sys} = U_f U_m U_{sd} \tag{7.15}$$

State vector of base is

$$Z_{base} = \begin{bmatrix} \ddot{x}_{base} \\ f_{base} \\ 1 \end{bmatrix} \tag{7.16}$$

State vector of end is

$$Z_{end} = \begin{bmatrix} \ddot{x}_{end} \\ f_{end} \\ 1 \end{bmatrix} \tag{7.17}$$

By applying boundary conditions which are

$$F_{end} = 0 \tag{7.18}$$

$$\ddot{x}_{base} = 0 \tag{7.19}$$

Base and end state vectors are

$$Z_{base} = \begin{bmatrix} 0 \\ f_{base} \\ 1 \end{bmatrix} \tag{7.20}$$

$$Z_{end} = \begin{bmatrix} \ddot{x}_{end} \\ 0 \\ 1 \end{bmatrix} \tag{7.21}$$

By having U_{sys}

$$Z_{end} = U_{sys} Z_{base} \tag{7.22}$$

U_{sys} Can be written in this form

$$U_{sys} = \begin{bmatrix} U_{11} & U_{12} & U_{13} \\ U_{21} & U_{22} & U_{23} \\ U_{31} & U_{32} & U_{33} \end{bmatrix} \tag{7.23}$$

Plugging in Z_{end} ,Z_{base} and U_{sys} in equation 7.22

$$\begin{bmatrix} \ddot{x}_{end} \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} U_{11} & U_{12} & U_{13} \\ U_{21} & U_{22} & U_{23} \\ U_{31} & U_{32} & U_{33} \end{bmatrix} \begin{bmatrix} 0 \\ f_{base} \\ 1 \end{bmatrix} \tag{7.24}$$

x_{end} And f_{base} are unknown From line 2 of equation 7.24

$$U_{22} f_{base} + U_{23} = 0 \tag{7.25}$$

$$f_{base} = \frac{-U_{23}}{U_{22}} \tag{7.26}$$

State vector of base is

$$Z_{base} = \begin{bmatrix} 0 \\ \frac{-U_{23}}{U_{22}} \\ 1 \end{bmatrix} \tag{7.27}$$

By having Z_{base} , state vector of every point in the system can be obtained.

Now by having the initial conditions (x_0 , $\dot{x_0}$ and $\ddot{x_0}$), we can have the displacement of the system during the time and have a time-domain output.

7.2 Numerical Integration of New Method

A_n, B_n, D_n and E_n are the coefficients of equations 7.1 and 7.2 which should be calculated in each time step based on the related numerical integration method. In below approaches to calculate coefficients will be explained in details.

7.2.1 Fox-Euler

Fox-Euler is based on the Taylor series and the assumption that the acceleration $\ddot{x_n}$ is equal to acceleration at end point during each time step. Taylor series give below equation

$$x(t_i) = x(t_{i-1}) + \Delta T \dot{x}(t_{i-1}) + (\frac{\Delta T^2}{2})\ddot{x}(t_i) \tag{7.28}$$

Comparison of this equation and equation 7.2 gives coefficients D and E.

$$D = \frac{\Delta T^2}{2} \tag{7.29}$$

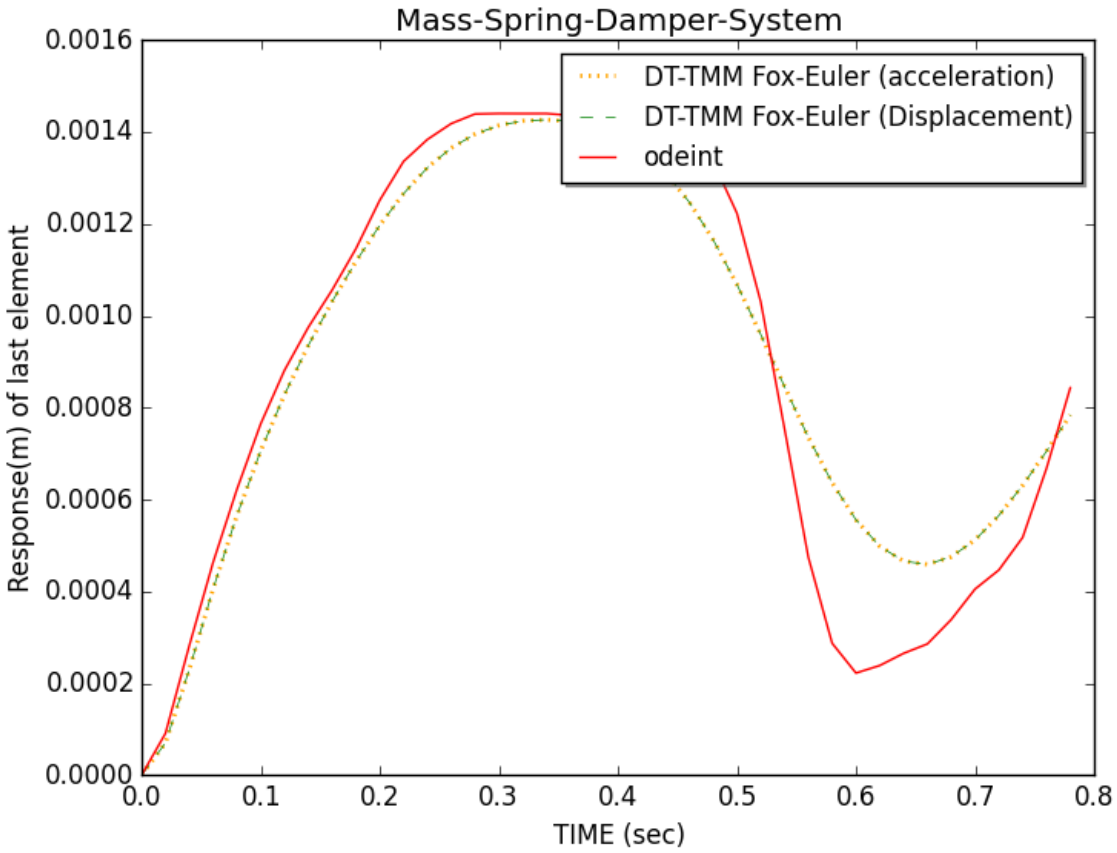


Figure 7.3: Comparison of Fox-Euler DT-TMM based on acceleration and displacement with *odeint*

$$E = x(t_{i-1}) + \Delta T \dot{x}(t_{i-1}) \tag{7.30}$$

We have below equation which gives us the velocity when the acceleration is constant.

$$\dot{x}(t_i) = \ddot{x}(t_i)\Delta T + \dot{x}(t_{i-1}) \tag{7.31}$$

This equation is like equation 7.1 when

$$A = \Delta T \tag{7.32}$$

$$B = \dot{x}(t_{i-1}) \tag{7.33}$$

Graph 7.3 shows the comparison of DT-TMM based on acceleration and displacement with *odeint*. As can be seen the result of DT-TMM for acceleration and displacement is same.

