A FAST, NON-LINEAR, FINITE ELEMENT SOLVER FOR EARTHQUAKE RESPONSE OF BUILDINGS

A thesis

submitted in fulfilment

of the requirements of the Degree

of

Master of Mechanical Engineering

in the

University of Canterbury

by

D. R. Hewett

ABSTRACT

Design of buildings in earthquake regions requires that the building is made to withstand certain large earthquake magnitudes with a degree of permanent, but energy absorbing, damage. To accurately determine the behaviour of the building while damage occurs, a non-linear analysis must be used as these effects are non-linear. These computations are often very slow as the building's response must be calculated many times a second.

This thesis seeks to find a faster alternative to the Newmark-β and similar numerical integration schemes commonly used in non-linear seismic structural analysis. Faster computation would enable rapid simulation thus speeding up the design process. It would also allow large Monte Carlo analyses to be done to improve research analysis and allow designers to better account for variability in materials, construction, soil site and other factors that can significantly affect response.

For the purposes of this investigation simple two node finite elements were used. The non-linear component consists of the well-accepted Ramberg-Osgood hysteresis model. The alternative approach used in this thesis is to solve non-linear first order differential equations using a Runge-Kutta based solution. This approach, with added new computational methods, should be more efficient than directly solving the second order equation of motion with Newmark-β.

Different test cases were run to establish performance differences in a variety of potential user cases. These cases involved testing different models against both real earthquake data and synthetic input accelerations. In all test cases, the Newmark-β solution yielded the same results as the new solution, as long as a small enough time step was used. When a small time step was used and the results agreed, the new solution was much faster than the Newmark-β solution.

In particular, the new numerical solution approach was significantly faster than Newmark-β when the accuracy demanded was 1% or less. As the tolerance was tightened the advantage of the new solution increased exponentially. From this project a set of MATLAB scripts has been created that will reproduce the results given and can also be used to analyse other building models. The overall approach used is also entirely generalisable.

ACKNOWLEDGEMENTS

I would like to sincerely thank and acknowledge the professionals who supported this thesis.

Firstly thank you to Professor Geoff Chase, my principal advisor, for his help and guidance throughout the thesis. His flexibility in providing help has been invaluable in both the research and writing phases. We must have used every form of modern communication (email, Skype, text), but it worked!

I would like to thank Dr Chris Hann for his specialist knowledge and patience in helping develop the algorithm in this thesis. It was thrilling to apply modern maths techniques to this engineering problem.

Also, I would like to thank Professor Athol Carr for the technical discussions, especially about his RUAUMOKO software. This was very useful in understanding the existing methods used in non-linear earthquake analysis.

Finally, I would like to thank my parents who supported and encouraged me during the research and writing of this thesis.

Table of Contents

A Fast, Non-Linear, Finite Element Solver for Earthquake Response of Buildings 1

Abstract 2

Acknowledgements 4

Table of Contents 5

List of Figures 8

1 Introduction 1

1.1 Motivation 1

1.2 Objectives and Scope 2

1.3 Literature Survey 3

1.4 Newmark-β Solution Methods 4

1.5 Proposed ODE Solution Approach 6

1.6 Ramberg-Osgood Hysteresis 6

1.7 Chapter Summary 9

2 Derivation of Numerical Solution Methods 10

2.1 Dynamic-Equilibrium Condition 10

2.2 Elements 12

2.3 Newmark-β Derivation 14

2.4 Incremental Equilibrium 17

2.5 Derivation of First Order Equations for ODE Solution 19

2.6 Continuous Formulation Check 22

2.7 Chapter Summary 23

3 Methods 25

3.1 Code Flowcharts 25

Run Solution Function 26

Define Problem 26

Create System to be Solved 26

Process Results 26

3.2 Input Acceleration 35

3.3 Test Cases 36

3.4 Test System 39

4 Results 40

4.1 Test Cases 40

4.2 Chapter Summary 58

5 Summary and Conclusions 59

6 Future Work 60

7 References 61

APPENDIX A: Manual 64

A.1 Meshing 64

A.2 Solving with Newmark-β 67

A.3 Solving with the ODE algorithm 68

A.4 Post processing 68

APPENDIX B: Test Case Code........70

B.1 Single Element........70

B.2 Two Elements........72

B.3 Single Floor........74

B.4 SAC-3 Building Model........76

LIST OF FIGURES

Figure 1.1: Ramberg Osgood Hysteresis MODEL (Carr, 2008) 8

Figure 2.1: Depiction of stiffness matrix at the start of the timestep (K_original), the tangent stiffness at this timestep (K_tangent) and the stiffness at this timestep (K_secant). ...11

Figure 2.2: 2D beam element 12

Figure 2.3: Acceleration History 14

Figure 3.1: Code Overview Flowchart 26

Figure 3.2: Flowchart of Newmark-β Code 29

Figure 3.3: Flowchart for the Outer ODE function 31

Figure 3.4: Inner ODE Function Flowchart 33

Figure 3.5: Synthetic Input 35

Figure 3.6: El Centro Earthquake record (North-South Component) 36

Figure 3.7: Single Element Test Case 37

Figure 3.8: Two Element Test Case 37

Figure 3.10: SAC-3 Structure Mesh 38

Figure 3.9: Single Floor Test Case 38

Figure 4.1: Horizontal Displacement of First Storey Nodes. Ode45 deviates from the others which all overlap 41

Figure 4.2: Single Element Results with synthetic input of Figure 3.5 the solid red line shows the time for the ODE solution 43

Figure 4.3: Single Element Results with El Centro ground motion input the solid red line shows the time for the ODE solution 45

Figure 4.4: Demonstration of the need for turning point accuracy Using a single element as in Figure 3.7 with the input acceleration vector given in Figure 4.6 46

Figure 4.5: Close up of the first turning point from Figure 4.4 47

Figure 4.6: Turning point demonstration input acceleration vector 47

Figure 4.7: Two elements results with synthetic input of Figure 3.5 the solid red line shows the time for the ODE solution..48

Figure 4.8: Two Elements results with El Centro ground motion input the solid red line shows the time for the ODE solution..49

Figure 4.9: Demonstration of similar but incorrect Newmark-β result 51

Figure 4.10: Single Floor results with synthetic input of Figure 3.5 the solid red line shows the time for the ODE solution..52

Figure 4.11: Single Floor results with El Centro ground motion input the solid red line shows the time for the ODE solution..53

Figure 4.12: SAC3 results with synthetic input of Figure 3.5 the solid red line shows the time for the ODE solution..54

Figure 4.13: SAC3 results with El Centro ground motion input the solid red line shows the time for the ODE solution...55

Figure 4.14: Ramberg Osgood Hysteresis Model (Carr, 2008)...56

Figure 4.15: Demonstration of the Effect of yield force on error for Newmark-β..................57

1 Introduction

1.1 Motivation

Currently non-linear earthquake analysis is mostly conducted only at research institutes, rather than in engineering consultancies. However, non-linear seismic response analysis is important for determining, accurately, the response of critical buildings and infrastructure. There are two reasons for this outcome. One is the complexity of such analysis. The second is the computational time required for results that can be trusted. In addition, while many software packages exist, results can be highly variable and difficult to verify, casting doubt on the tools and approach. This thesis explores an alternative method of doing non-linear time-history analysis on a building to make the process faster while maintaining accuracy compared to existing methods.

Computationally, non-linear dynamic seismic analysis of buildings and critical infrastructure is typically done with a Newmark-β solution or similar numerical integration approach. This numerical solution is simple in development, but does not represent the current way other science disciplines solve differential equations. Other methods have existed for decades, but have been avoided in engineering because of the added complexity in development or derivation for sometimes minimal return in improved accuracy.

If an alternative could be developed that would give trustworthy, accurate results much more quickly then it would allow for more test cases to be analysed in design. More analyses and productivity would give a designer a better idea of the building's potential behaviour in an earthquake, as well as a greater sense of sanity-checking and trust in the outcomes. In an extreme case, it would be desirable to run a Monte Carlo analysis on building designs to account for variations and random real world differences to the design to ensure its robustness. Finally, and more directly, seismic life safety is a function, in large events, of non-linear structural response. Thus, life-safety and effective, productive non-linear analyses are inherently linked.

Currently, methods for fast numerical solution of dynamic non-linear earthquake responses do not really exist. Most non-linear numerical solutions are suited to vehicle crash analysis, which involves very large deformations and high strain rate over a very short time span. In non-linear, dynamic earthquake analysis there can be large deformations but they occur over a relatively longer time. However, as a portion of an entire earthquake record, these highly non-linear periods can be quite small. What is missing from current numerical solution approaches is the ability to both analyse the simple linear parts of the response quickly, and the complex non-linear parts accurately. Such a separation of tasks could dramatically speed up the numerical solution.

1.2 Objectives and Scope

The first objective was to create a standard non-linear Newmark-β solution in MATLAB. This solution will serve as the gold standard comparator for any alternative method in terms of accuracy and speed. This numerical solution was written in MATLAB code, rather than using an existing compiled program (like ANSYS(TM) or RUAUMOKO(TM)) to make speed comparisons meaningful. Hence, it forms the foundation of the comparisons made in this thesis. Note that while MATLAB can be relatively slow compared to optimised C or FORTRAN, the speed increases of these codes should be relatively unchanged for any given approach developed here. Thus, any improvements should carry through to optimised code.

Secondly a faster alternative to this standard approach was to be developed and tested on a series of example problems. Particular attention is paid to speed increases. Note that all numerical solutions were optimised as much as possible using MATLAB code alone, so that comparisons were valid and fair. Finally, while MATLAB was used, a further 10-100x speed increase might be expected with optimised C or FORTRAN code and processor-specific compilation.

To simplify the solutions and methods developed, only one element type and only one hysteretic model were used. However, the methods developed in this thesis can be readily extended to include other elements and hysteresis models. In this thesis, this breadth was not a main focus, as the primary outcomes were methods development and demonstrating significant reductions in computational cost (increase in speed).

As noted, further absolute performance gains can be had by using a compiled language, rather than MATLAB. This approach would have taken significantly longer to code and is not likely to make a large difference in the relative speed comparisons obtained. Both numerical solutions derived here, as designed and compared, would likely benefit similarly from using a faster language. Based on the results in this thesis, a case for developing such a numerical solution could be made.

1.3 Literature Survey

Numerical solution of the non-linear dynamic response of structures by step-by-step numerical integration methods is well presented in “Dynamics of Structures”(Clough & Penzien, 1975) but the original methods first appeared in a paper by Nathan M. Newmark in 1959 (Newmark, 1959). Step-by-step methods must be used in the non-linear case as the principle of linear superposition does not apply. Any non-linearity in the response invalidates other linear analyses, such as frequency domain or modal domain approaches. The book presents both constant average and linear acceleration Newmark-β schemes, although several other forms and extensions exist.

The ordinary differential equation (ODE) numerical solution recommended by the MATLAB help files(MathWorks, 2010) is a Runge-Kutta(Dormand & Prince, 1980) based function. This Runge-Kutta solution has been found to be efficient at solving most ODE problems when compared with other ODE numerical solutions for nonstiff systems(Hosea & Shampine, 1994). In particular, they also offer good accuracy and variable step times.

Similar ODE numerical solutions are used in many other fields where non-linear systems need to be solved e.g. (Hann, Chase, & Shaw, 2005). In that specific example paper, the objective is to speed up the analysis of a series of non-linear dynamic equations modelling the insulin-glucose metabolism of critically ill patients. The equations of motion used here in non-linear structural analysis can be arranged in the same fundamental form as their model so the same fundamental approach can be taken.

The non-linear component used here is Ramberg-Osgood hysteresis(Ramberg & Osgood, 1943). This hysteresis model describes the non-linear relationship between stress and strain when near, or in excess of, the material yield strength. It has been particularly useful in modelling the behaviour of metals undergoing strain hardening(Palermo & Vecchio, 2004). It is also widely used in structural analysis codes (Carr, 2008) and is well accepted in the profession.

1.4 Newmark-β Solution Methods

To solve a non-linear structural system a full time history Newmark-β or similar, numerical integration scheme is typically used. Alternatives, such as modal or frequency domain analysis, cannot be used as the principle of superposition does not hold in the non-linear case. Hence, step-by-step integration is required.

Typically, a constant average acceleration Newmark-β solution is used. This choice is made because it has an unconditionally stable time step. Hence, for relatively large time steps the solution will remain stable and can give good results if the higher modes do not make a significant contribution to the response. However, unconditionally stable does not imply unconditionally accurate, and a proper time step must be chosen. For the non-linear case, it enables good results if the non-linearity is not a significant contributor to the response and a small enough time step is used(Lu, Chung, Wu, & Lin, 2006).

In numerical integration, a time step must be selected that is small enough to properly capture the behaviour of the system. This small time step results in accurate results, but at the cost of computational speed. When a large time step is selected the results can be inaccurate and unstable, but also run much faster. Hence, there exists a fundamental trade off of speed and accuracy. In most cases a small time step is needed for only brief periods during the analysis, while a large time step is adequate for much of the rest of the analysis.

The downside to the large time steps with hysteresis curves is that turning points in the hysteretic behaviour are not accurately found. If the actual turning point where direction or stiffness changes lies in between time points when using large time steps, the Newmark-β method will not attempt to find the actual turning point, but continue on. As will be seen later, a slight change in the turning point can greatly impact the following response, and thus cause significant inaccuracy in the outcome.

Newmark-β methods are favoured in earthquake analysis for many reasons. In particular, it is a simple method and non-linear effects can be added quite easily. The downside of the method is that it is very inefficient compared with other (apparently) more complex methods. This last point illustrates the real-world trade off between complexity, efficacy and uptake that confronts many technology solutions.

1.5 Proposed ODE Solution Approach

To improve on the performance of the Newmark-β and similar second order ordinary differential equation (ODE) numerical integration schemes a generic ODE solution was selected as the foundation. Instead of directly solving the second order differential equations, as is done with Newmark- β it splits up these second order ODEs into two, coupled, first order differential equations. These resulting equations are then solved with an appropriate ODE solution. MATLAB has several such numerical solutions built in, with the Runge-Kutta based ODE45 method typically recommended as a good starting point(MathWorks, 2010).

MATLAB's ODE45 function uses an explicit Runge-Kutta (Dormand & Prince, 1980) method. The step size is variable, which means that at places in the analysis that need a small time step it can be resolved accurately, while marching through simpler regions much more quickly with a larger time step. Once a function has been created in MATLAB for a given system of equations other ODE solution functions can be used to find which has the best performance.

To write the ODE function representing the non-linear system of equations, they must first be converted into a continuous form. The Newmark-β solution has the advantage of always moving only forward in time. However, this assumption does not hold true for all ODE numerical solutions. In particular, some numerical solution methods will go back in time with a smaller step size or do multiple solutions at different times for a single step to increase accuracy(Shapine & Gordon, 1975). The derivation of this continuous form is required for this research and is given in the next chapter.

1.6 Ramberg-Osgood Hysteresis

The Ramberg-Osgood hysteresis model was originally formulated at NASA for modelling metals near their yield points(Ramberg & Osgood, 1943). The model is still useful in modelling the steel structure of buildings, which have similar behaviours. Its main features are smooth strength degradation around the yield point and strain hardening(Bruneau, Uang, & Whittaker, 1998). In structural analysis it is used to capture the plastic behaviour of beams as they undergo yielding to dissipate energy during large seismic events.

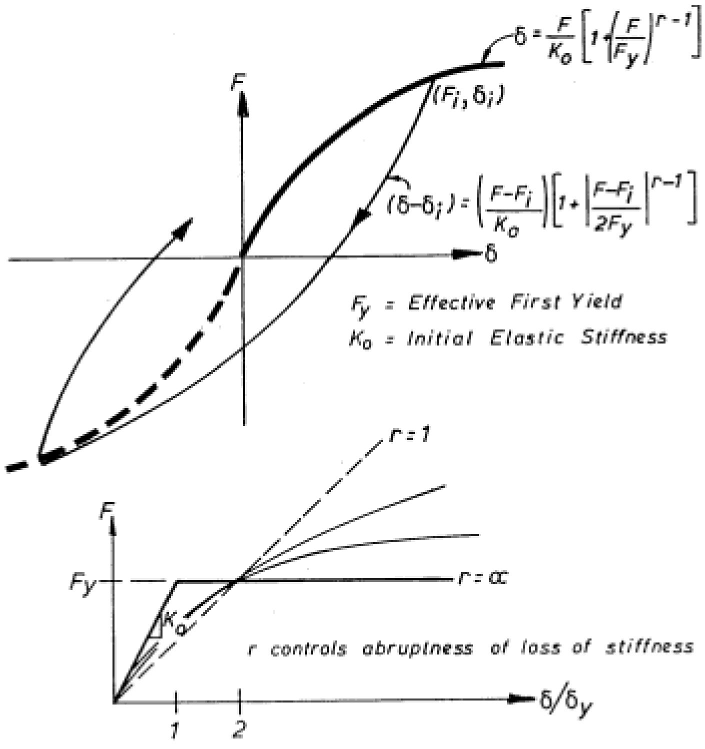
The Ramberg-Osgood hysteresis curve smoothly transitions from the un-deformed stiffness as it degrades. Then as the element is unloaded the original linear stiffness returns. This behaviour closely models the strain hardening exhibited by various metals used in structures. It also captures the same fundamental shape seen in structural push-over tests.