7.2.2 Newmark Beta

Newmark Beta method consider linear change of acceleration in each time step. We have below equations for this method which give us velocity and displacement.

$$x(t_{i+1}) = x(t_i) + \Delta T \dot{x}(t_i) + (0.5 - \beta)\Delta T^2 \ddot{x}(t_i) + \beta \Delta T^2 \ddot{x}(t_{i+1}) \tag{7.34}$$

$$\dot{x}(t_{i+1}) = \dot{x}(t_i) + (1 - \gamma)\Delta T \ddot{x}(t_i) + \gamma \Delta T \ddot{x}(t_{i+1}) \tag{7.35}$$

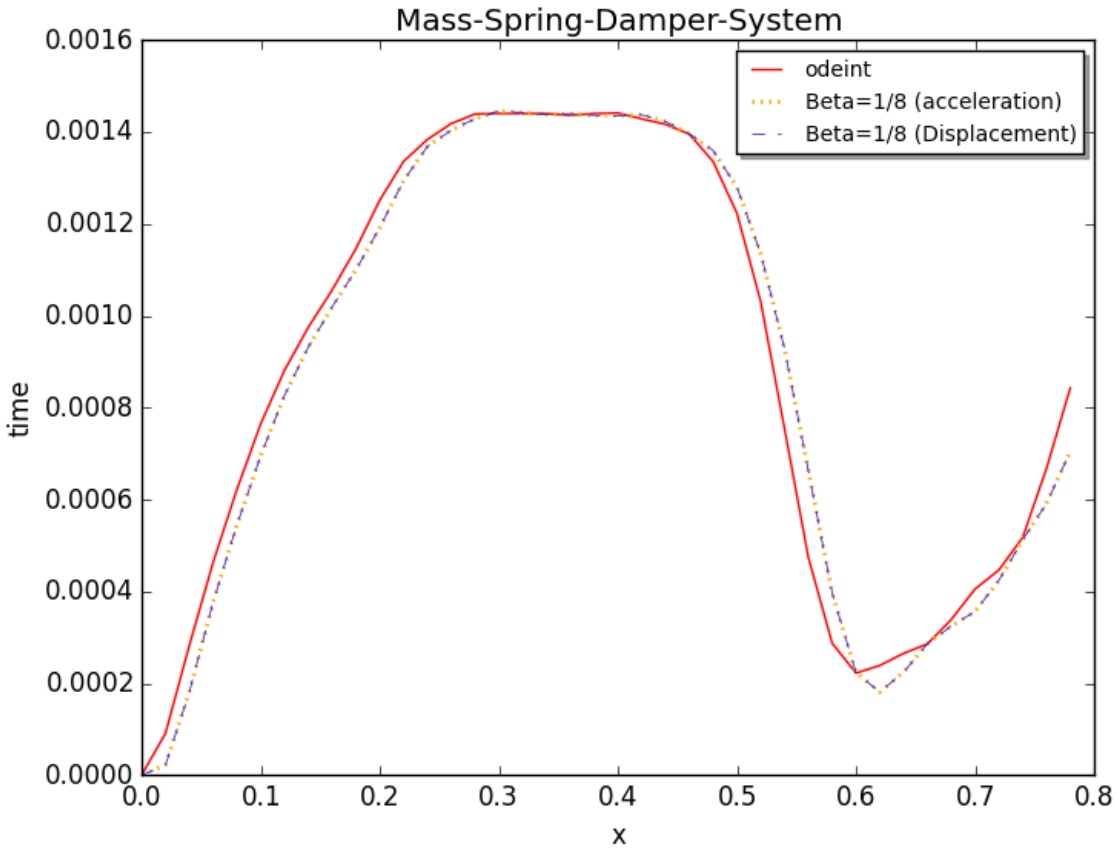


Figure 7.4: Comparison of Newmark Beta DT-TMM based on acceleration and displacement with *odeint*

Comparing these equations with 7.1 and 7.2 equations gives the DT-TMM coefficients.

$$D = \beta \Delta T^2 \tag{7.36}$$

$$E = x(t_i) + \Delta T \dot{x}(t_i) + (0.5 - \beta)\Delta T^2 \ddot{x}(t_i) \tag{7.37}$$

$$A = \gamma \Delta \tag{7.38}$$

$$B = \dot{x}(t_i) + (1 - \gamma)\Delta T \ddot{x}(t_i) \tag{7.39}$$

Figure 7.4 shows the comparison of DT-TMM based on acceleration and displacement with *odeint* for Newmark Beta when $\beta = \frac{1}{8}$. The result of DT-TMM for acceleration and displacement is same.

7.2.3 Wilson θ

Wilson θ method is based on the assumption that acceleration change linearly between to time. Below equations gives the velocity and displacement.

$$\dot{x}_{t+\theta\Delta T} = \dot{x}_t + \frac{\theta \Delta T}{2}(\ddot{x}_t + \ddot{x}_{t+\theta\Delta T}) \tag{7.40}$$

$$x_{t+\theta\Delta T} = x_t + \theta \Delta T \dot{x}_t + \frac{\theta^2 \Delta T^2}{6}(2\ddot{x}_t + \ddot{x}_{t+\theta\Delta T}) \tag{7.41}$$

By comparing equations 7.40 and 7.41 coefficients A, B, D and E can be reached.

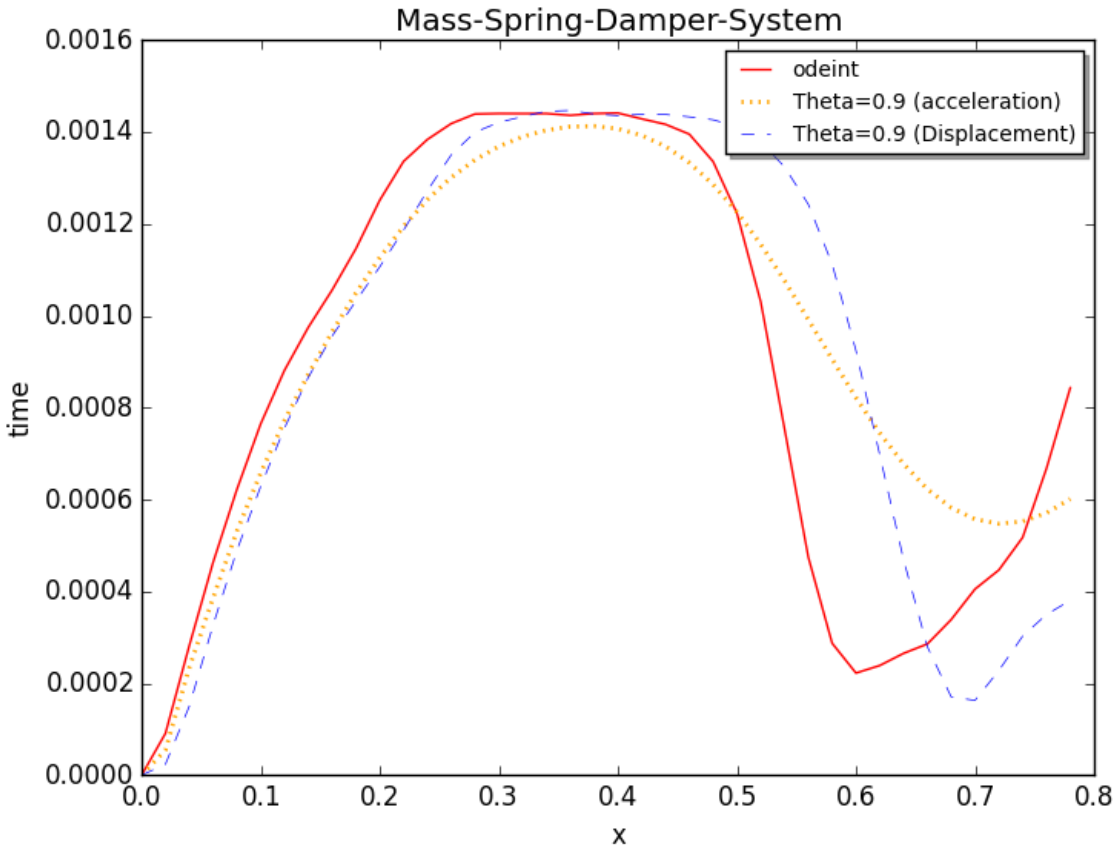


Figure 7.5: Comparison of Wilson θ DT-TMM based on acceleration and displacement with *odeint* when $\theta = 0.9$

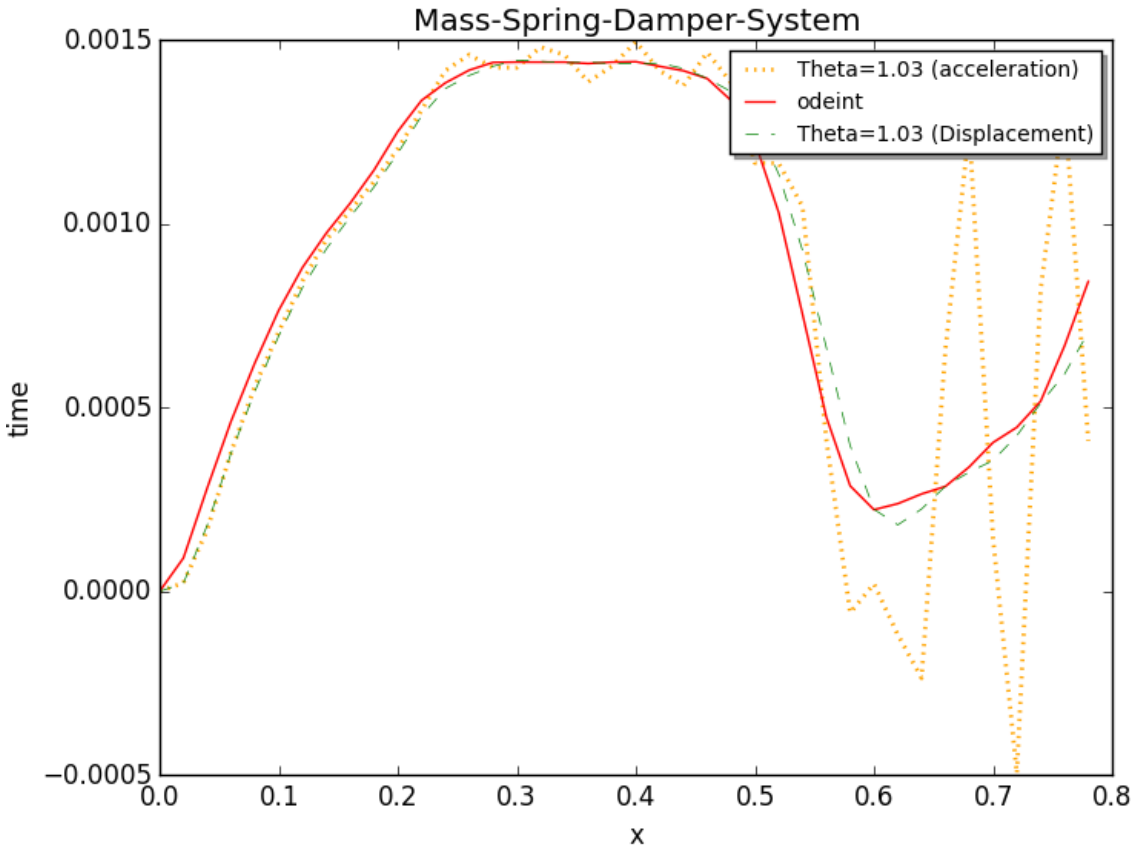


Figure 7.6: Comparison of Wilson θ DT-TMM based on acceleration and displacement with *odeint* when $\theta = 1.03$

$$A = \frac{\theta \Delta T}{2} \tag{7.42}$$

$$B = \dot{x}_t + \frac{\theta \Delta T}{2} \ddot{x}_t \tag{7.43}$$

$$D = \frac{\theta^2 \Delta T^2}{6} \tag{7.44}$$

$$E = x_t + \theta \Delta T \dot{x}_t + \frac{\theta^2 \Delta T^2}{3} \ddot{x}_t \tag{7.45}$$

Below graph show the comparison of *odeint* with DT-TMM based on the acceleration and displacement for Wilson θr when $\theta = 0.9$. As can be seen, the output based on displacement is more accurate. Figure 25 shows that Wilson θ based on the acceleration for $\theta > 1$ is unstable.