The behaviour of the Ramberg-Osgood method during the first loading is defined:

$$\delta = \frac{F}{K_0}\left[1 + \left|\frac{F}{F_y}\right|^{r-1}\right] \qquad 1.6.1$$

Where δ is the stress, F is the force, K_0 is the initial elastic stiffness, F_y is the effective first yield and r is the Ramberg-Osgood constant. Once the turning point (F_i, δ_i, see Figure 1.1) has been reached, the equation for the curve is defined on the return by:

$$\delta - \delta_i = \left(\frac{F - F_i}{K_0}\right)\left[1 + \left|\frac{F - F_i}{2F_y}\right|^{r-1}\right] \qquad 1.6.2$$



$\delta = \frac{F}{K_o}\left[1+\left|\frac{F}{F_y}\right|^{r-1}\right]$
(F_i, δ_i)
$(\delta-\delta_i) = \left(\frac{F-F_i}{K_o}\right)\left[1+\left|\frac{F-F_i}{2F_y}\right|^{r-1}\right]$
F
δ
F_y = Effective First Yield
K_o = Initial Elastic Stiffness
r = 1
F
Fy
$r = \infty$
K_o
r controls abruptness of loss of stiffness
1
2
δ/δ_y

FIGURE 1.1: RAMBERG OSGOOD HYSTERESIS MODEL (CARR, 2008)

Equations (1.6.1) and (1.6.2) can be rearranged for $\mathrm{K}_{tangent}$:

$$\mathrm{K}_{tangent} = \frac{\mathrm{K}_0}{1 + \left|\frac{F}{Fy}\right|^{r-1}} \qquad 1.6.3$$

For subsequent loadings $\mathrm{K}_{tangent}$ can then be defined:

$$\mathrm{K}_{tangent} = \frac{\mathrm{K}_0}{1 + \left|\frac{F - F_i}{2F_y}\right|^{r-1}} \qquad 1.6.4$$

These definitions enable calculation of the tangent stiffness matrix at any point in the analysis. Both numerical solutions use them as will be seen later.

One important fact to note from Equations (1.6.1) and (1.6.2) is that the stiffness is never the same from one time step to the next. In other hysteresis models like the bilinear model significant speed improvements can be had by knowing that the stiffness is constant in certain regions. This makes the Ramberg-Osgood model computationally expensive by comparison as the stiffness changes every time step and thus must be recalculated every time step.

1.7 Chapter Summary

The motivation of this thesis is that it would be useful to be able to conduct non-linear seismic analysis on building models in design. Currently this is not feasible as the existing numerical solutions are too slow to give trustworthy results in a timely manner.

This thesis takes the approach of using more efficient numerical integration schemes than the commonly used Newmark-β scheme. The schemes tested are implemented as MATLAB functions including a Runge-Kutta based solution. By solving the governing equations more efficiently the time taken for trustworthy results can be substantially reduced.

The non-linear component used in this thesis is a Ramberg-Osgood hysteresis model. This model is commonly used in structural seismic analysis for the frame response. This model incorporates strain hardening and does not remain constant from one time step to the next. This makes it one of the more computationally expensive hysteresis models.

2 Derivation of Numerical Solution Methods

2.1 Dynamic-Equilibrium Condition

The equations of motion for a system can be written in general matrix-vector form:

$$\mathbf{f}_I + \mathbf{f}_D + \mathbf{f}_S = \mathbf{p}(t) \qquad 2.1.1$$

Where $\mathbf{f}_I$ is a vector of the inertia forces for each DOF, $\mathbf{f}_D$ is a vector of the damping forces and $\mathbf{f}_S$ are the elastic forces.

The elastic force is defined:

$$\mathbf{f}_S = \mathrm{K}\mathbf{v}(t) \qquad 2.1.2$$

where $\mathbf{v}$ is the displacement vector and K is the matrix of stiffness coefficients. Note that K is constant for linear systems, but non-linear and time-varying in other more complex cases.

By assuming that the damping depends on the velocity (viscous damping) the damping force is defined:

$$\mathbf{f}_D = \mathrm{C}\dot{\mathbf{v}}(t) \qquad 2.1.3$$

where C is the damping matrix and $\dot{\mathbf{v}}$ is the velocity vector.

Finally, the inertial forces are defined:

$$\mathbf{f}_I = \mathrm{M}\ddot{\mathbf{v}}(t) \qquad 2.1.4$$

Where M is the mass matrix and $\ddot{\mathbf{v}}$ is the acceleration vector.

Substituting Equations (2.1.2) – (2.1.4) into the equation of motion in Equation (2.1.1) yields:

$$\mathrm{M}\ddot{\mathbf{v}}(t) + \mathrm{C}\dot{\mathbf{v}}(t) + \mathrm{K}\mathbf{v} = \mathbf{p}(t) \qquad 2.1.5$$

This equation requires the secant stiffness matrix, for non-linear analysis at any given time step. It can be hard to calculate accurately when the tangent stiffness matrix is changing each time step according to the hysteresis rule in use. Figure 2.1 shows the secant stiffness (K_secant) for a typical non-linear force displacement response. To avoid calculating the secant stiffness matrix every time step, which can be computationally expensive, an incremental equation of motion is used instead.

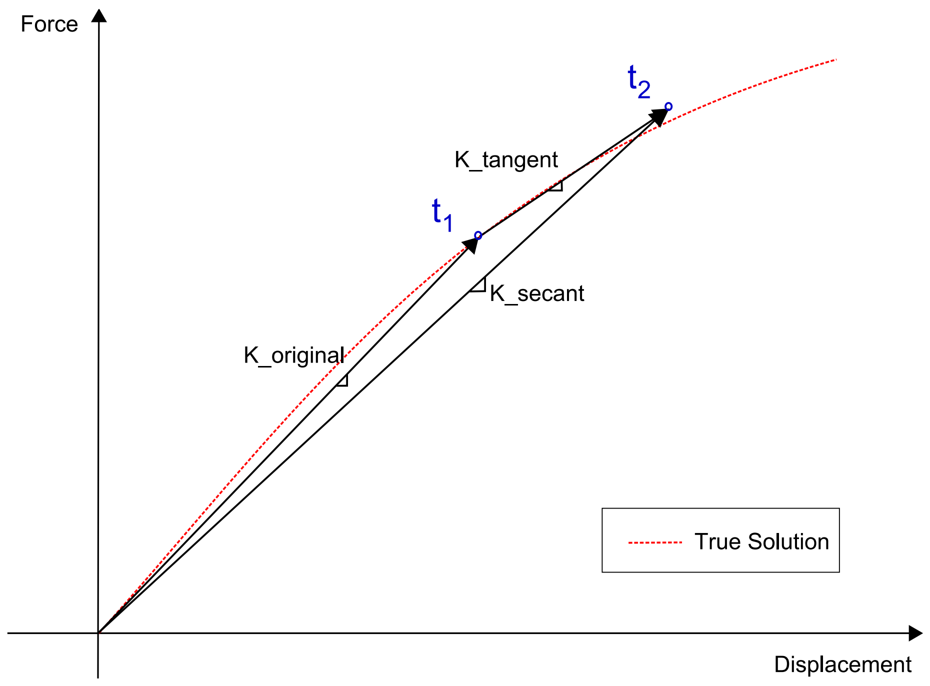


FIGURE 2.1: DEPICTION OF STIFFNESS MATRIX AT THE START OF THE TIMESTEP (K_ORIGINAL), THE TANGENT STIFFNESS AT THIS TIMESTEP (K_TANGENT) AND THE STIFFNESS AT THIS TIMESTEP (K_SECANT).

2.2 Elements

The elements used in this thesis are 2D beam elements with two nodes and three degrees of freedom per node as shown in Figure 2.2.

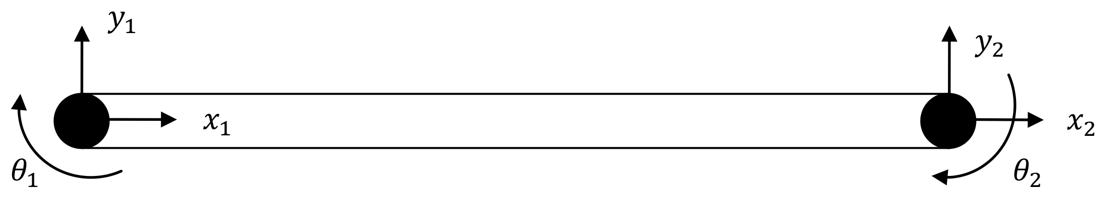


FIGURE 2.2: 2D BEAM ELEMENT

For the purposes of this thesis a lumped mass model was used. This is where the mass matrix is diagonal. The matrix is defined:

$$M = \begin{bmatrix} \frac{DL}{2} & & & & & \\ & \frac{DL}{2} & & & & \\ & & \frac{4}{420}DL^3 & & & \\ & & & \frac{DL}{2} & & \\ & & & & \frac{DL}{2} & \\ & & & & & \frac{4}{420}DL^3 \end{bmatrix} \qquad 2.2.1$$

Where L is the element length and D is the weight of the element per unit length.

The stiffness matrix is defined:

$$\mathrm{K} = \begin{bmatrix} \frac{AE}{L} & & & -\frac{AE}{L} & & \\ & \frac{12EI}{L^3} & \frac{6EI}{L^2} & & -\frac{12EI}{L^3} & \frac{6EI}{L^2} \\ & \frac{6EI}{L^2} & \frac{4EI}{L} & & -\frac{6EI}{L^2} & \frac{2EI}{L} \\ -\frac{AE}{L} & & & \frac{AE}{L} & & \\ & -\frac{12EI}{L^3} & -\frac{6EI}{L^2} & & \frac{12EI}{L^3} & \frac{6EI}{L^2} \\ & \frac{6EI}{L^2} & \frac{2EI}{L} & & \frac{6EI}{L^2} & \frac{4EI}{L} \end{bmatrix} \quad 2.2.2$$

A Caughey damping model was used for the damping matrix in this thesis. This allowed setting the damping for each mode in the response. The calculation for the Caughey damping matrix is outlined here.

First the eigen-problem is solved as defined:

$$\mathrm{KV} = \mathrm{MV\Lambda} \quad 2.2.3$$

Where V is a matrix of eigenvectors and Λ is a diagonal matrix of eigenvalues. The modal damping matrix, C^*, is then defined:

$$\mathrm{C}^* = 2\zeta\Omega \quad 2.2.4$$

Where ζ is a vector of damping ratios for each mode and Ω is the square root of each value in Λ. This modal damping matrix can now be converted to the damping matrix, C:

$$\mathrm{C} = \mathrm{MVC^*V'M} \quad 2.2.5$$

A major downside of this damping model is that the C matrix is almost full so the system matrices cannot be stored as sparse matrices. This will disadvantage both algorithms in both speed and memory usage.

2.3 Newmark-β Derivation

Start by considering the acceleration history at two points, n and $n+1$, as in Figure 2.3.

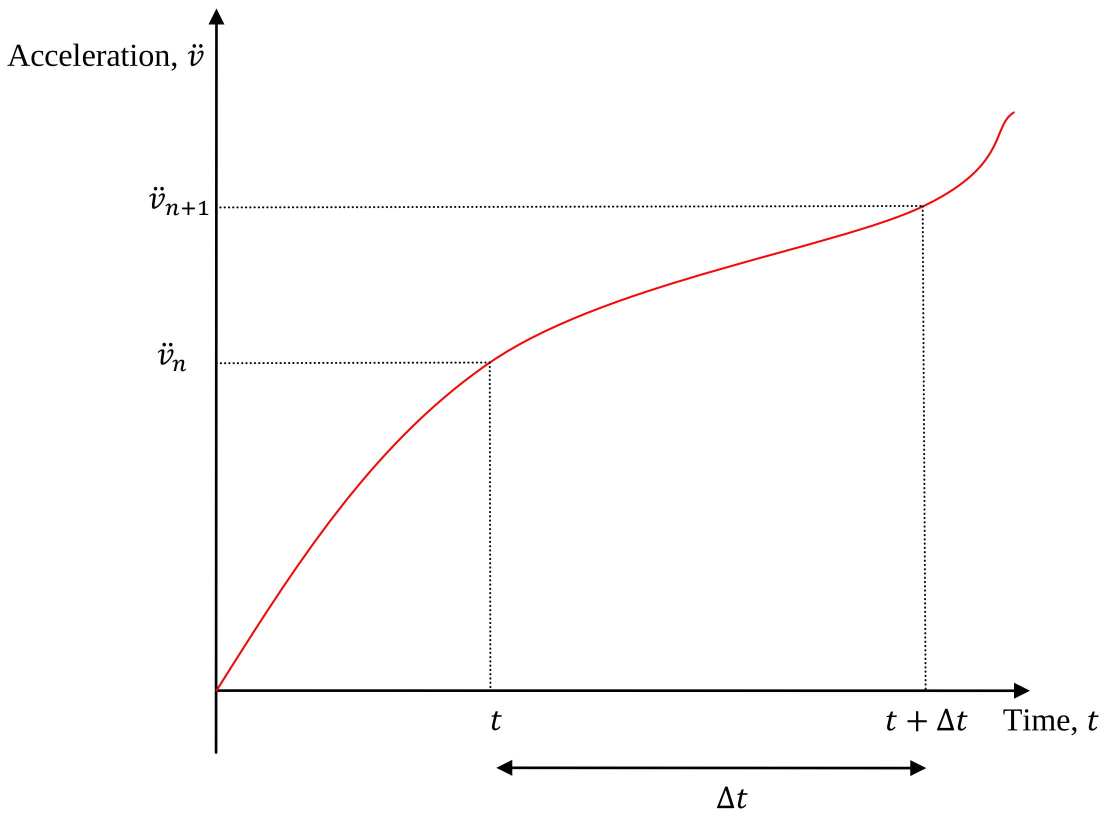


FIGURE 2.3: ACCELERATION HISTORY

Assume acceleration between n and $n+1$ to be:

$$\ddot{\mathbf{v}}_\gamma = (1-\gamma)\ddot{\mathbf{v}}_n + \gamma\ddot{\mathbf{v}}_{n+1} \tag{2.3.1}$$

Integrating to find velocity:

$$\begin{aligned}\dot{\mathbf{v}}_{n+1} &= \int_{t=0}^{t=n+1} \ddot{\boldsymbol{v}}(t)\,dt = \int_{0}^{n\Delta t} \ddot{\mathbf{v}}(t)\,dt + \int_{n\Delta t}^{(n+1)\Delta t} \ddot{\mathbf{v}}(t)\,dt \\ &= \dot{\mathbf{v}}(t=n) + \int_{n\Delta t}^{(n+1)\Delta t} (1-\gamma)\ddot{\mathbf{v}}_n + \gamma\ddot{\mathbf{v}}_{n+1}\,dt \\ &= \dot{\mathbf{v}}_n + (1-\gamma)\ddot{\mathbf{v}}_n\Delta t + \gamma\ddot{\mathbf{v}}_{n+1}\Delta t\end{aligned} \qquad 2.3.2$$

Now assume that the acceleration between n and $n+1$ is defined:

$$\ddot{\mathbf{v}}_\beta = (1-2\beta)\ddot{\mathbf{v}}_n + 2\beta\ddot{\mathbf{v}}_{n+1} \qquad 2.3.3$$

Integrate once to find velocity in terms of $\ddot{v}_\beta$:

$$\begin{aligned}\dot{v}_{n+1} &= \int_{t=0}^{t=n+1} \ddot{\mathbf{v}}(t)\,dt = \int_{0}^{n\Delta t} \ddot{\mathbf{v}}(t)\,dt + \int_{n\Delta t}^{(n+1)\Delta t} \ddot{\mathbf{v}}(t)\,dt \\ &= \dot{\mathbf{v}}_n + \ddot{\mathbf{v}}_\beta\Delta t\end{aligned} \qquad 2.3.4$$