7.3 Pyhton Code for n Degree of Freedom with New Approach of DT-TMM

Below code shows the calculation of A, B, D and E for Fox-Euler, Newmark Beta and Wilson .

```
if Method==0:
    for n in range(DOF):
        A[n+1] = (dt*theta)/2.0
        B[n+1] = ((dt*theta)/2.0)* (xddot[n][p-1]) + xdot[n][p-1]
        E[n+1] = (((dt**theta)**2)/3.0) * (xddot[n][p-1])
        + (theta*dt * xdot[n][p-1])+x [n][p-1]
        D[n+1] = ((dt**theta)**2)/6.0

if Method==1:
    for n in range(DOF):
        A[n+1] = dt
        B[n+1] = xdot[n][p-1]
        E[n+1] = x[n][p-1] + (dt * xdot[n][p-1])
        D[n+1] = (dt**2)/2
if Method==2:
    for n in range(DOF):
        A[n+1] = gamma*dt
        B[n+1] = xdot[n][p-1] + dt*(1-gamma)*xddot[n][p-1]
        E[n+1] = x[n][p-1] + dt*xdot[n][p-1]
        + (0.5 - beta) * xddot[n][p-1] * (dt**2)
        D[n+1] = beta*(dt**2)
```

New transferee matrices (U_f, U_{sd} and U_m) based on the acceleration calculated with python as following

```
for n in range(DOF):
    for i in range(3):
        for j in range(3):
            if i==j:
                Um[i][j]=1.0
    Um[1][0]=m[n]


    for i in range(1,3):
        for j in range(1,3):
            if i==j:
                Usd[i][j]=1
        Usd[0][0]=(k[n]*D[n]+c[n]*A[n])/(k[n]*D[n+1]+c[n]*A[n+1])
        Usd[0][1]=(1.0/(k[n]*D[n+1]+c[n]*A[n+1]))
        Usd[0][2]=
        (c[n]*(B[n]-B[n+1])+k[n]*(E[n]-E[n+1])) / (k[n]*D[n+1]+c[n]*A[n+1])

    for i in range(3):
        for j in range(3):
            if i==j :
                Uf[i][j]=1.0
        Uf[1][2]=-f[n]
```

CHAPTER 8

CONCLUSION

8.1 Conclusion

In this research, new methods for modeling and controlling large systems with a lot of subsystems and flexible structures studied. There are many ways that mentioned systems can be modeled, but each of them has its benefits and drawbacks.

The Transfer Matrix Method which is called TMM is one of these methods that that is very efficient to model large structures. TMM reduces the size of matrices which results in fewer computation. Also, it is easy to add or eliminate one subsystem to our system without any issues. TMM like any method has its limitations which decrease its application. This method just perform frequency domain output, since it is not possible to have time-domain response of the system. Moreover, due to using Laplace transform, it can not model non-linear systems.

To overcome the drawbacks of TMM, numerical integration method can be combined. This new method is called Discrete Time Transfer Matrix Method (DT-TMM). By using DT-TMM, real response of the system during the time can be achieved. In this way, we can have time-domain output. Also, non-linear systems can be modeled by DT-TMM. It should be mentioned that DT-TMM has the benefits of TMM at the same time. It means, DT-TMM has low size of matrices, fewer computation and flexibility too. There are some numerical integration methods can be combined with TMM which some of them studied in this research. Every numerical method has different accuracy which depends on some factors like time step.

In chapter 3, state vectors and transfer matrix method is introduced. A one and two degree of freedom modeled in this chapter to make the procedure clear.

Chapter 4, talks about the DT-TMM method. In this chapter, One degree of freedom

mass-spring- damper system modeled with DT-TMM method. To have TMM and numerical integration method simultaneously, acceleration and velocity described in terms of displacement. Also, DT-TMM Python code for one degree of freedom explained in details.

To compare the output of DT-TMM with a real system, in chapter 5, a Python code is written for a system with n degree of freedom based on the ODE integration. The algorithm of this code explained. The output of DT-TMM and *ode* integration matched each other which shows the accuracy of the DT-TMM method.

In chapter 6, different numerical integration methods which can be used in the DT-TMM investigated. Fox-Euler, Wilson *theat* , Newmark Beta, Houbolt and Park Stiffly stable are the method which studied. Coefficients of the DT-TMM based on acceleration calculated for each method and response of a two degree of freedom system compared with *ode* integration for each of them. To have more accurate comparison, a 15 degree of freedom which was studied by Kumar and Sankar [24] with time duration 0.8 second studied for all five numerical methods. Newmark Beta when $\beta = \frac{1}{8}$ and $\gamma = \frac{1}{2}$ has the most accurate response.

At the end, in chapter 7, a new approach which is based on the acceleration instead of displacement is studied. In this new method, velocity and displacement described based on the acceleration. Fox-Euler, Wilson *theat* , Newmark Beta are the numerical methods which studied in this way. Results show that Newmark Beta is more accurate and stable with this approach. Also, output of Newmark Beta when using acceleration and displacement is same. However, the result for Fox-Euler and Wilson *theat* are different.

8.2 Future Work

As is shown, the DT-TMM is an efficient and accurate method that can be used in modeling of different systems. However discretizing is an issue in this way which may be lead to instability.

DT-TMM because of the numerical integration method is very sensitive to time step, especially in complicated systems. Finding a way to decrease this problem is a very interesting topic that can be studied in the future. Also, there are other methods that their compatibility with TMM needs to be studied.

REFERENCES

1-Pestel, E. C. and Leckie, F. A., Matrix Methods in Elastomechanics. New York: McGraw Hill, 1963.

2- Book, W. J., Modeling, Design and Control of Flexible Manipulator Arms. PhD thesis, Massachusetts Institute of Technology, Apr. 1974.

3- Book, W. J., Majette, M. W., and Ma, K., "Distributed systems analysis package (dsap) and its application to modeling flexible manipulators," tech. rep., NASA, July 1979. Final Report, Subcontract No. 551 to Charles Stark Draper Laboratory, NASA Contract NAS9-13809.

4- Book, W. J., Majette, M. W., and Ma, K., "Frequency domain analysis of the space shuttle manipulator arm and its payloads," tech. rep., NASA, February 1981. Final Report, Subcontract No. 586 to Charles Stark Draper Laboratory, NASA Contract NAS9-13809.

5- R. Krauss, "An Improved Technique for Modeling and Control of Flexible Structures," 2006.

6- R. W. Krauss and W. J. Book, "Transfer Matrix Modeling of Systems With Noncollocated Feedback," Journal of Dynamic Systems, Measurement, and Control, vol. 132, 2010.

7- R. W. Krauss and W. J. Book, "A Python Module for Modeling and Control Design of Flexible Robots," IEEE, 2007.

8- J.-F. Yu, H.-C. Lien and B. P. Wang, "Exact Dynamic Analysis of Space Structures Using Timoshenko Beam Theory," AIAA Journal, vol. 42, no. 4, pp. 833-839, 2004.

9- J. Xiao-jun and F. Shi-dong, "Analysis of the flexural vibration of ship's tail shaft by transfer matrix method," Journal of marine science and application, vol. 7, pp. 179-183, 2008.

10- B. He, X. Rui and H. Zhang, "Transfer Matrix method For Natural Vibration Analysis of Tree System," Mathematical Problems in Engineering, vol. 2012, 2012.

11- X. Rui, G. Wang, Y. Lu and L. Yun, "Transfer Matrix Method For Linear Mulitbody System," Multibody System Dynamics, vol. 19, pp. 179-207, 2008.

12- R. Krauss, O. Brüls and W. Book, "Two Competing Linear Models For Flexible Robots: Comparison, Experimental Validation, and Refinement," in American Control

Conference, Portland, 2005.

13- C. Guo-long and N. Wu, "Dynamic charachteristics of a WPC-comparison of transfer matrix method and FE method," Journal of Marine Science and Application, vol. 2, no. 2, 2003.

14- Mihail Boiangiu1 , Valentin Ceausu1 and Costin D Untaroiu2 "A transfer matrix method for free vibration analysis of Euler-Bernoulli beams with variable cross section"

15- M. A. Dokanish, "A new approach for plate vibration: combination of transfer matrix and finite element technique," Journal of Mechanical Design, vol. 94, pp. 526–530, 1972

16- J. W. Zu and Z. Ji, "An improved transfer matrix method for steady-state analysis of nonlinear rotorbearing systems," Journal of Engineering for Gas Turbines and Power, vol. 124, no. 2, pp. 303–310, 2002.

17- A. M. Ellakany, K. M. Elawadly, and B. N. Alhamaky, "A combined transfer matrix and analogue beam method for free vibration analysis of composite beams," Journal of Sound and Vibration, vol. 277, no. 4-5, pp. 765–781, 2004

18- Mitao Ohga,Tsunemi Shigematsu, and Takashi Hara, "Combined finite element-transfer matrix method," J. Eng. Mech., 1984, 110(9): pp. 1335-1349.

19- Bao Rong, Xiaoting Rui ,GuopingWang "Modified Finite Element Transfer Matrix Method for Eigenvalue Problem of Flexible Structures" Journal of Applied Mechanics Copyright © 2011 by ASME MARCH 2011, Vol. 78 / 021016-1

20-U. Lee, "Vibration analysis of one-dimensional structures using the spectral transfer matrix method," Engineering Structures, vol. 22, no. 6, pp. 681–690, 2000.

21- S.-C. Hsieha, J.-H. Chenb, and A.-C. Lee, "A modified transfer matrix method for the coupling lateral and torsional vibrations of symmetricrotor-bearing systems," Journal of Sound and Vibration, vol. 289, pp. 294–333, 2006.

22- G. C. Horner andW. D. Pilkey, "The riccati transfer matrix method," Journal ofMechanical Design , vol. 1, pp. 297–302, 1978.

23- Y. M. Huang and C. D. Horng, "Extended transfer matrix method with complex numbers for branched torsional systems," Journal of Vibration and Control, vol. 7, no. 2, pp. 155–166, 2001.

24- A. S. Kumar and T. S. Sankar, "A New Transfer Matrix Method For Response Analysis Of Large Dynamic Systems," Computers and Structures Journal, vol. 23, no. 4,

pp. 545-552, 1986.

25- X. Rui, B. He, Y. Lu, W. Lu and G. Wang, "Discrete Time Transfer Matrix Method for Mutibody Sytem Dynamics," Multibody System Dynamics, vol. 14, pp. 317-344, 2005.

26- X. Rui, G. Wang, L. Yun, B. He, F. Yang and B. Rong, "Advances in Transfer Matrix Method Of Multibody System," in ASME 2009 International Design Engineering Technical Conferences Computers and Information in Engineering Conference, San Diego, 2009.

27- B. Rong, X. Rui, G. Wang and F. Yang, "Discrete Time Transfer Matrix Method for Dynamics of Multibody System with Real-Time Control," Journal of Sound and Vibration, vol. 329, pp. 627-643, 2010.

28- B. He, G. Wang and X. Rui, "Riccati Discrete Time Transfer Matrix Method for Elastic Beam Undergoing Large Overall Motion," Multibody System Dynamics, vol. 18, pp. 579-598, 2007.