Integrate (2.3.4) and substitute (2.3.3) to find the displacement:

$$\begin{aligned}\mathbf{v}_{n+1} &= \int_{0}^{\mathrm{n}\Delta t} \dot{\mathbf{v}}_n\,dt + \int_{\mathrm{n}\Delta t}^{(n+1)\Delta t} \dot{\mathbf{v}}_n\,dt + \int_{\mathrm{n}\Delta t}^{(n+1)\Delta t} \ddot{\mathbf{v}}_\beta\Delta t\,dt \\ &= \mathbf{v}_n + \dot{\mathbf{v}}_n\Delta t + \frac{\ddot{\mathbf{v}}_\beta(\Delta t)^2}{2} \\ &= \mathbf{v}_n + \dot{\mathbf{v}}_n\Delta t + (1-2\beta)\ddot{\mathbf{v}}_n\frac{(\Delta t)^2}{2} + \beta\ddot{\mathbf{v}}_{n+1}(\Delta t)^2\end{aligned} \qquad 2.3.5$$

In this thesis only constant average acceleration is considered so by setting $\gamma = \frac{1}{2}$ and $\beta = \frac{1}{4}$ in Equations (2.3.2) and (2.3.5) the following equations for the velocity and displacement at the end of the time step are found:

$$\dot{\mathbf{v}}_{n+1} = \dot{\mathbf{v}}_n + \frac{\Delta t}{2}(\ddot{\mathbf{v}}_n + \ddot{\mathbf{v}}_{n+1}) \tag{2.3.6}$$

$$\mathbf{v}_{n+1} = \mathbf{v}_n + \Delta t\dot{\mathbf{v}}_n + \frac{(\Delta t)^2}{4}(\ddot{\mathbf{v}}_n + \ddot{\mathbf{v}}_{n+1}) \tag{2.3.7}$$

Now rearrange (2.3.7) for $\ddot{\mathbf{v}}_{n+1}$:

$$\frac{(\Delta t)^2}{4}\ddot{\mathbf{v}}_{n+1} = \mathbf{v}_{n+1} - \mathbf{v}_n - \Delta t\dot{\mathbf{v}}_n - \frac{(\Delta t)^2}{4}\ddot{\mathbf{v}}_n \tag{2.3.8}$$

$$\ddot{\mathbf{v}}_{n+1} = \frac{4}{(\Delta t)^2}(\mathbf{v}_{n+1} - \mathbf{v}_n - \Delta t\dot{\mathbf{v}}_n) - \ddot{\mathbf{v}}_n \tag{2.3.9}$$

Now substitute (2.3.9) into (2.3.6) to find $\dot{\mathbf{v}}_{n+1}$:

$$\dot{\mathbf{v}}_{n+1} = \dot{\mathbf{v}}_n + \frac{\Delta t}{2}\left(\ddot{\mathbf{v}}_n + \frac{4}{(\Delta t)^2}(\mathbf{v}_{n+1} - \mathbf{v}_n - \Delta t\dot{\mathbf{v}}_n) - \ddot{\mathbf{v}}_n\right) \tag{2.3.10}$$

$$\dot{\mathbf{v}}_{n+1} = \dot{\mathbf{v}}_n + \frac{2}{\Delta t}(\mathbf{v}_{n+1} - \mathbf{v}_n - \Delta t\dot{\mathbf{v}}_n) \tag{2.3.11}$$

Equations (2.3.9) and (2.3.11) can now be substituted into the equation of motion, Equation (2.1.5):

$$\begin{aligned} &\mathrm{M}\left(\frac{4}{(\Delta t)^2}(\mathbf{v}_{n+1} - \mathbf{v}_n - \Delta t\dot{\mathbf{v}}_n) - \ddot{\mathbf{v}}_n\right) \\ &+ \mathrm{C}\left(\dot{\mathbf{v}}_n + \frac{2}{\Delta t}(\mathbf{v}_{n+1} - \mathbf{v}_n - \Delta t\dot{\mathbf{v}}_n)\right) + \mathrm{K}\mathbf{v}_{n+1} = \mathbf{p}(t) \end{aligned} \tag{2.3.12}$$

$$\left(\frac{4\mathrm{M}}{(\Delta t)^2}+\frac{2\mathrm{C}}{\Delta t}+\mathrm{K}\right)\mathbf{v}_{n+1}=\mathbf{p}(t)+\mathrm{M}\left(\frac{4}{(\Delta t)^2}(\mathbf{v}_n+\Delta t\dot{\mathbf{v}}_n)+\ddot{\mathbf{v}}_n\right) + C\left(\frac{2}{\Delta t}\mathbf{v}_n+\dot{\mathbf{v}}_n\right) \qquad 2.3.13$$

Equation (2.3.13) can be expressed in terms of an effective stiffness matrix and loading vector:

$$\widetilde{\mathrm{K}}\mathbf{v}_{n+1}=\widetilde{\mathbf{p}}(t) \qquad 2.3.14$$

Where $\widetilde{\mathrm{K}}$ and $\widetilde{\mathbf{p}}$ are defined:

$$\widetilde{\mathrm{K}}=\frac{4\mathrm{M}}{(\Delta t)^2}+\frac{2\mathrm{C}}{\Delta t}+\mathrm{K} \qquad 2.3.15$$

$$\widetilde{\mathbf{p}}(t)=\mathbf{p}(t)+\mathrm{M}\left(\frac{4}{(\Delta t)^2}(\mathbf{v}_n+\Delta t\dot{\mathbf{v}}_n)+\ddot{\mathbf{v}}_n\right)+C\left(\frac{2}{\Delta t}\mathbf{v}_n+\dot{\mathbf{v}}_n\right) \qquad 2.3.16$$

The algorithm for this constant average Newmark-β scheme is as follows

1. Define initial conditions $\mathbf{v}_0$, $\dot{\mathbf{v}}_0$ and $\ddot{\mathbf{v}}_0$
2. Solve (2.3.14) for $\mathbf{v}_{n+1}$ then use (2.3.9) and (2.3.11) to find $\ddot{\mathbf{v}}_{n+1}$ and $\dot{\mathbf{v}}_{n+1}$ respectively.
3. Repeat step 2 until at the end of the acceleration vector $\mathbf{p}(t)$

2.4 Incremental Equilibrium

In Clough & Penzien (1975) the incremental equilibrium equation is defined based on the original Newmark-β equations of motion:

$$\widetilde{\mathrm{K}}_{\mathrm{t}}\Delta\mathbf{v}=\Delta\widetilde{\mathbf{p}}[1] \qquad 2.4.1$$

[1] Eqn 15-5 (Clough & Penzien, 1975)

Where $\widetilde{\mathrm{K}}_t$ is the effective tangent stiffness matrix and $\Delta\widetilde{\mathbf{p}}$ is the effective incremental loading. The effective tangent stiffness matrix is defined:

$$\widetilde{\mathrm{K}}_{\mathbf{t}} = \mathrm{K}_{\mathrm{t}} + \frac{2}{\Delta t}\mathrm{C} + \frac{4}{\Delta t^2}\mathrm{M} \qquad 2.4.2$$

And the effective incremental loading is defined:

$$\Delta\widetilde{\mathbf{p}} = \Delta\mathbf{p} + 2\mathrm{C}\dot{\mathbf{v}}_0 + \mathrm{M}\left[\frac{4}{h}\dot{\mathbf{v}}_0 + 2\ddot{\mathbf{v}}_0\right] \qquad 2.4.3$$

Using the displacement increment calculated at a given time step from Equation (2.4.1), the incremental velocity can be found using:

$$\Delta\dot{\mathbf{v}} = \frac{2}{\Delta t}\Delta\mathbf{v} - 2\dot{\mathbf{v}}_0 \qquad 2.4.4$$

The initial acceleration vector should be calculated at the start of the step by solving for $\ddot{\mathbf{v}}_0$ in Equation (2.1.1), yielding:

$$\mathrm{M}\ddot{\mathbf{v}}_0 = \mathbf{p}_0 - \mathbf{f}_{D_0} - \mathbf{f}_{S_0} \qquad 2.4.5$$

Thus, the overall process using incremental equation is defined:

1. Define initial conditions $\mathbf{v}_0$ and $\dot{\mathbf{v}}_0$
2. Find the change in the forcing vector $\mathbf{p}(t)$, $\Delta\mathbf{p}$, since the previous time step.
3. Use (2.4.5) to find $\ddot{\mathbf{v}}_0$, the acceleration at the start of the time step
4. Use $\Delta\mathbf{p}$ to solve (2.4.1) for the incremental displacement $\Delta\mathbf{v}$
5. Use (2.4.4) to find the incremental velocity $\Delta\dot{\mathbf{v}}$
6. Finally add the incremental displacements and velocities to a running sum (the actual displacement and velocity).
7. Store the actual values for this time step
8. Repeat from step 2 until at the end of the acceleration vector $\mathbf{p}(t)$

2.5 Derivation of First Order Equations for ODE Solution

Start with the second order system of equations defined:

$$\mathrm{M}\ddot{\mathbf{v}} + \mathrm{C}\dot{\mathbf{v}} + \mathrm{K}\mathbf{v} = -\mathrm{M}\ddot{\mathbf{x}}_g \qquad 2.5.1$$

Introduce a second variable by setting $\mathbf{v}_1 = \mathbf{v}$ and $\mathbf{v}_2 = \dot{\mathbf{v}}$. Now, solving for the derivatives of these new variables yields:

$$\begin{aligned} \dot{\mathbf{v}}_1 &= \mathbf{v}_2 \\ \dot{\mathbf{v}}_2 &= \mathrm{M}^{-1}(\mathbf{p} - \mathrm{C}\mathbf{v}_2 - \mathrm{K}\mathbf{v}_1) \end{aligned} \qquad 2.5.2$$

Or in matrix form with $\check{\mathbf{v}} = [\mathbf{v}_1 \quad \mathbf{v}_2]^{\mathrm{T}}$:

$$\dot{\check{\mathbf{v}}} = \mathrm{A}\,\check{\mathbf{v}} + \begin{bmatrix} \mathbf{0} \\ \mathrm{M}^{-1}\mathbf{p} \end{bmatrix} \qquad 2.5.3$$

Where A is defined:

$$\mathrm{A} = \begin{bmatrix} 0 & \mathrm{I} \\ -\mathrm{M}^{-1}\mathrm{K} & -\mathrm{M}^{-1}\mathrm{K} \end{bmatrix} \qquad 2.5.4$$

Where 0 is a matrix of (n x n) made of all 0 values and I is the identity matrix of size (nxn).

Equations (2.5.3) and (2.5.4) are the standard state space formulation for the 2nd order mechanical vibration problems.

The hysteresis rule defines the relationship between stress and strain. This relationship is captured in the tangent stiffness matrix. To make use of this fact we can follow the elastic force defined:

$$\mathbf{F} = \mathbf{F}_{old} + \mathrm{K}_{tangent}(\mathbf{v} - \mathbf{v}_{old}) \qquad 2.5.5$$

Recall that in the case of Ramberg-Osgood hysteresis, the tangent stiffness matrix for the initial loading is defined:

$$\mathrm{K}_{tangent} = \frac{\mathrm{K}_0}{1 + \left|\frac{\mathbf{F}}{\mathbf{F}_y}\right|^{r-1}} \tag{2.5.6}$$

As the time step goes to zero the differential equation (DE) becomes:

$$\begin{gathered} M\ddot{\mathbf{v}} + C\dot{\mathbf{v}} + \mathbf{F} = -M\ddot{\mathbf{x}}_g \\ \frac{d\mathbf{F}}{d\mathbf{v}} = K_{tangent} = \frac{\mathrm{K}_0}{1 + \left|\frac{\mathbf{F}}{\mathbf{F}_y}\right|^{r-1}} \end{gathered} \tag{2.5.7}$$

However, $d\mathbf{F}/d\mathbf{v}$ can be redefined using the chain rule, yielding:

$$\frac{d\mathbf{F}}{d\mathbf{v}} = \frac{d\mathbf{F}}{dt}\frac{dt}{d\mathbf{x}} = \frac{\dot{\mathbf{F}}}{\dot{\mathbf{v}}} \tag{2.5.8}$$

So the set of DEs thus becomes:

$$\begin{gathered} M\ddot{\mathbf{v}} + C\dot{\mathbf{v}} + \mathbf{F} = -M\ddot{\mathbf{x}}_g \\ \dot{\mathbf{F}} = \frac{\dot{\mathbf{v}}\mathrm{K}_0}{1 + \left|\frac{\mathbf{F}}{\mathbf{F}_y}\right|^{r-1}} \end{gathered} \tag{2.5.9}$$

Or in first order form:

$$\begin{aligned} \dot{\mathbf{v}}_1 &= \mathbf{v}_2 \\ \dot{\mathbf{v}}_2 &= \mathrm{M}^{-1}(\mathbf{p} - \mathrm{C}\mathbf{v}_2 - \mathrm{K}\mathbf{v}_1) \\ \dot{\mathbf{F}} &= \frac{\mathbf{v}_2\mathrm{K}_0}{1 + \left|\frac{\mathbf{F}}{\mathbf{F}_y}\right|^{r-1}} \end{aligned} \tag{2.5.10}$$

Equation (2.5.10) is for the initial loading only. However, this formulation can be easily modified for subsequent loadings by altering the third line accordingly, using Equation (1.6.4) to give:

$$\begin{aligned}
\dot{\mathbf{v}}_1 &= \mathbf{v}_2 \\
\dot{\mathbf{v}}_2 &= \mathrm{M}^{-1}(\mathbf{p} - \mathrm{C}\mathbf{v}_2 - \mathrm{K}\mathbf{v}_1) \\
\dot{\mathbf{F}} &= \frac{\mathbf{v}_2 \mathrm{K}_0}{1 + \left|\frac{\mathbf{F} - \mathbf{F_i}}{2\mathbf{F}_y}\right|^{r-1}}
\end{aligned} \qquad 2.5.11$$

The initial loading is only until the element changes direction. The element changes direction when the velocity (the first derivative of displacement) crosses zero.

Now Equation (2.5.10) can be used for the first loading and Equation (2.5.11) for subsequent loadings. The algorithm for the general ODE solve with one element is:

1. Set initial conditions
2. Run one of MATLAB's ode numerical solutions to solve Equation (2.5.10) with events[2] turned on stopping at velocity ($\mathbf{v}_2$) crossing zero.
3. Store the data vectors.
4. Run the ode solver again this time with Equation (2.5.11) and with the initial conditions equal to the state of the system at the end of the previous solve. Again events must be turned on and the termination condition is the velocity crossing zero.
5. Add the data from this solve to the data vectors.
6. Check if we have reached the end of the input vector. If not repeat from step 4.

This algorithm must be slightly altered for a system with more than one element. With more than one element the displacement inside the element must be used in Equations (2.5.10) and

[2] See http://www.mathworks.com/access/helpdesk/help/techdoc/ref/ode23tb.html for an explanation of ODE events

(2.5.11) and similarly with the velocity for the termination condition. To do this the relative displacement of each of the element's nodes is used in place of the absolute values.

Additionally the termination condition must only be enabled after the second piece of input data. It is impossible for the system to turn in this space as there is only one piece of data so far. If the system is allowed to turn before the 2nd piece of data ODE numerical solutions that use more than one solve per step (for example MATLAB's ode113) can think the system has turned if the first piece of acceleration data is zero as the velocity is also zero.

2.6 Continuous Formulation Check

In the single degree of freedom case, a continuous formulation can be derived to check Equations 2.5.10. Starting with the single degree of freedom equation of motion with scalars:

$$M\ddot{x} + C\dot{x} + Kx = -M\ddot{x_g} \tag{2.6.1}$$

Where:

$$K = \frac{K_0}{1 + \left|\frac{F}{F_y}\right|^{r-1}} \tag{2.6.2}$$

And:

$$F = Kx \tag{2.6.3}$$

If we assume that the displacement is positive ($x > 0$) and thus $F = Kx > 0$, and put $r = 2$, then an analytical solution to K can be found, and is defined:

$$K = \frac{K_0}{1 + \frac{F}{F_y}} = \frac{K_0}{1 + \frac{Kx}{F_y}} \tag{2.6.4}$$

$$0 = xK^2 + F_yK - F_yK_0 \tag{2.6.5}$$

Equation (2.6.5) is quadratic in K. Solving for K, where $K > 0$ must be positive for a realistic result, yields:

$$K = \frac{-F_y + \sqrt{F_y^2 + 4xF_yK_0}}{2x} \tag{2.6.6}$$

Recalling the linear ODE equation of motion:

$$\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= -\frac{1}{M}(Cx_2 + Kx_1) - \ddot{x}_g \end{aligned} \tag{2.6.7}$$

Equation (2.6.6) can be substituted into Equation (2.6.7) yielding:

$$\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= -\frac{1}{M}\left(Cx_2 + \frac{1}{2}\left(-F_y + \sqrt{F_y^2 + 4x_1F_yK_0}\right)\right) - \ddot{x}_g \end{aligned} \tag{2.6.8}$$

This is a continuous first order ODE that can be used to check the more general form in Equation (2.5.10). The algorithm used for verification of the initial loading of the other numerical solutions is:

1. Set initial conditions
2. Run one of MATLAB’s ode numerical solutions to solve Equation (2.6.8)
3. Compare results with the other numerical solutions

2.7 Chapter Summary

This chapter first presented the equation of motion (EOM), the governing equation for the analysis in this thesis. The numerical solutions presented in this thesis solve this governing equation to find the motion of the structure to be analysed.