APPENDIX A

DT-TMM (One Degree of Freedom)

```
Created on Sun Feb 21 13:14:22 2016

@author: Vahid Alizadehyazdi
"""

from scipy import *
import numpy

"""""DOF: Degree of Freedom(one mass, one damper, one spring
   Usd: transfer matrix of spring / damper
   Um: Transfer matrix of mass
   Uf: Transfer matrix of applied force
   Usys: Transfer matrix of the system   """""
Usd=numpy.zeros((3,3))
Um=numpy.zeros((3,3))
Uf=numpy.zeros((3,3))
Usys=numpy.zeros((3,3))


""""""""""Constatnt coefficients of Newmark Beta method""""""""""
beta = 1.0 / 6.0
gamma = 0.5

""""""""""m: mass,  k: spring constant,  c: damping coefficient, f: force""""""""""

m=array([2.0])
k=array([12.0])
c=array([3.0])
f=array([1.0])

""""""""""T: time, dt: time step , N: numebr of steps """""""""""
T=10
dt=.02
N = int(T/dt)

""""""""""defining arrays of displacement, velocity and acceleration   """""""""""
x=zeros(N)
xdot=zeros(N)
xddot=zeros(N)

""""""""""" DT-TMM  """""""""""
for p in range(1,N):

    """" Calculation of A,B,D and E coefficients for each time step""""""
    A = 1.0/(beta*dt**2)
    B = -1.0/(beta*dt**2)*(x[p-1] + dt*xdot[p-1] + (0.5-beta)*dt**2*xddot[p-1])
    E = xdot[p-1] + dt*((1.0-gamma)*xddot[p-1]+gamma*B)
    D = gamma/(beta*dt)


    """""Calculation of Um for each time step"""""
```

```
    for i in range(3):
       for j in range(3):
          if i==j:
           Um[i][j]=1
       Um[1][0]=m[0]*A
       Um[1][2]=m[0]*B

    """"""Calciulation of Usd for each time step""""""
    for i in range(1,3):
       for j in range(1,3):
          if i==j:
            Usd[i][j]=1
       Usd[0][0]=(k[0]/(k[0]+c[0]*D))
       Usd[0][1]=(1.0/(k[0]+c[0]*D))
       Usd[0][2]=(-c[0]*(E)/(k[0]+c[0]*D))

    """"Calculation of Uf for each time step""""""
    for i in range(3):
        for j in range(3):
            if i==j :
               Uf[i][j]=1.0
        Uf[1][2]=-f[0]

    """""""""Calculation of Usys and Fbase for each time step"""
    v=dot(Uf,Um)
    Usys=dot(v,Usd)
    Fbase= - Usys[1][2]/Usys[1][1]

    """ Calculation of x, xdot and xddot for each time step""""""
    x[p] =  (Usys[0][1]*Fbase) + Usys[0][2]
    xdot[p] = D*x[p]+E
    xddot[p]=A*x[p]+B
""" Plotting the graph""""""
figure(1)
clf()
t = arange(0.0, T, dt)
plot(t,x)
plt.ylabel('x')
plt.xlabel('time')
show()
```

APPENDIX B

ODE Integration (n Degree of Freedom)

```
Created on Sun Feb 14 21:59:04 2016

@author: valizad
"""

from scipy.integrate import odeint
import numpy
import pylab
"""Input of this program:
1:Degrre of freedom,
2:initial conditions,
3:array of m, array of k, array of c ,and array of f
output of this program: response of last element versus time
(also can have response of each element versus time)"""


""" DOF = Degree of freedom"""
DOF=1

""" n=number of states"""
n= (2*DOF)

""" x-dot = Ax+B
 Matrix B is a n*1 matrix which shows applid force to elements
 Matrix A is a n*n matrix which includes 4 matices(A1,A2,A3,A4) of DOF*DOF
 A=[[A1, A2],[A3,A4]],
 A1: a zero DOF*DOF matrix
 A2: a I DOF*DOF matrix
 A3: a K(Spring Constant) DOF*DOF matrix
 A4: a C(Damper Constant) DOF*DOF matrix
"""
A=numpy.zeros((n,n))
B=numpy.zeros((n,1))
"""  m is the mass of the elements (kg)"""
m=array([2.0])

"""  k is the stiffness coefficients (N/m)"""

k=array([12])

"""  c is the damping coefficients (N*s/m)"""

c=array([3])
"""f is the force applied to the element (N)"""

f=array([1])

""" Calcaulation of matrix B """

for i in range(DOF):
    B[i][0]=0.0
```

```
for i in range(DOF,n):
    B[i][0]=f[i-DOF]/m[i-DOF]

""" Calculation of matrix A1"""

for i in range(DOF):
      for j in range(DOF):
         A[i][j]= 0

""" Calculation of matrix A2"""

for i in range (DOF):
      for j in range(DOF,n):
          if j-i==DOF :
              A[i][j]=1

""" Calculation of matrix A3"""

for i in range (DOF,n):
      for j in range(DOF-1):
          if i-j==DOF :
              A[i][j]=-(k[j]+k[j+1])/m[i-DOF]
A[(n-1)][(DOF-1)]=-k[(DOF-1)]/m[DOF-1]

p=DOF
q=1
for i in range (DOF-1):
       A[p][q]=k[q]/m[p-DOF]
       p=p+1
       q=q+1

p=DOF+1
q=0
for j in range (DOF-1):
       A[p][q]=k[q+1]/m[p-DOF]
       p=p+1
       q=q+1

""" Calculation of matrix A4"""

for i in range (DOF,n-1):
      for j in range(DOF,n-1):
          if i==j :
              A[i][j]=-(c[j-DOF]+c[j+1-DOF])/m[i-DOF]
A[n-1][n-1]=-c[DOF-1]/m[DOF-1]

p=DOF
q=DOF+1.0
for i in range (DOF-1):
       A[p][q]=c[q-DOF]/m[p-DOF]
       p=p+1
       q=q+1

p=DOF+1
q=DOF
```

```
for j in range (DOF-1):
       A[p][q]=c[q-DOF+1]/m[p-DOF]
       p=p+1
       q=q+1


x=zeros(n)
z=zeros(n)
state=zeros(n)
def integ(state,t):
    """unpack the state vector"""
    for  i in range(n):
     z[i]=state[i]

    """compute state derivatives"""
    p=zeros(n)
    for i in range(n):
      for j in range(n):
        x[i]=A[i][j]*z[j]
        p[i]=p[i]+x[i]

      x[i] =p[i]

    for i in range(n):
        x[i]=x[i]+B[i][0]
    """return the state derivatives"""
    v=zeros(n)
    for i in range(n):
        v[i]=x[i]
    return v
"""initial condition"""

state0 = zeros(n)
t = arange(0.0, 10.0, 0.02)

state = odeint(integ, state0, t)

""" ploting the response of last element"""

plt.plot(t,state[:,(DOF-1)],
         'red',
         linestyle=':',
         linewidth=1,
         label='odeint')
legend = plt.legend(loc='upper right', shadow=True, fontsize='medium')


xlabel('TIME (sec)')
ylabel('Response(x) ')
title('DOF=2')
```