Secondly the constant average acceleration Newmark-β numerical integration scheme is derived and the algorithm for its use is presented. This method is the typical approach for linear dynamic analysis. An enhancement to this Newmark-β is then presented using the incremental form of the EOM. This enhancement allows solving with non-linear hysteresis models without needing to calculate the secant stiffness matrix.

The continuous form of the EOM is then derived for use in the ODE solver algorithm. The algorithm is also outlined. This algorithm is later shown to be a more efficient alternative to the Newmark-β algorithm.

Finally the continuous form is solved analytically for the single degree of freedom case. This analytical solution can then be used to verify the other two numerical solutions.

3 METHODS

3.1 CODE FLOWCHARTS

The general outline of the process of solving the earthquake response is given in Figure 3.1. The solution method section can be either the Newmark-β (Figure 3.2) or the ODE solver (Figure 3.3). The actual ODE function is outlined in Figure 3.4.

The first section of the code is problem definition. This consists of importing an acceleration vector and the corresponding *dt*, the time between each acceleration data point. The building metrics are then defined. They include: number of floors, number of bays, elements per column, height of each floor and bay width. The material and cross-section properties are also defined.

With all this information stored a script is used to generate the mesh and boundary condition matrix. The mesh generation is done by creating a node list and a vector of element objects connecting the nodes. This information is returned in the form used by the solution methods.

This collection of system defining objects and matrices is then given to either the Newmark-β or ODE solution function. These functions are very similar with key differences being that the Newmark-β code needs a user defined solver time step to use and the ODE function can accept an alternative ODE solver to use in the numerical integration. The details of these algorithms are given later in this section. Upon completion of the algorithm the displacement data is returned with a corresponding time vector.

With these results the motion of the building can be visualised in several ways. The motion of particular nodes be tracked or a series of images of the building during the analysis can be produced from which a video can be created.

The speed of all the steps here are largely unimportant as none of them are repeated even when using multiple input acceleration vectors. It is only the time taken by the solution function that is of interest.

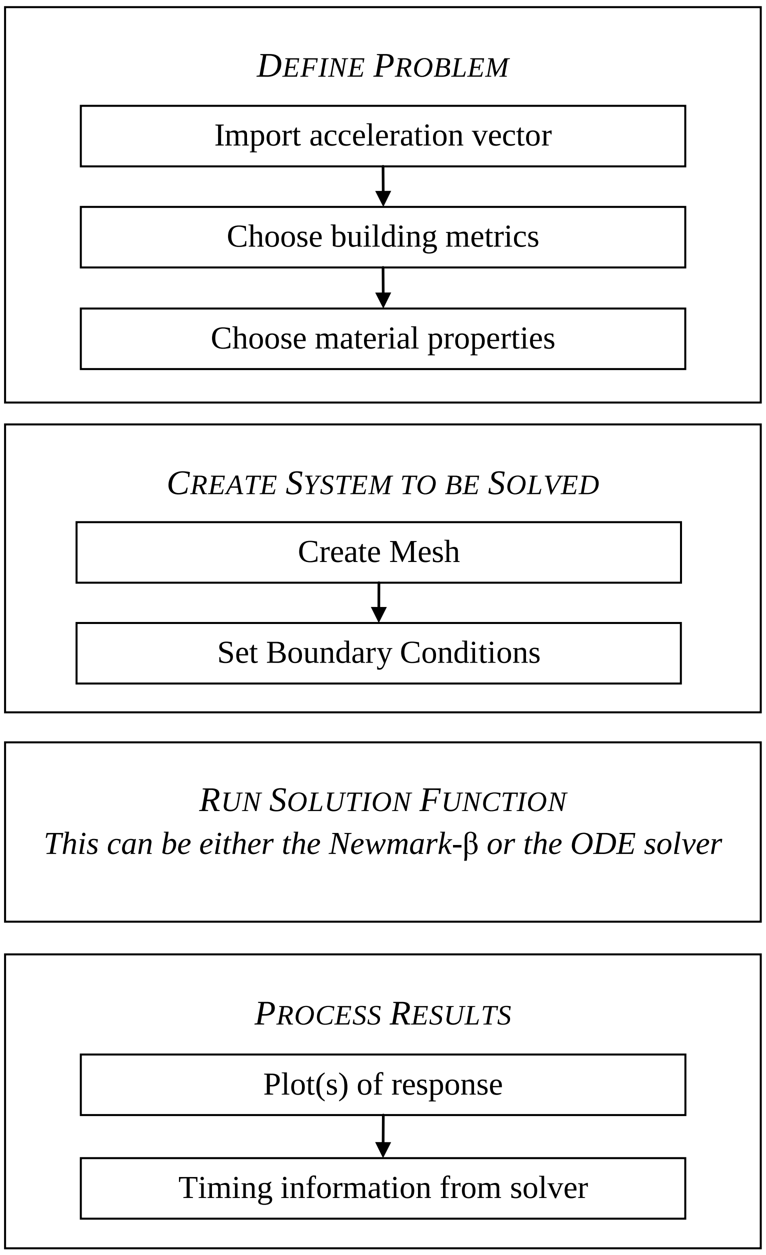

DEFINE PROBLEM
Import acceleration vector
Choose building metrics
Choose material properties
CREATE SYSTEM TO BE SOLVED
Create Mesh
Set Boundary Conditions
RUN SOLUTION FUNCTION
This can be either the Newmark-β or the ODE solver
PROCESS RESULTS
Plot(s) of response
Timing information from solver

FIGURE 3.1: CODE OVERVIEW FLOWCHART

The Newmark-β function begins by creating a time vector that will correspond to the output data using the user selected solver time step. The acceleration vector is also interpolated to give the acceleration at each solver time step. There is a provision for storing less steps than are actually calculated in the case of very small solver time steps. This works to significantly reduce memory requirements while making negligible change in result. The solver still uses the small time step to numerically integrate; it is only the storing step that is altered.

The global mass and stiffness matrices are calculated using the system defining data that was found earlier. From these the damping matrix can be found using the damping model selected. It is trivial to create an alternative damping model function in place of the Caughey function.

The mass matrix is then inverted and stored. This is a computationally expensive process so it would be undesirable to be doing this at every time step. This makes the assumption that the mass matrix does not change with time. Typically inverting matrices is avoided in computational mathematics but here we know the matrix is diagonally dominant (see Equation (2.2.1)). This guarantees that it is invertible (Farid, 1995).

Up to this point everything that has been done is only done once and need not be repeated for running the same model with different input accelerations. This means the speed is largely inconsequential. The actual time stepping loop starts here. First a forcing vector is created using the boundary conditions and the acceleration vector; this is the application of the type 2 boundary conditions. By using this vector and the previous step’s forcing vector (all zeros in the case of the first step) an incremental forcing vector is found.

This forcing vector and the previous step’s displacement vector (all zeros in the case of the first step) are passed to each element object. These element objects use this information together with the hysteresis rule defined for each element to produce a local tangent stiffness matrix at this time step.

The local tangent stiffness matrices from each element can then be assembled into a global tangent stiffness matrix. These two steps involving applying the hysteresis rule to each element and then assembling the global stiffness matrix is very computationally expensive and is also done in the ODE function. As each of the elements is potentially changing direction and thus stiffness at different times they must be found individually. It is important to note that any parallelisation or other method to speed this process up will substantially benefit both solvers.

If a different hysteresis rule were required this need only be made into a function and passed to each element in place of the Ramberg-Osgood function. This would seamlessly integrate and no further changes need be made.

The effective stiffness and effective loading can now be calculated using Equations (2.4.3) and (2.4.2). The initial acceleration is also found using Equation (2.4.5). After the type 1 boundary conditions have been applied Equation (2.4.1) can be solved to find the incremental displacement.

The incremental displacement is used to find the incremental velocity using Equation (2.4.4). The actual displacement, velocity and acceleration are found by adding the incremental values. These are added to the storage vector to be returned and the next time step is started.

There is also an added feature to reduce memory requirements. In order for the Newmark-β algorithm to accurately find the turning points in the hysteresis loops very small time steps must be used. Although this accuracy is required to find the subsequent displacements it is not required for post processing. This result allows the algorithm to use the very small time step but not store all of the data that was used.

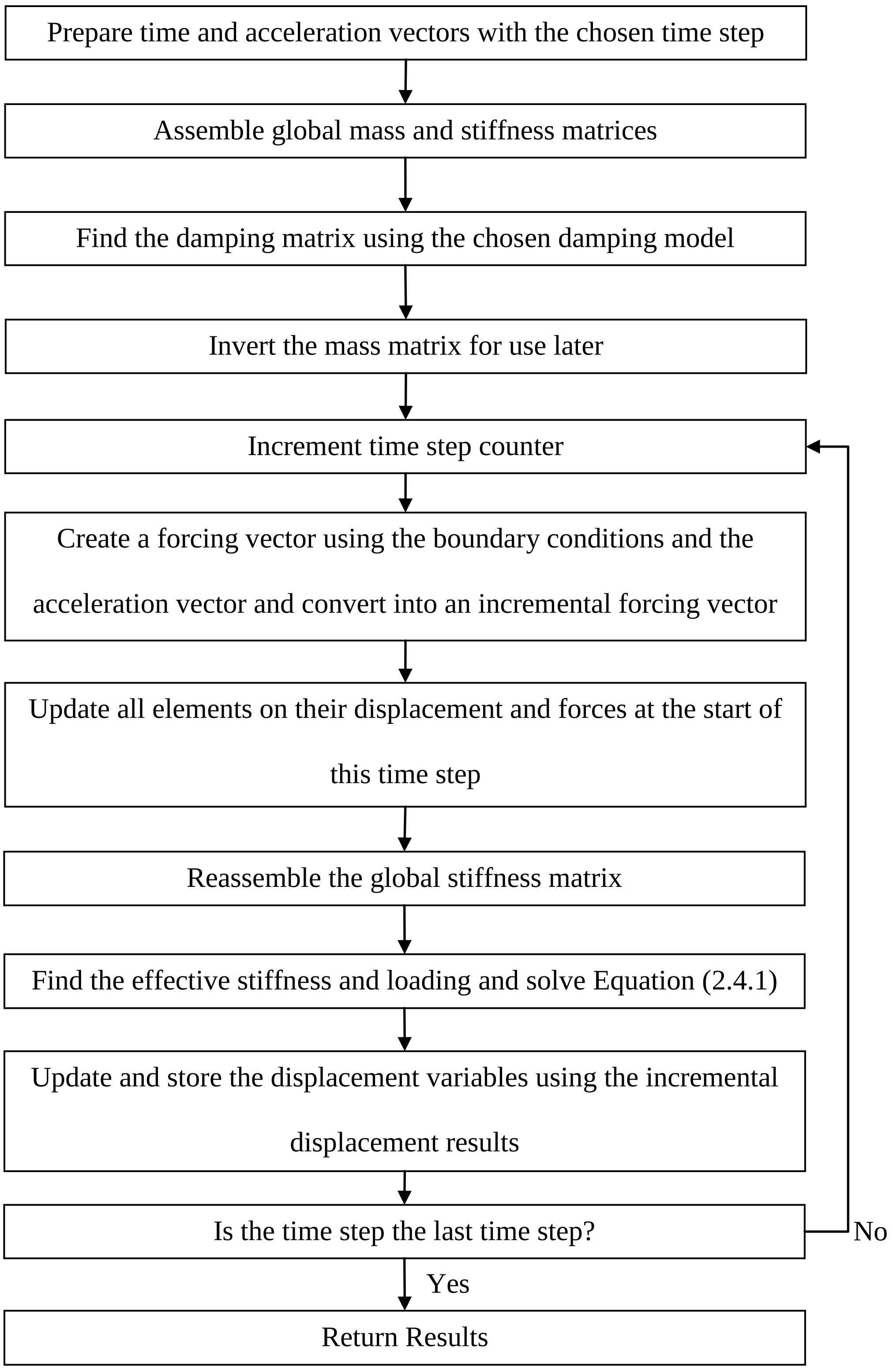

Prepare time and acceleration vectors with the chosen time step
Assemble global mass and stiffness matrices
Find the damping matrix using the chosen damping model
Invert the mass matrix for use later
Increment time step counter
Create a forcing vector using the boundary conditions and the acceleration vector and convert into an incremental forcing vector
Update all elements on their displacement and forces at the start of this time step
Reassemble the global stiffness matrix
Find the effective stiffness and loading and solve Equation (2.4.1)
Update and store the displacement variables using the incremental displacement results
Is the time step the last time step?
No
Yes
Return Results

FIGURE 3.2: FLOWCHART OF NEWMARK-B CODE

The outer ODE function starts with finding the global mass and stiffness matrices as in the Newmark-β function. The solver time vector is not able to be calculated at this point however as the ODE code proceeds with a variable time step. As with the Newmark-β function, the damping matrix is also found using the chosen damping model function.

A forcing vector is created using the boundary conditions; this is applying the type 2 boundary conditions. The type 1 boundary conditions are applied by removing the fixed degrees of freedom from the system matrices. As with the Newmark-β function the mass matrix is inverted and stored for later use.

The next step is to create a list of degrees of freedom relating to each element. This is used later in using turning points in the hysteresis calculation for each element.

Finally the ode options are set up. The MATLAB ODE solver to be used is chosen either by the default or the user selection. The solver is given the time step of the acceleration data as a maximum step size. This prevents the solver from missing out any of the input data. The solver is also given a relative tolerance of 1e-10 and events are turned on. Turning on events will cause the solver to stop when certain criteria defined in the inner ODE function are met. The very small relative tolerance has very little impact on performance and ensures that the solver is finding all the points to great accuracy.

The ODE solver is then run for the initial loading (up to the first element turn) with initial conditions being all zero. The inner ODE function solved by the MATLAB ODE solver is described later in this section. On completion of this initial run the displacement, velocity, force and time matrices/vector are returned. The end values of these are used as the initial conditions for the following run.

At the end of each run the results data is appended to the results so far and at the end the final results are returned. There is also provision for checking that too much data is being stored and an interpolation will be run occasionally to reduce the memory requirements. This is done in both solver methods.

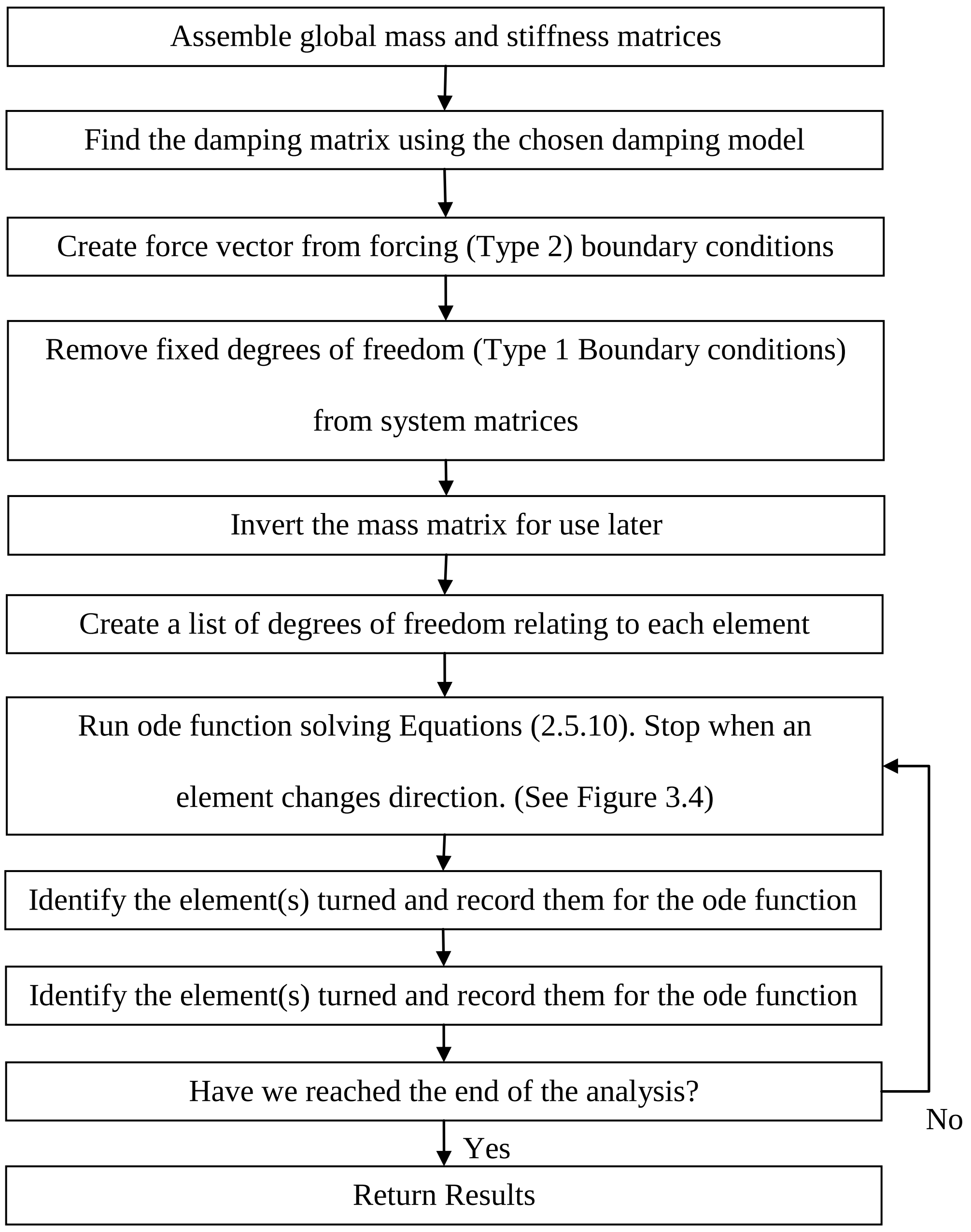

Assemble global mass and stiffness matrices
Find the damping matrix using the chosen damping model
Create force vector from forcing (Type 2) boundary conditions
Remove fixed degrees of freedom (Type 1 Boundary conditions) from system matrices
Invert the mass matrix for use later
Create a list of degrees of freedom relating to each element
Run ode function solving Equations (2.5.10). Stop when an element changes direction. (See Figure 3.4)
Identify the element(s) turned and record them for the ode function
Identify the element(s) turned and record them for the ode function
Have we reached the end of the analysis?
No
Yes
Return Results

FIGURE 3.3: FLOWCHART FOR THE OUTER ODE FUNCTION

The inner ODE function starts by taking the acceleration vector and corresponding time vector and interpolating to find the acceleration at this time step. The first 2 equations from Equations (2.5.10) can be solved to find the tangent of the displacement and velocity. This leaves only the tangent of the force which depends on the hysteresis model.

To apply the Ramberg-Osgood hysteresis model to each element first each element is selected in turn. The element has a yield force defined in the element object. The local velocity vector and force vector are found using the list of degrees of freedom relating to each element which was found in the outer ODE function. The local initial stiffness is known and stored in the element object.

By looking at the data passed to the inner ODE function it is determined if this element has turned and when. Based on this either Equation (2.5.10) or (2.5.11) is solved to find the tangent of the force at this time step.

If another hysteresis model were to be used then this procedure would need to be changed accordingly. This could be achieved by having each element set a hysteresis model as in the Newmark-β but instead of the local tangent stiffness matrix being returned the tangent force is returned.

The tangent of the force for each element is assembled into a global force and returned along with the global tangents of the displacement and velocity. Just like the incremental Newmark-β algorithm Figure 3.2 the secant stiffness is not calculated.

The inner ODE function also gets called by the MATLAB ODE solver to find when to stop. The stopping condition is any element turning. The element velocity crossing zero is used as this stopping condition. To find the velocity of each element the global velocity vector must be split into local velocity for each element. The termination condition is then defined as any of those velocities crossing zero. The MATLAB ODE solvers will find this stopping condition accurately as it is able to reduce the step size near interesting points.

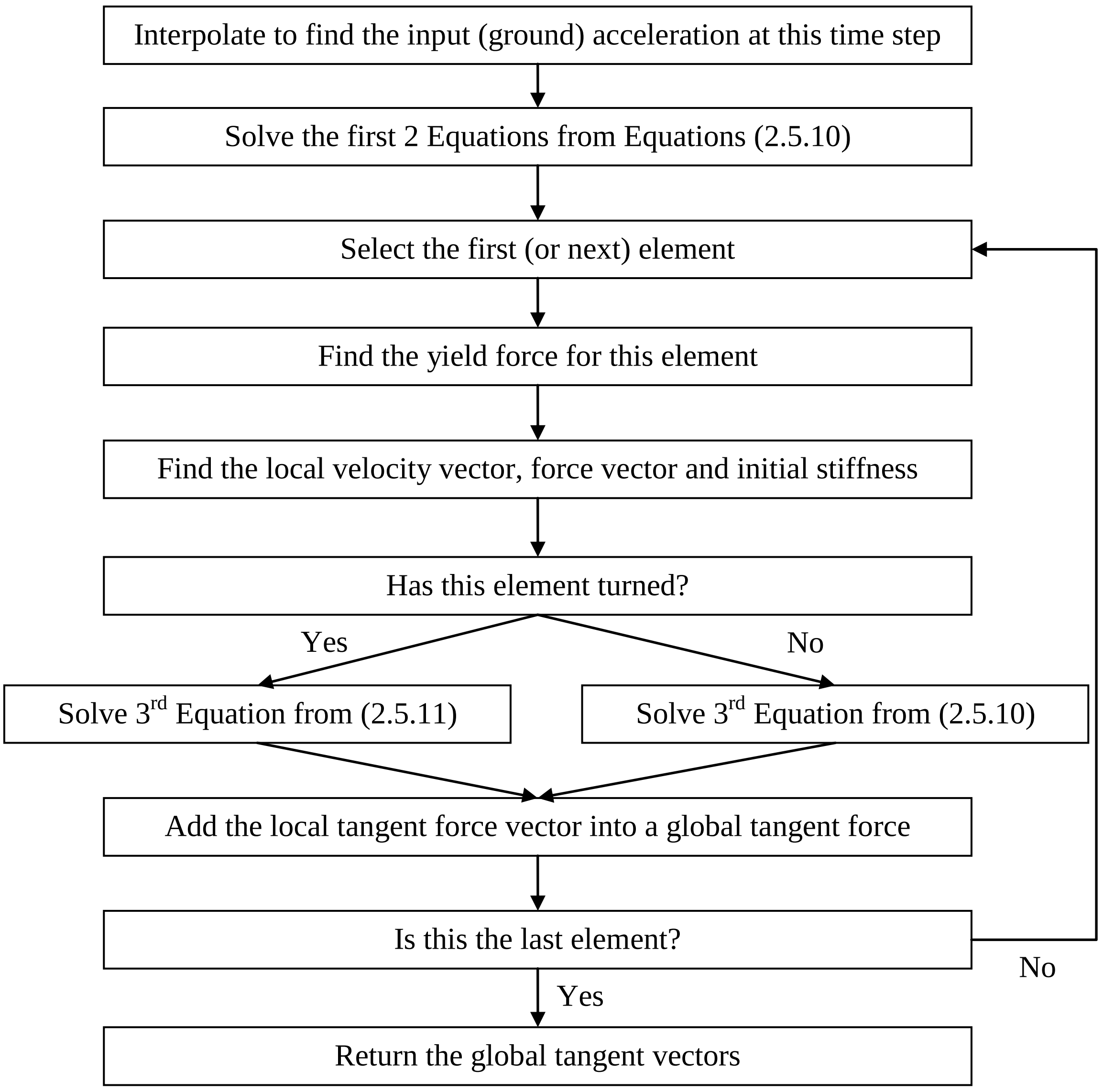

Interpolate to find the input (ground) acceleration at this time step
Solve the first 2 Equations from Equations (2.5.10)
Select the first (or next) element
Find the yield force for this element
Find the local velocity vector, force vector and initial stiffness
Has this element turned?
Yes
No
Solve 3rd Equation from (2.5.11)
Solve 3rd Equation from (2.5.10)
Add the local tangent force vector into a global tangent force
Is this the last element?
No
Yes
Return the global tangent vectors

FIGURE 3.4: INNER ODE FUNCTION FLOWCHART

The main difference between the algorithms is that the ODE code is solving the continuous form the equation of motion. Both algorithms are essentially the same except that the Newmark-β code is solving incrementally each time step while the ODE code is able to solve the continuous form at any time point. This flexibility allows more efficient numerical integration algorithms to be used.

Although they are essentially doing the same thing the algorithms are coded very differently. The Newmark-β function simply steps through each time step as in Figure 3.2 but the ODE function is split up into two distinct parts. The outer part as presented in Figure 3.3 is preparing the data needed for the MATLAB ODE solver to run. The inner part is called by MATLAB's ODE solver and is presented in Figure 3.4. It is this inner code that is called many times while the outer code sets up the data for the solver to run again after each turn of an element.

Both algorithms have the same problem in terms of computational intensity. They both need to find if each element has turned and what that element's local stiffness is, based on the hysteresis rule. The ODE code will only be faster if it is able to do this expensive operation less often than the Newmark-β code.

This most expensive part of the analysis involves dealing with each element individually. The result from each element does not depend on any other element. This is termed a highly parallelisable problem (Berkman, Galil, Schieber, & Vishkin, 1989). If this section of the code were to be run in parallel we could expect performance increases roughly proportional to the number of threads on the system.

The way the Newmark-β function is coded will tend to give it an advantage in the MATLAB code that will not exist when using a compiled language. This is because the other very computationally expensive part of the Newmark-β algorithm is the solving of Equation (2.4.1). This is done with a built in MATLAB command (Mathworks, 2010). This part of the code will not benefit from coding in a compiled language as it already is. However, it can be parallelised as the matrix involved is sparse (Amestoy, Duff, L'Excellent, & Li, 2001).

3.2 Input Acceleration

For the purposes of benchmarking and verifying using the models presented in the next section two input accelerations were used. The first is a synthetic acceleration record shown in Figure 3.5. The second is the North-South component of the El Centro earthquake recorded on May 18 1940. The acceleration record is shown in Figure 3.6.

The synthetic acceleration record begins with a ramp in one direction followed by an equal ramp in the opposite direction. The first ramp is sufficiently strong to cause plastic deformation while the second stops the building accelerating. The building is then allowed to respond during a period of no acceleration followed by a final ramp. This last ramp is large enough to again cause plastic deformation. This will tend to show up slight inaccuracies in following the hysteresis model as the building is plastically deformed early on and then again loaded at the end. The record is also very short at only 2.5 seconds which allows for quick testing.

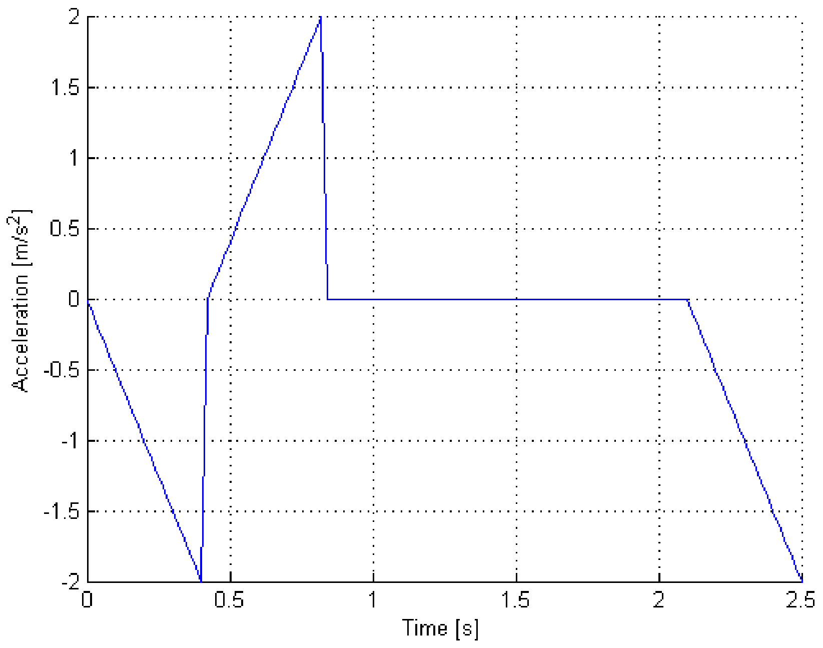


FIGURE 3.5: SYNTHETIC INPUT

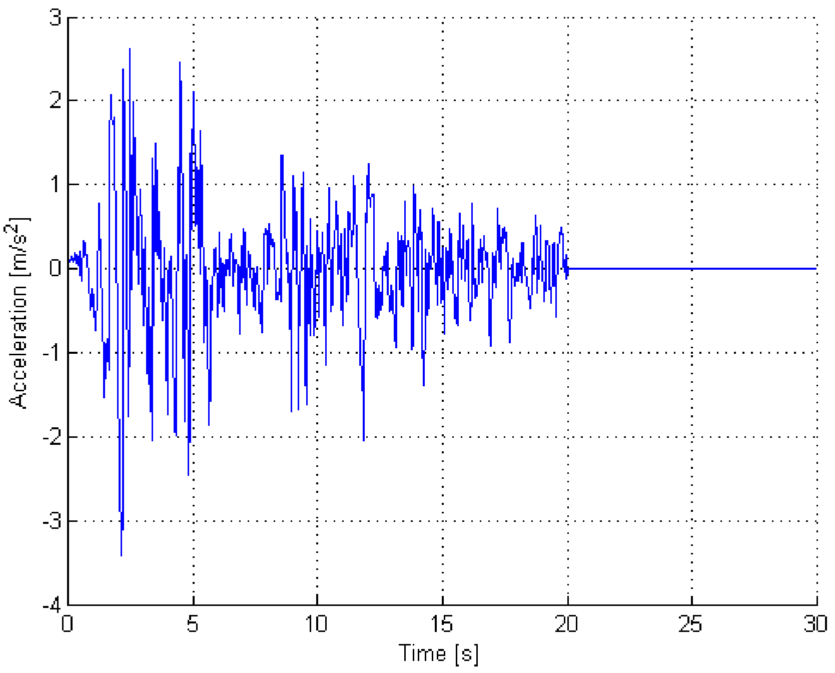


FIGURE 3.6: EL CENTRO EARTHQUAKE RECORD (NORTH-SOUTH COMPONENT)

3.3 Test Cases

The first test case is a single element with two nodes as shown in Figure 3.7. The first node is held laterally, vertically and rotationally. The second node has a lateral acceleration applied to it. This simple test case has an analytical solution and is well known. Thus, it is a good starting point.

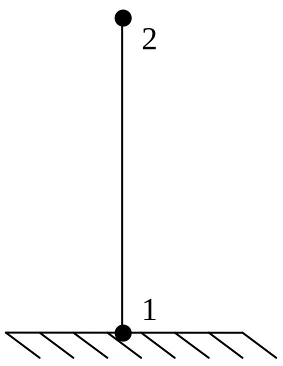


FIGURE 3.7: SINGLE ELEMENT TEST CASE

The second test case uses two elements and three nodes arranged as in Figure 3.8. As in the first case the first node is held and the other two nodes have lateral accelerations applied to them. This is a modest, but controllable addition of complexity to the first case.

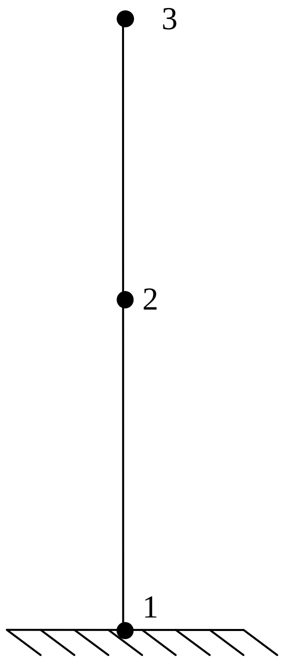


FIGURE 3.8: TWO ELEMENT TEST CASE

The third case uses three elements to form a single floor frame. There are two vertical elements and one horizontal arranged as in Figure 3.9. Nodes 1 and 2 are held while nodes 3 and 4 have identical lateral accelerations applied. This test case is a simple partial frame, which is the "base" element of more complex framed structures. Hence, solution of this case is readily extended to other cases.