APPENDIX C

DT-TMM n Degree of Freedom

```
Created on Tue Mar 01 12:11:48 2016

@author: Vahid Alizadehyazdi
"""

from scipy import *
import control
import numpy

"""""""""""""""Degree of Freedom""""""""""""""""""""
DOF=15

""""
Method 0 = Wilson Theat
Method 1 = Fox-Euler
Method 2 = Newmark Beta Method
Method 3 = Houbolt
Method 4 =  Park Stiffly"""""

Method=2
"""Usd: transfer matrix of spring / damper
   Um: Transfer matrix of mass
   Uf: Transfer matrix of applied force
   Usys: Transfer matrix of the system   """""""""""""""""""""""""""""""
Usd=numpy.zeros((3,3))
Um=numpy.zeros((3,3))
Uf=numpy.zeros((3,3))
Usys=numpy.zeros((3,3))

"""""""""""""""""""""""""""""""""""""""""""""""""""
beta = 1.0 / 4.0
gamma = 0.5
theta=1.4
m=array([10.0,\
                9.0, \
                8.0, \
                7.0, \
                6.0, \
                6.0, \
                5.5, \
                5.0, \
                4.5, \
                4.0, \
                3.5, \
                4.0, \
                2.5, \
                2.0, \
                2.0, \
                ])

k=array([55000.0,\
                50000.0, \
                45000.0, \
```

```
                40000.0, \
                40000.0, \
                35000.0, \
                30000.0, \
                25000.0, \
                20000.0, \
                15000.0, \
                10000.0, \
                9000.0, \
                8000.0, \
                7000.0, \
                6000.0, \
                ])

c=array([0.0,\
                0.0, \
                0.0, \
                0.0, \
                0.0, \
                0.0, \
                0.0, \
                0.0, \
                0.0, \
                0.0, \
                0.0, \
                0.0, \
                0.0, \
                0.0, \
                ])

f=array([0.0,\
                0.0, \
                0.0, \
                0.0, \
                0.0, \
                0.0, \
                0.0, \
                0.0, \
                0.0, \
                0.0, \
                0.0, \
                0.0, \
                0.0, \
                0.0, \
                1.0, \
                ])
"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
T=0.8
dt=0.02
t = arange(0.0,T, dt)
N = len(t)
"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
A=zeros(DOF+1)
B=zeros(DOF+1)
```

```
E=zeros(DOF+1)
D=zeros(DOF+1)
************************************************************
x=numpy.zeros((DOF,N))
xdot=numpy.zeros((DOF,N))
xddot=numpy.zeros((DOF,N))
************************************************************
h=np.zeros((DOF,3,3))
************************************************************
for p in range(1,N):
    Usys_Total=np.eye(3)
    if Method==0:
        for n in range(DOF):
            A[n+1] = 6.0/(theta*dt)**2
            B[n+1] = -6.0/(theta*dt)**2 * (x[n][p-1] + theta*dt*xdot[n][p-1]
            + ((theta*dt)**2) /3.0 * xddot[n][p-1] )
            E[n+1] = -3.0/(theta*dt) * (x[n][p-1] + 2*theta*dt/3.0 * xdot[n][p-1]
            + ((theta*dt)**2) /6.0 * xddot[n][p-1] )
            D[n+1] = 3.0/(theta*dt)
    if Method==1:
        for n in range(DOF):
            A[n+1] = 2.0/(dt**2)
            B[n+1] = -2.0/(dt**2)*(x[n][p-1] + dt*xdot[n][p-1])
            E[n+1] = -(2.0/dt*x[n][p-1] + xdot[n][p-1])
            D[n+1] = 2.0/dt
    if Method==2:
        for n in range(DOF):
            A[n+1] = 1.0/(beta*dt**2)
            B[n+1] = -1.0/(beta*dt**2)*(x[n][p-1] + dt*xdot[n][p-1]
            + (0.5-beta)*dt**2*xddot[n][p-1])
            E[n+1] = xdot[n][p-1] + dt*((1.0-gamma)*xddot[n][p-1]+gamma*B[n+1])
            D[n+1] = gamma/(beta*dt)
    if Method==3:
        for n in range(DOF):
            if p==1:
                A[n+1] = 6.0/(dt**2)
                B[n+1] = -2.0/(dt**2)*(3*x[n][p-1]+3*dt*xdot[n][p-1]+
                dt**2*xddot[n][p-1])
                E[n+1] = -1.0/(2*dt)*(6*x[n][p-1]+4*dt*xdot[n][p-1]+
                dt**2*xddot[n][p-1])
                D[n+1] = 3.0/dt

            elif p==2:
                A[n+1] = 2.0/(dt**2)
                B[n+1] = -1.0/(dt**2)*(4*x[n][p-1]-2*x[n][p-2]+2*dt**2*xddot[n][p-2])
                E[n+1] = -1.0/(6*dt)*(16*x[n][p-1]-5*x[n][p-2]+dt**2*xddot[n][p-2])
                D[n+1] = 11.0/(6*dt)

            else:
                A[n+1] = 2.0/(dt**2)
                B[n+1] = -1.0/(dt**2)*(5*x[n][p-1]-4*x[n][p-2]+x[n][p-3])
                E[n+1] = -1.0/(6*dt)*(18*x[n][p-1]-9*x[n][p-2]+2*x[n][p-3])
                D[n+1] = 11.0/(6*dt)

    if Method==4:
```

```
            for n in range(DOF):
                if p<=2:
                    A[n+1] = 2.0/(dt**2)
                    B[n+1] = -2.0/(dt**2)*(x[n][p-1] + dt*xdot[n][p-1])
                    E[n+1] = -(2.0/dt*x[n][p-1] + xdot[n][p-1])
                    D[n+1] = 2.0/dt

                else:
                    A[n+1] = 100.0/(36*dt**2)
                    B[n+1] = 1.0/(36*dt**2) * (-150.0 * x[n][p-1] + 60.0 * x[n][p-2]
                    - 10.0 * x[n][p-3]) + 1.00/(6*dt) * (-15.0 * xdot[n][p-1] + 6.0 * xdot[
                    - xdot[n][p-3])
                    E[n+1] = 1.0/(6*dt) * (-15.0 * x[n][p-1] + 6.0 * x[n][p-2] - x[n][p-3])
                    D[n+1] = 10.0/(6*dt)

        for n in range(DOF):
            for i in range(3):
                for j in range(3):
                    if i==j:
                        Um[i][j]=1
            Um[1][0]=m[n]*A[n+1]
            Um[1][2]=m[n]*B[n+1]

            for i in range(1,3):
                for j in range(1,3):
                    if i==j:
                        Usd[i][j]=1
            Usd[0][0]=((k[n]+c[n]*D[n])/(k[n]+c[n]*D[n+1]))
            Usd[0][1]=(1.0/(k[n]+c[n]*D[n+1]))
            Usd[0][2]=-c[n]*(E[n+1]-E[n])/(k[n]+c[n]*D[n+1])

            for i in range(3):
                for j in range(3):
                    if i==j :
                        Uf[i][j]=1.0
            Uf[1][2]=-f[n]

            v=dot(Uf,Um)
            Usys=dot(v,Usd)
            Usys_Total= dot(Usys,Usys_Total)
            h[n]=Usys_Total

            if n==DOF-1:
                Fbase= - (h[n][1][2])/(h[n][1][1])

                for n in range(DOF):
                    x[n][p] =  (h[n][0][1]*Fbase) + h[n][0][2]
                    xdot[n][p] = D[n+1]*x[n][p]+E[n+1]
                    xddot[n][p]=A[n+1]*x[n][p]+B[n+1]

plt.plot(t,x[ DOF-1 , : ],
         'orange',
         linestyle='-',
         linewidth=0.5,
         label='Beta=1/4')
```

```
legend = plt.legend(loc='upper right', shadow=True, fontsize='medium')
plt.xlabel('x')
plt.ylabel('time')
show()
```