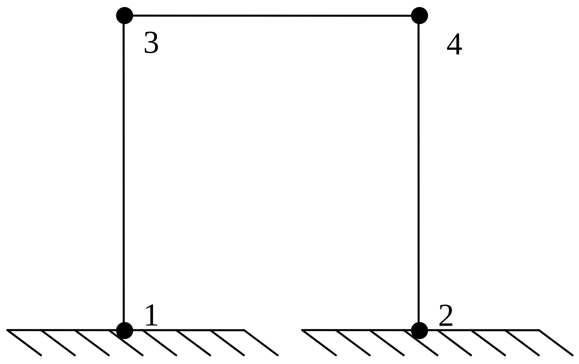


FIGURE 3.9: SINGLE FLOOR TEST CASE

The final case considered is a frame developed in the SAC Project. The SAC Project produced structures designed for Los Angeles, Seattle and Boston (Krawinkler & Gupta, 1998). The structure used here is the SAC-3 model. This model is a 3-bay, 3-story building shown in Figure 3.10 with columns and beams defined in Table 3.1.

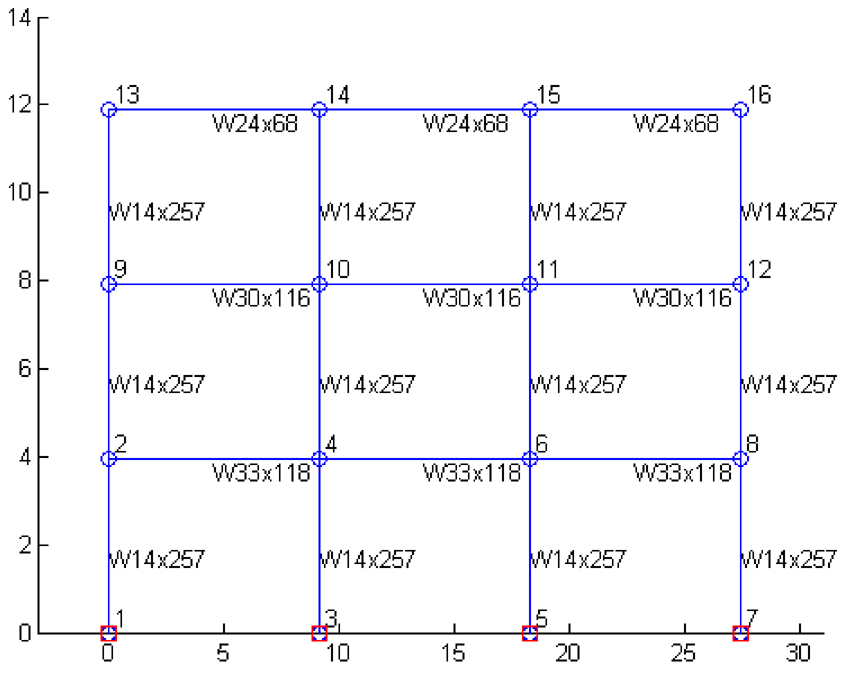


FIGURE 3.10: SAC-3 STRUCTURE MESH

TABLE 3.1: SAC3

Name	Area [m^2]	Second Moment of Area [m^4]
W24x68	1.297×10^{-2}	7.617×10^{-4}
W30x116	2.206×10^{-2}	2.052×10^{-3}
W14x257	4.877×10^{-2}	1.415×10^{-3}
W33x118	2.239×10^{-2}	2.456×10^{-3}

3.4 Test System

MATLAB was run using the “-singleCompThread” switch to avoid any issues with multi-threading. Both numerical solutions would likely benefit from multi-threaded coding but this was outside the scope of this thesis. In the case of a Monte Carlo analysis parallelising the code would not be required as multiple cases could be run in parallel with each case executed as a single thread. This represents a highly parallelisable problem (Berkman, Galil, Schieber, & Vishkin, 1989).

The test system was as follows:

Software:	MATLAB 2009b 64bit
Operating System:	Microsoft Windows 7 64bit
Hardware:	Intel Core 2 Quad Q9300 @ 3.60 Ghz

4 RESULTS

4.1 TEST CASES

The set up code for each of the test cases are given in APPENDIX B: Test Case Code. The manual for using the set of scripts used in the test case code is given in APPENDIX A: Manual.

The first test was to determine the most accurate and efficient ODE solver for Equations (2.5.10) and (2.5.11). For this task the SAC-3 building was used with one element per column as this is one of the more complex building models used to test the Newmark-β algorithm. The ground acceleration input used was the North-South component of the El Centro earthquake.

The first storey nodes had some discrepancy between the ODE numerical solutions, as shown in Figure 4.1. In this figure, the ode45 method ends up with a slightly different result to the other three methods. Ode45 is an explicit Runge-Kutta algorithm and is generally considered the best starting point. However, it is not well suited to the very tight tolerances used here when compared with ode113 and ode15s. The relative tolerance used here was 10^{-10}, whereas ode45 is considered more efficient when using the default of 10^{-3} and where only a medium level of accuracy is required. In contrast, ode113 is well suited to problems requiring high accuracy and where the functions are expensive to calculate (Shampine & Reichelt, 1997). The computational speed results for each solver are given in Table 4.1.

The maximum error in the ode45 solution compared with the others in this case is 5%. The ode45 algorithm is not considered appropriate for problems requiring high accuracy (MathWorks, 2010). In this case the ode45 solution has not accurately resolved one of the early turning points and thus there was a significant error introduced on subsequent unloading and loading of the element. This need for accurately finding the turning points in the hysteresis loops is further demonstrated in Figure 4.4. The high accuracy required at the hysteresis loop turning points precludes ode45 as a valid algorithm for this problem.

In the linear portions of the response all solvers give equivalent solutions. When there is significant non-linear behaviour there is significant error in one of the algorithms tested. This demonstrates the need for an appropriate solver for problems with large contributions from non-linear effects.

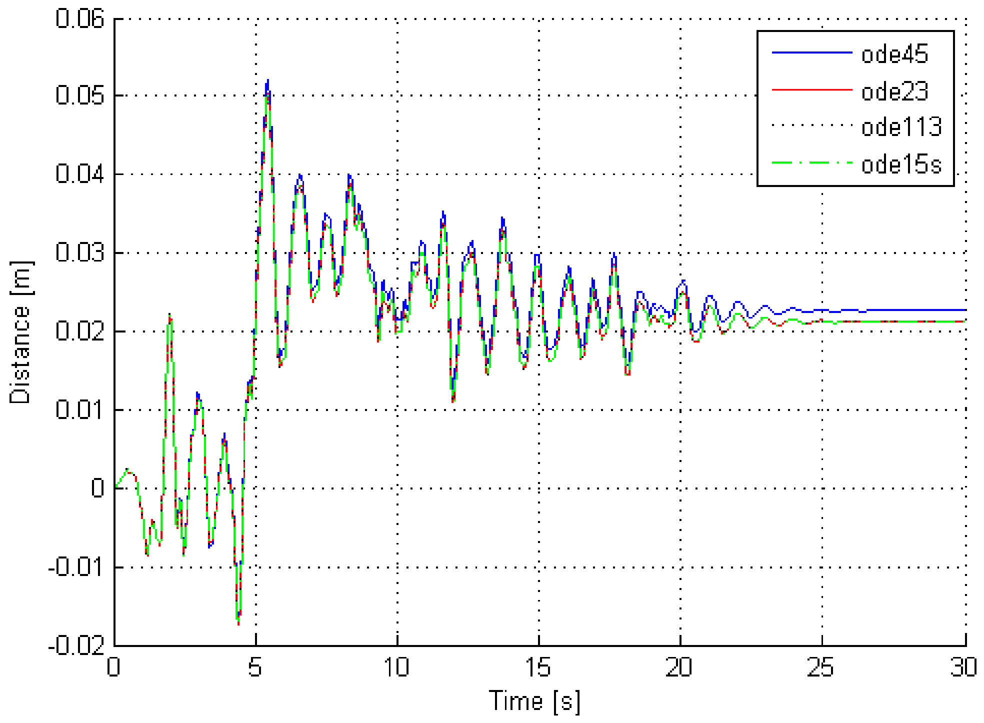


FIGURE 4.1: HORIZONTAL DISPLACEMENT OF FIRST STOREY NODES. ODE45 DEVIATES FROM THE OTHERS WHICH ALL OVERLAP

TABLE 4.1: TIME RESULTS FOR THE ODE SOLUTION BENCHMARK

Solver	Solution Time [s]
ode45	292
ode23	2047
ode113	263
ode15s	880

Table 4.1 shows that the ode113 method is slightly faster than ode45 and yields the same result as the other two numerical solutions. This outcome gives confidence that ode113 is giving the correct solution. Therefore, from this point on, the comparisons between the ODE solution and the Newmark solution are done using ode113.

These times are very different with ode23 taking an order of magnitude longer than the others. This result comes from the solvers actually being completely different. The MATLAB interface for each of these solvers is identical but the solvers themselves perform the numerical integration in different ways. They are therefore appropriate for different kinds of problems.

The first test after selecting the ODE solver to use was the single element case shown in Figure 3.7. The benchmark results are given in Figures 4.2 and 4.3. Figure 4.2 shows the results using the synthetic input and Figure 4.3 shows the results from the El Centro ground acceleration input. The ODE code was run once in each case with the time taken represented by the solid red line. The Newmark-β code was run with different time step values and the difference between each Newmark-β solution and the equivalent ODE solution were calculated to give the error line shown in blue with a cross at each data point. The time required for the Newmark-β runs as the time step is reduced is given by the dashed green line.

The synthetic acceleration input was solved more quickly by the ODE solver than the Newmark-β solver with a time step of 10^{-3} seconds. As the synthetic acceleration gave such a simple response with few turning points in the hysteresis curve the Newmark-β code gave little error. As more turning points are introduced the Newmark-β solution will tend to compound errors at each point. This can lead to very large errors depending on where the turning points are in relation to the time steps the numerical integration occurs at. If the turning point is half way between two time steps the error can be very large. If the turning points are very close to the time steps the error can be much lower.

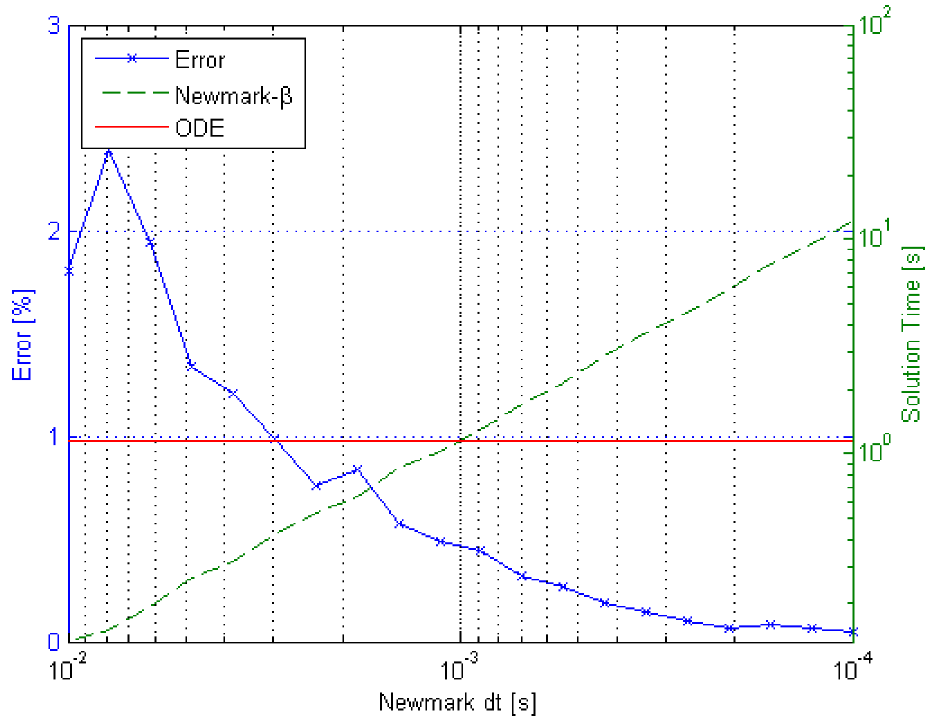


FIGURE 4.2: SINGLE ELEMENT RESULTS WITH SYNTHETIC INPUT OF FIGURE 3.5 THE SOLID RED LINE SHOWS THE TIME FOR THE ODE SOLUTION

The synthetic input gives an almost steadily reducing error in the Newmark-β solution with decreasing solution time step. This result is very favourable for the Newmark-β algorithm as it gives trust in the result. When the error is erratic the result cannot be trusted.

This favourable result is likely due to both the simple model consisting of only a single element and also the simple input acceleration in Figure 3.5. As is seen later in this chapter, this steady decrease in error is no longer seen when a more complex model or input acceleration are used.

The El Centro ground motion input gave a much more complex output with many turns in the hysteresis curve. This more complex input and response caused the ODE code to take almost as long as the Newmark-β solution with a time step 10^{-3} seconds. It also caused much greater errors to appear in the Newmark-β result, indicating its potential unsuitability in such large, non-linear cases. As there were more turns of the hysteresis loop in the non-linear portion of the response, there was much greater opportunity for small errors in the Newmark-β result to compound as is demonstrated in Figure 4.4.

It is important to note that the low points in the error of the Newmark-β solution with relatively large time steps are not necessarily reliable indicators. They are the result of chance. By changing the time step, the points at which each step occurs can change. This can put the points the closer or further away from the real turns in the hysteresis loops. The result of this is that sometimes a larger time step can give a lower error as the turns in the hysteresis loop were more closely modelled.

When solving with Newmark-β the time step is usually progressively reduced until two successive results agree within some pre-set tolerance. This approach means that these low points would not be used as the final result in a Newmark-β analysis. In this case the Newmark-β code requires a time step of $1.8\text{x}10^{-3}$ seconds or less to get a reliable error of less than 1%.

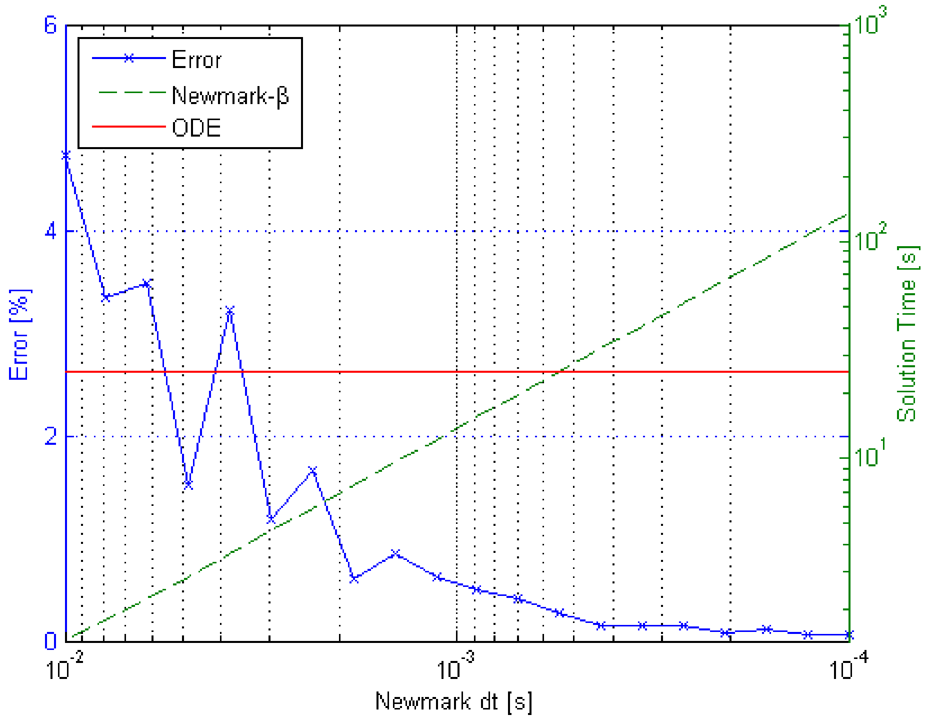


FIGURE 4.3: SINGLE ELEMENT RESULTS WITH EL CENTRO GROUND MOTION INPUT THE SOLID RED LINE SHOWS THE TIME FOR THE ODE SOLUTION

The erratic reduction of error in this simple single element test case calls for a closer look at the effect of turning point accuracy on the error. Figure 4.4 shows this importance of accuracy in finding the turning point in the hysteresis loop on the subsequent response after the turning point. The single element case was again considered this time with the input acceleration in Figure 4.6. The main feature of this acceleration vector is the sharp change from negative to positive acceleration which will cause a sudden change in direction. The element is then released, allowing the element to return after experiencing this turn in the hysteresis curve.

Although it appears as though the Newmark-β solution turns at almost the same time and displacement as the ODE code at just past 0.4 seconds, the slight difference (1.2%) create a large difference in the following unloading of the element. At 0.8 seconds the error has reached over 17% which is very significant and this error carries on to the end of the analysis.

Figure 4.5 shows this point more clearly. When the Newmark-β code is used with a much smaller time step, of 5x10^{-4} the turning point is more accurately identified and the subsequent response agrees with the ODE result.

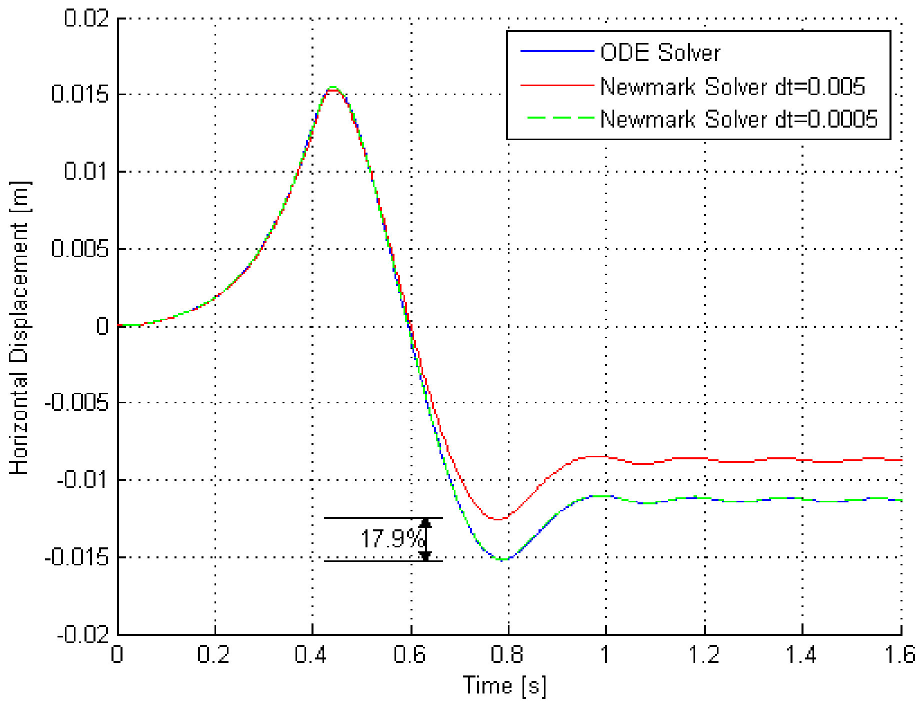


FIGURE 4.4: DEMONSTRATION OF THE NEED FOR TURNING POINT ACCURACY USING A SINGLE ELEMENT AS IN FIGURE 3.7 WITH THE INPUT ACCELERATION VECTOR GIVEN IN FIGURE 4.6

The turning point at 0.4 seconds is shown in Figure 4.5. This zoomed in figure clearly shows the small, 1.2%, difference in horizontal displacement between the two Newmark-β solutions. It is also important to note that the turn happened at the same time, so the only discrepancy is the 1.2% displacement error.

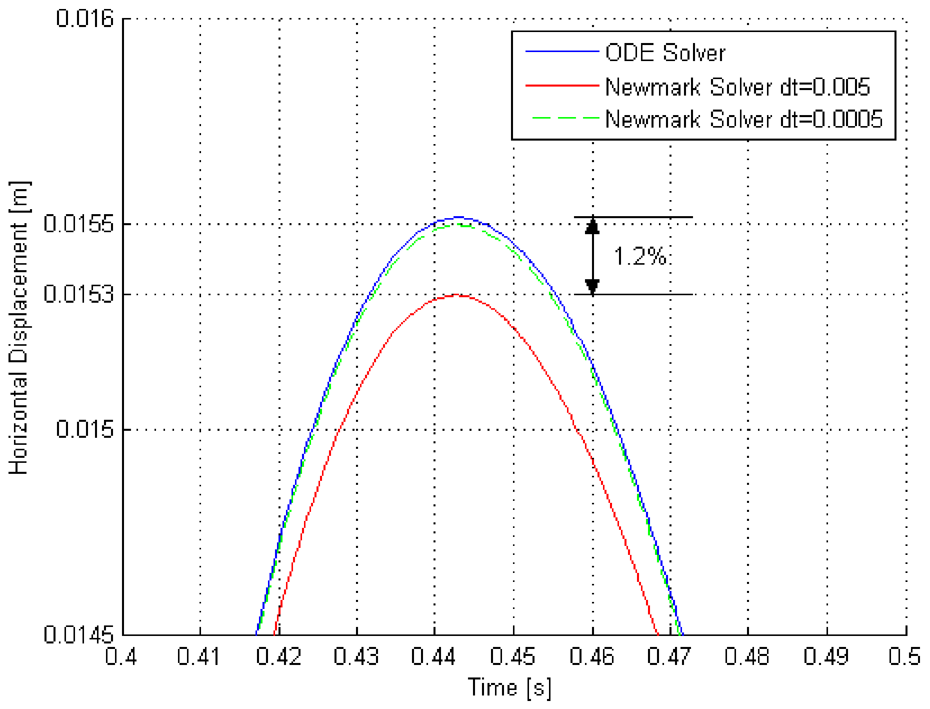


FIGURE 4.5: CLOSE UP OF THE FIRST TURNING POINT FROM FIGURE 4.4

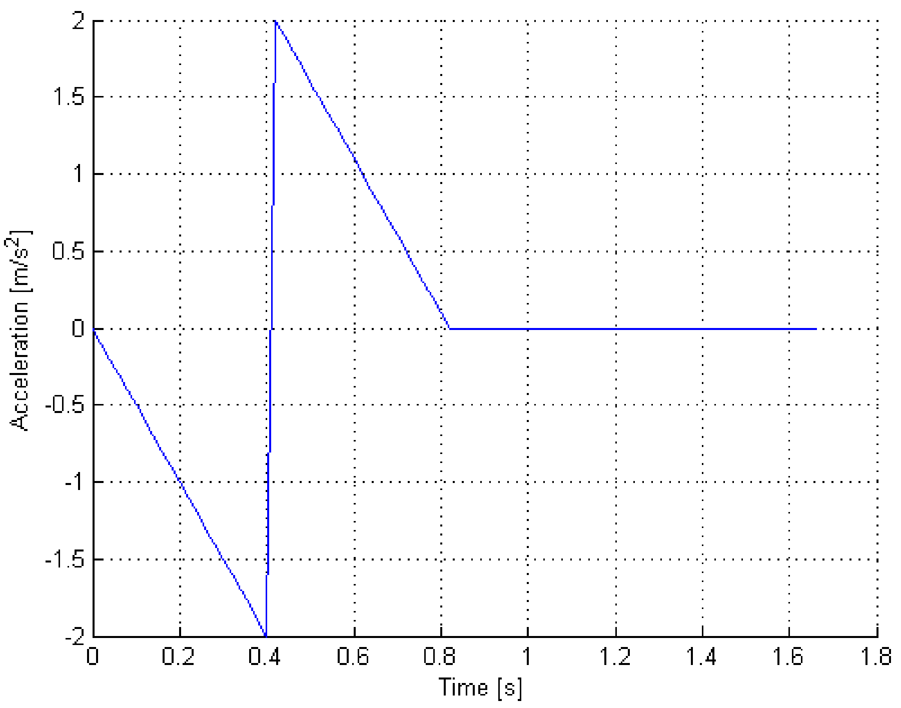


FIGURE 4.6: TURNING POINT DEMONSTRATION INPUT ACCELERATION VECTOR

The second test case with two elements depicted in Figure 3.8 increased the number of degrees of freedom and thus the solve time. It also added to the need for accuracy of the motion of the first floor node as the motion of the second node depends on it. The benchmark results for the synthetic input are given in Figure 4.7 and El Centro results are given in Figure 4.8. The El Centro input gave a complicated output with many turns in the hysteresis loops. The time step required for the Newmark-β solution was very small. The small time step was needed as any error in either element would compound causing substantial discrepancies in the final result.

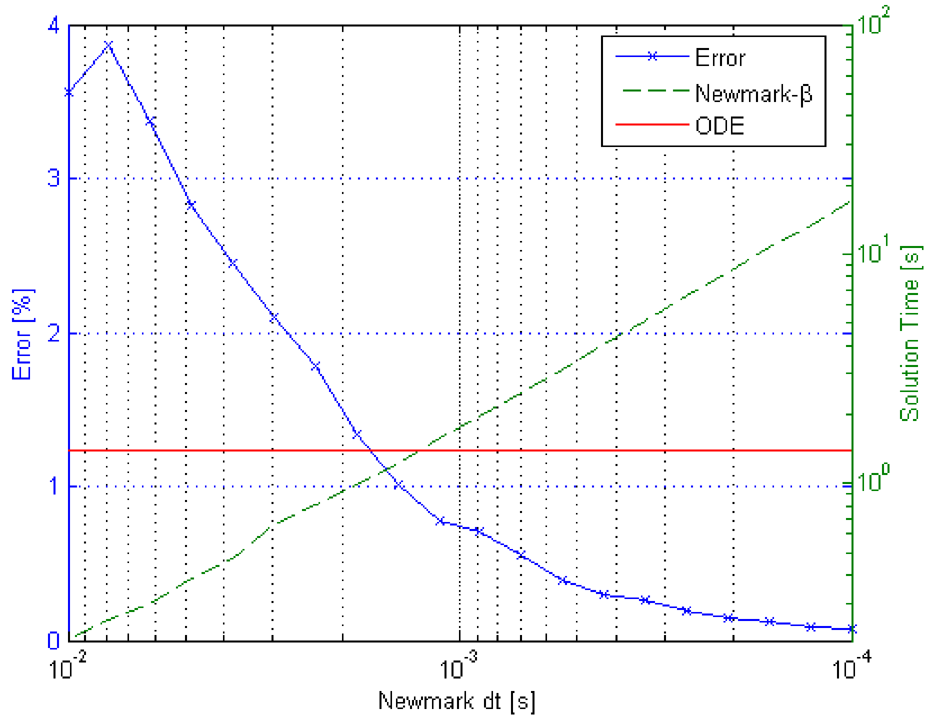


FIGURE 4.7: TWO ELEMENTS RESULTS WITH SYNTHETIC INPUT OF FIGURE 3.5 THE SOLID RED LINE SHOWS THE TIME FOR THE ODE SOLUTION

As with the synthetic input for the single element case in Figure 4.2 the errors in the Newmark-β solutions are mostly decreasing with decreased solver time step. This is again due to the simple nature of both the model and the input acceleration. There are only a few turns in the response and so less turns for the Newmark-β solution to miss.

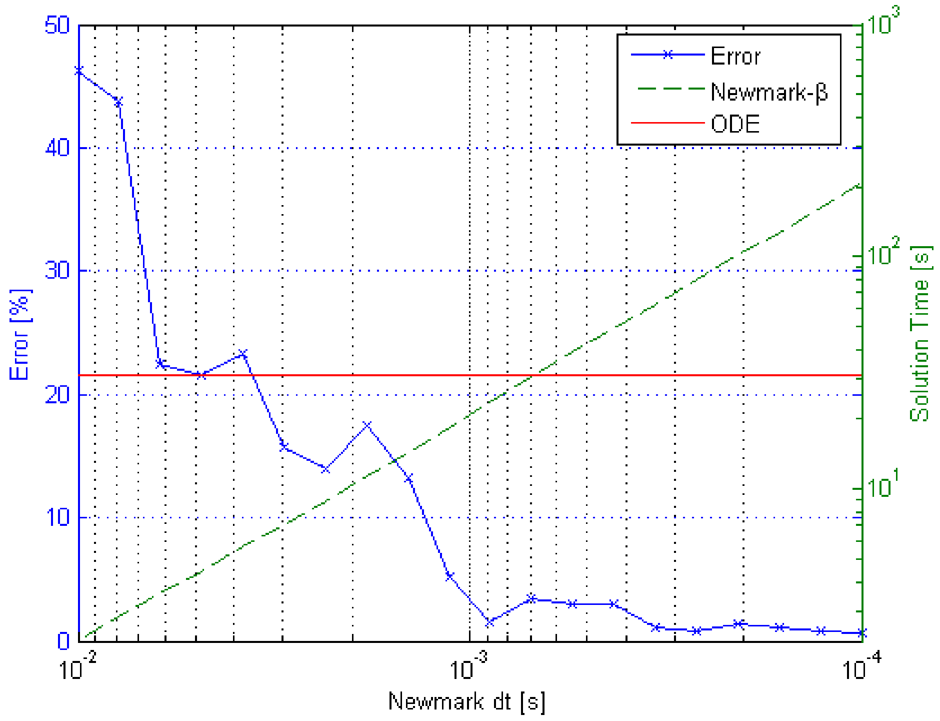


FIGURE 4.8: TWO ELEMENTS RESULTS WITH EL CENTRO GROUND MOTION INPUT THE SOLID RED LINE SHOWS THE TIME FOR THE ODE SOLUTION

In contrast to the synthetic results, the El Centro ground motion input errors in Figure 4.8 have multiple false minimums but still overall tends towards agreement with the ODE solution. Of particular note is the two points with very similar errors around a time step of 4×10^{-3}. These could lead a user to think the result is correct when in fact it differs from the true solution by more than 20%. These similar errors are likely caused by the time steps landing in similar places around the turning points in the hysteresis curves of the two elements. There is also a case with three errors very close to each other around a time step of 3×10^{-4}. Once again there is still substantial error in the solution even though these successive Newmark-β solutions agree and differ in time step by a factor of around two.

The error in the Newmark-β results in Figures 4.7 and 4.8 are very different. This shows the Newmark-β error to change based on the input acceleration used, it is input dependant.

Therefore, if multiple acceleration inputs are to be used on a given model a convergence analysis must be done for each one to ensure accuracy.

The false minimums in Figure 4.8 also occur in some of the other test cases and represent a large pitfall in use of the Newmark-β algorithm. For an analyst to avoid using one of these false minimums as the solution a much smaller time step must be used in addition to the one that found a sufficiently accurate result. This adds substantially to the computational time for a Newmark-β analysis. Even though an adequate result might be found at a time step of 10^{-4} seconds, the user needs to try something much smaller and check the two solutions agree to some predefined tolerance in order to have confidence in the result.

As a follow up to the two element case the actual displacements are compared to the ODE solution for two different Newmark-β time steps in Figure 4.9. Although the larger time step gives a sensible looking result it is not accurate. The errors are 13.6% in magnitude. In addition the peaks are reached at different times separated by 0.7 seconds. Both errors are potentially very significant.

This large discrepancy is caused by the non-linear effect not being modelled accurately. The Newmark-β algorithm requires a very small time step even in this simple case due to the large contribution of the non-linear effects. The demonstration shows how easily the Newmark-β algorithm can lead an analyst astray. Sensible looking results were obtained with a time step of 10^{-3} seconds but they were quite different to the result using a time step of 10^{-4} seconds.