APPENDIX D

DT-TMM n Degree of Freedom Based on the Acceleration

```
Created on Fri Mar 11 11:07:18 2016

@author: valizad
"""

from scipy import *
import control
import numpy

"""""""""""""""Degree of Freedom"""""""""""""""""""
DOF=15

""""
Method 0 = Wilson Theat
Method 1 = Fox-Euler
Method 2 = Newmark Beta Method""""""


Method=0
"""'"Usd: transfer matrix of spring / damper
   Um: Transfer matrix of mass
   Uf: Transfer matrix of applied force
   Usys: Transfer matrix of the system   """""""""""""""""""""""""""""""
Usd=numpy.zeros((3,3))
Um=numpy.zeros((3,3))
Uf=numpy.zeros((3,3))
Usys=numpy.zeros((3,3))

"""""""""""""Newmark Beta method""""""""""""""""""""""""""""""""""""
beta = 1.0 / 8.0
gamma = 0.5
theta=1.03
m=array([10.0,\
                9.0, \
                8.0, \
                7.0, \
                6.0, \
                6.0, \
                5.5, \
                5.0, \
                4.5, \
                4.0, \
                3.5, \
                4.0, \
                2.5, \
                2.0, \
                2.0, \
                ])

k=array([55000.0,\
                50000.0, \
                45000.0, \
                40000.0, \
```

```
                40000.0, \
                35000.0, \
                30000.0, \
                25000.0, \
                20000.0, \
                15000.0, \
                10000.0, \
                9000.0, \
                8000.0, \
                7000.0, \
                6000.0, \
                ])

c=array([0.0,\
                0.0, \
                0.0, \
                0.0, \
                0.0, \
                0.0, \
                0.0, \
                0.0, \
                0.0, \
                0.0, \
                0.0, \
                0.0, \
                0.0, \
                0.0, \
                0.0, \
                ])

f=array([0.0,\
                0.0, \
                0.0, \
                0.0, \
                0.0, \
                0.0, \
                0.0, \
                0.0, \
                0.0, \
                0.0, \
                0.0, \
                0.0, \
                0.0, \
                0.0, \
                1.0, \
                ])
""""""""""""""""""""""""""""""""""""""""""""""""""""""
T=0.8
dt=0.02
t = arange(0.0,T, dt)
N = len(t)
""""""""""""""""""""""""""""""""""""""""""""""""""""""
A=zeros(DOF+1)
B=zeros(DOF+1)
E=zeros(DOF+1)
```

```
D=zeros(DOF+1)
""""""""""""""""""""""""""""""""""""""""""""""""""""""
x=numpy.zeros((DOF,N))
xdot=numpy.zeros((DOF,N))
xddot=numpy.zeros((DOF,N))
""""""""""""""""""""""""""""""""""""""""""""""""""""""
h=np.zeros((DOF,3,3))
""""""""""""""""""""""""""""""""""""""""""""""""""""""
for p in range(1,N):
    Usys_Total=np.eye(3)

    if Method==0:
        for n in range(DOF):
            A[n+1] = (dt*theta)/2.0
            B[n+1] = ((dt*theta)/2.0)* (xddot[n][p-1]) + xdot[n][p-1]
            E[n+1] = (((dt**theta)**2)/3.0) * (xddot[n][p-1])
            + (theta*dt * xdot[n][p-1])+x [n][p-1]
            D[n+1] = ((dt**theta)**2)/6.0

    if Method==1:
        for n in range(DOF):
            A[n+1] = dt
            B[n+1] = xdot[n][p-1]
            E[n+1] = x[n][p-1] + (dt * xdot[n][p-1])
            D[n+1] = (dt**2)/2
    if Method==2:
        for n in range(DOF):
            A[n+1] = gamma*dt
            B[n+1] = xdot[n][p-1] + dt*(1-gamma)*xddot[n][p-1]
            E[n+1] = x[n][p-1] + dt*xdot[n][p-1]
            + (0.5 - beta) * xddot[n][p-1] * (dt**2)
            D[n+1] = beta*(dt**2)

    for n in range(DOF):
        for i in range(3):
            for j in range(3):
                if i==j:
                    Um[i][j]=1.0
        Um[1][0]=m[n]


        for i in range(1,3):
            for j in range(1,3):
                if i==j:
                    Usd[i][j]=1
            Usd[0][0]=(k[n]*D[n]+c[n]*A[n])/(k[n]*D[n+1]+c[n]*A[n+1])
            Usd[0][1]=(1.0/(k[n]*D[n+1]+c[n]*A[n+1]))
            Usd[0][2]=
            (c[n]*(B[n]-B[n+1])+k[n]*(E[n]-E[n+1])) / (k[n]*D[n+1]+c[n]*A[n+1])

        for i in range(3):
            for j in range(3):
                if i==j :
                    Uf[i][j]=1.0
            Uf[1][2]=-f[n]
```

```
            v=dot(Uf,Um)
            Usys=dot(v,Usd)
            Usys_Total= dot(Usys,Usys_Total)
            h[n]=Usys_Total

            if n==DOF-1:
                Fbase= - (h[n][1][2])/(h[n][1][1])

                for n in range(DOF):
                    xddot[n][p] =  (h[n][0][1]*Fbase) + h[n][0][2]
                    x[n][p] = D[n+1]*xddot[n][p]+E[n+1]
                    xdot[n][p]=A[n+1]*xddot[n][p]+B[n+1]

plt.plot(t,x[ DOF-1 , : ],
          'orange',
          linestyle=':',
          linewidth=2,
          label='Theta=1.03 (acceleration)')
legend = plt.legend(loc='upper right', shadow=True, fontsize='small')
plt.xlabel('x')
plt.ylabel('time')
show()
```