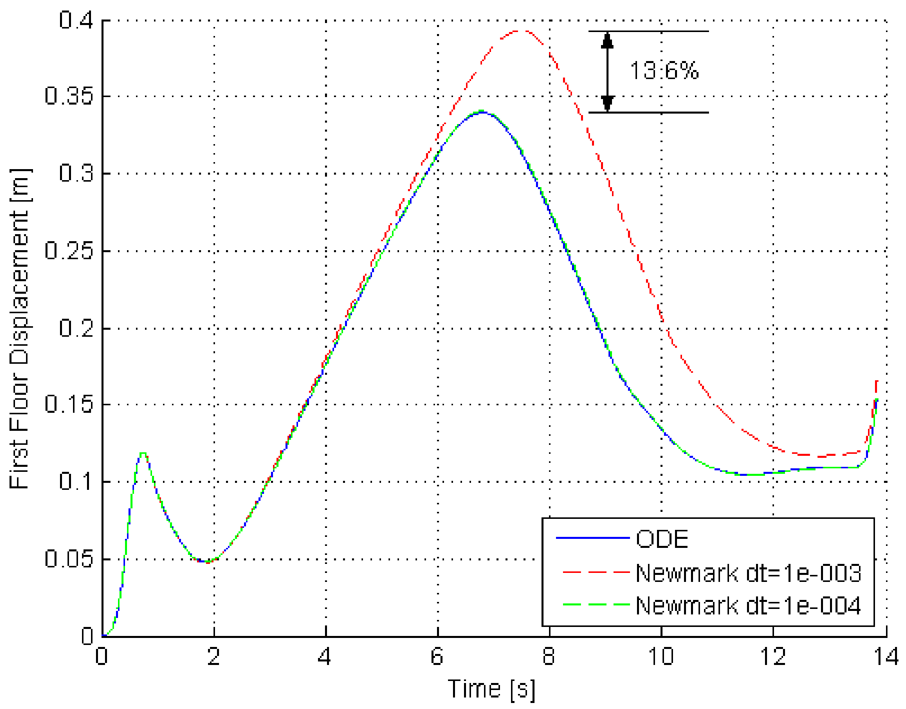


FIGURE 4.9: DEMONSTRATION OF SIMILAR BUT INCORRECT NEWMARK-β RESULT

The next test case is the single floor model made up three elements connecting four nodes, as shown in Figure 3.9. The results for the synthetic input and El Centro input are given in Figures 4.10 and 4.11 respectively. As with the two element test case the synthetic input was quick to solve with both methods. Also, as with the two element case, the El Centro input required very small time steps to achieve acceptable levels of accuracy.

The error in the El Centro result is much higher than the simple synthetic input. As with the previous test cases this is due to the large number of turns in the hysteresis loops. With a small error at each of these turns the total error at the end of the analysis can be very significant. This means a very small time step is required to achieve a reliable result from the Newmark-β algorithm when the non-linear effects make a significant contribution to the response.

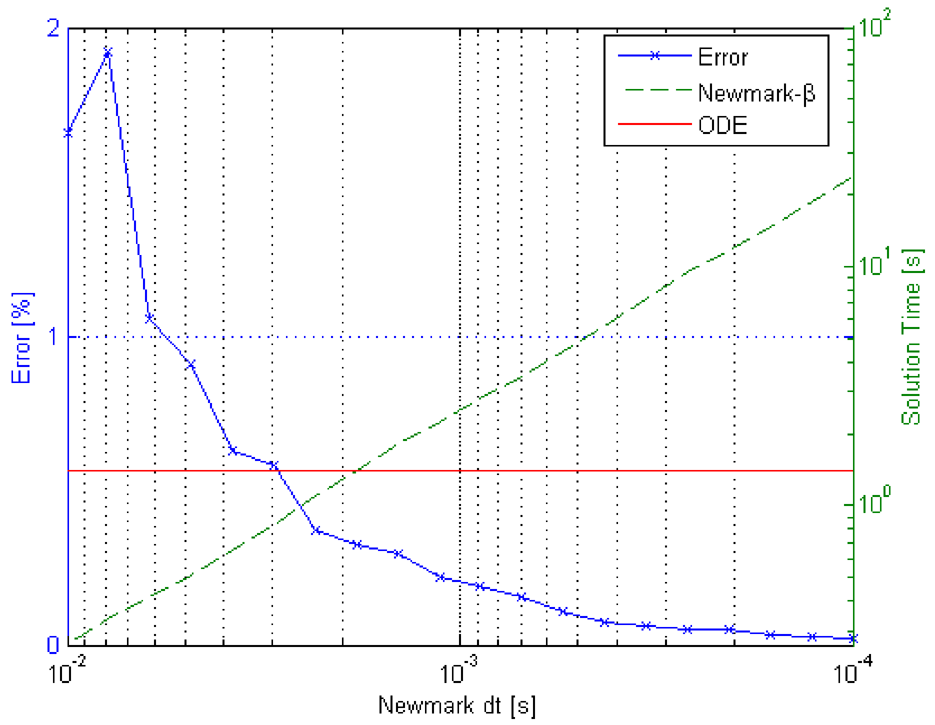


FIGURE 4.10: SINGLE FLOOR RESULTS WITH SYNTHETIC INPUT OF FIGURE 3.5 THE SOLID RED LINE SHOWS THE TIME FOR THE ODE SOLUTION

As with the single element and two element test cases with synthetic input the errors are decreasing steadily as the time step is reduced. This single floor test case is still a simple model as it closely resembles a single element with a weight at the top node. The result of this is a steady convergence of the Newmark-β solution to the true solution.

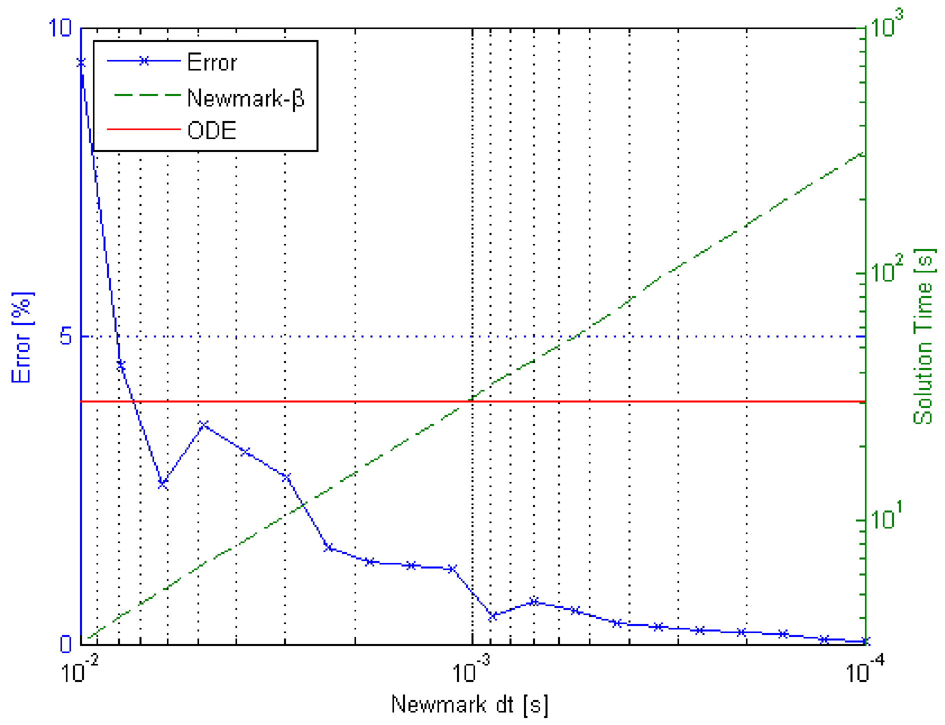


FIGURE 4.11: SINGLE FLOOR RESULTS WITH EL CENTRO GROUND MOTION INPUT THE SOLID RED LINE SHOWS THE TIME FOR THE ODE SOLUTION

The final test case is the SAC3 building shown in Figure 3.10. The SAC3 building model was particularly difficult for the Newmark-β algorithm to accurately follow, with large non-linear effects. This outcome is likely due to the number of elements in the model all depending on the displacement of the other elements. As seen in Figures 4.7 and 4.8 in this chapter, when non-linear deformations are inaccurate, these errors may flow-on to connected elements. Therefore, any slight inaccuracy in one would be compounded in this more complex model.

The accuracy of the ODE solution was not affected by this issue and gave consistent and thus accurate results much more quickly than the Newmark-β solution. The analysis was run for both the synthetic input (Figure 4.12) and the El Centro input (Figure 4.13). These figures show the Newmark-β solution converging to the ODE solution but only after the time step had dropped below 10^{-4}s. At this point the Newmark-β solution was taking more than five times as long as the ODE solution.

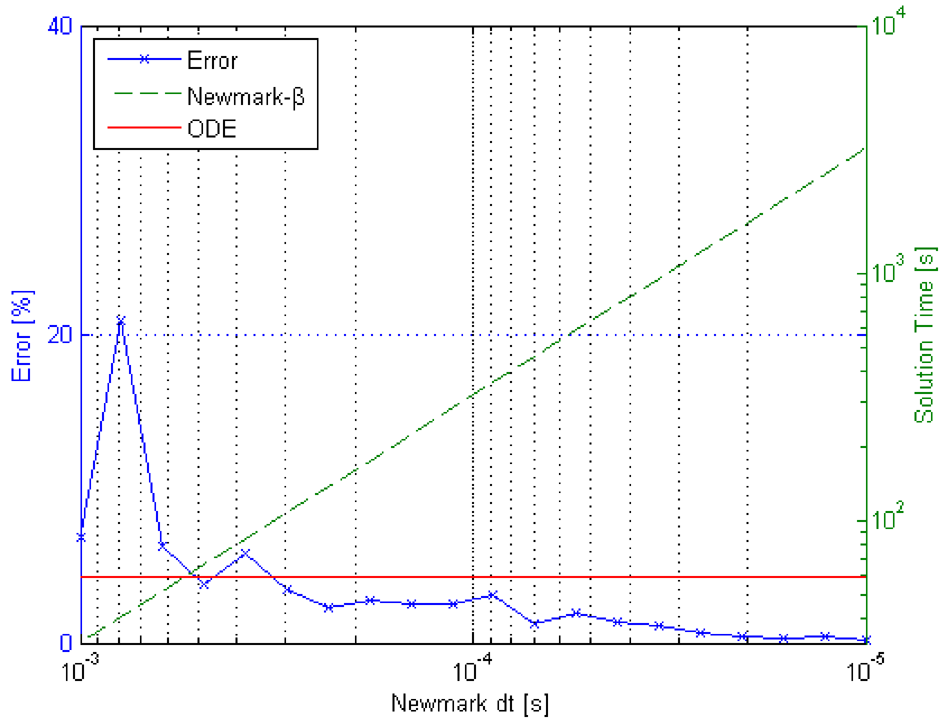


FIGURE 4.12: SAC3 RESULTS WITH SYNTHETIC INPUT OF FIGURE 3.5 THE SOLID RED LINE SHOWS THE TIME FOR THE ODE SOLUTION

Unlike the previous examples the SAC-3 building model is not simple. It consists of many elements all dependent on the movements of the other elements. The complex nature of the model results in an error curve that does not decrease steadily, even with the simple synthetic input. It has spikes and plateaus that throw doubt on the reliability of the result. As the time step is dropped below 10^{-4} seconds the errors start to decrease reliably and the result can be trusted. At this point the Newmark-β solution is taking more than five times longer than the ODE solution.

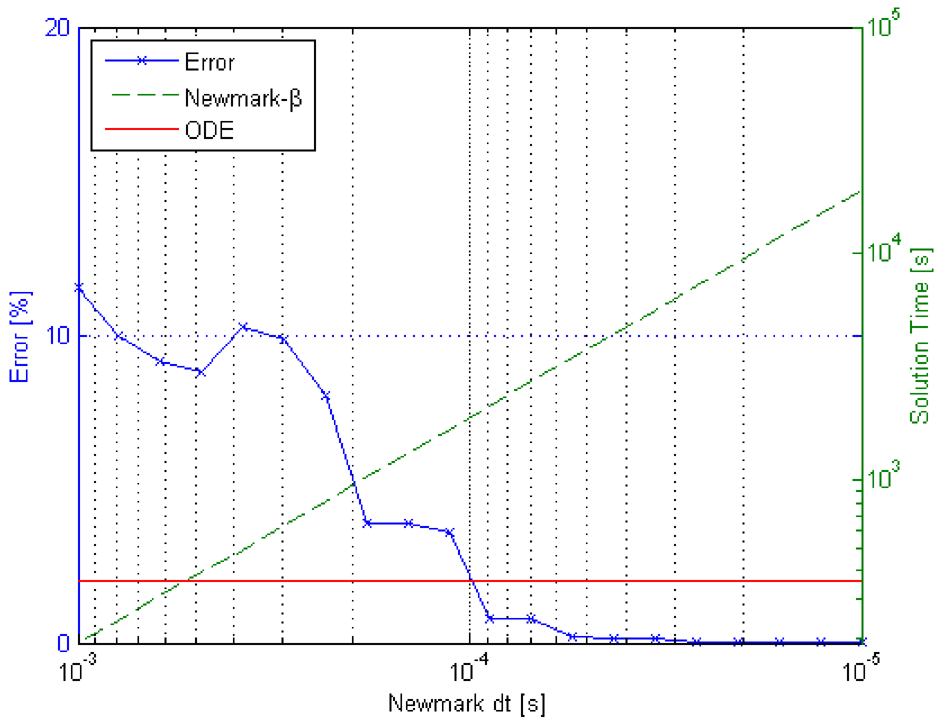


FIGURE 4.13: SAC3 RESULTS WITH EL CENTRO GROUND MOTION INPUT THE SOLID RED LINE SHOWS THE TIME FOR THE ODE SOLUTION

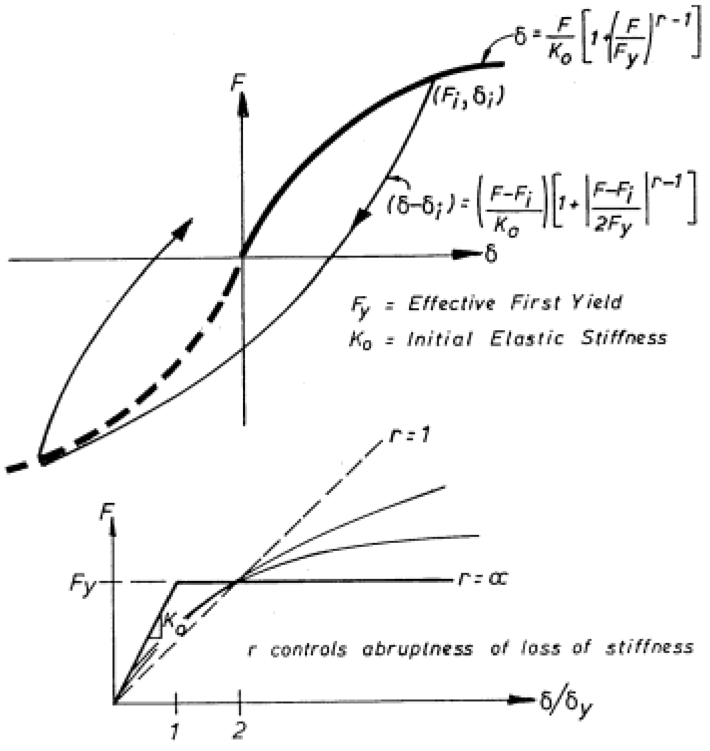
There are false minimums in the results from both input accelerations and the convergence again shows input dependence in the error of the Newmark-β algorithm. The enhanced complexity of the model has increased both of these effects. The synthetic input in Figure 4.12 shows the result appearing to converge at around 10^{-4} seconds. An analyst might use a time step of 10^{-4} seconds for further analysis thinking the solution had converged to within 0.5%. In reality the error is over 3% for the two input accelerations shown here.

The preceding figures (Figures 4.1 to 4.13) all demonstrate the ODE algorithm's consistency and accuracy. In every case, the Newmark-β solution converges to the ODE solution as the time step is reduced. This convergence adds confidence and reliability to the ODE algorithm.

These figures were all produced with a yield force, F_y, chosen to ensure some non-linear behaviour and permanent deflection. That is, the final displacement is different to the initial displacement. However, the amount of non-linear behaviour greatly affects the error of the

Newmark-β algorithm, and is a very significant result. To demonstrate this problem the single element test case was used with the synthetic input acceleration and a selection of different yield forces. The results are given in Figure 4.15.

The Newmark-β algorithm assumes that the stiffness remains constant over the time step. Looking at Figure 1.1 (reproduced here as Figure 4.14) we can see that the stiffness changes rapidly as the Ramberg-Osgood constant increases and as the force in the element approaches or exceeds the yield force. This result means that smaller time steps are required to model the areas where the stiffness is changing rapidly, such as when the hysteresis loop reaches corners or turning points. A result reinforced by the prior results in this chapter.



$\delta = \frac{F}{K_o}\left[1+\left|\frac{F}{F_y}\right|^{r-1}\right]$
(F_i, δ_i)
$(\delta-\delta_i) = \left(\frac{F-F_i}{K_o}\right)\left[1+\left|\frac{F-F_i}{2F_y}\right|^{r-1}\right]$
F
δ
F_y = Effective First Yield
K_o = Initial Elastic Stiffness
r = 1
F
Fy
r = ∞
K_o
r controls abruptness of loss of stiffness
δ/δ_y
1
2

FIGURE 4.14: RAMBERG OSGOOD HYSTERESIS MODEL (CARR, 2008)

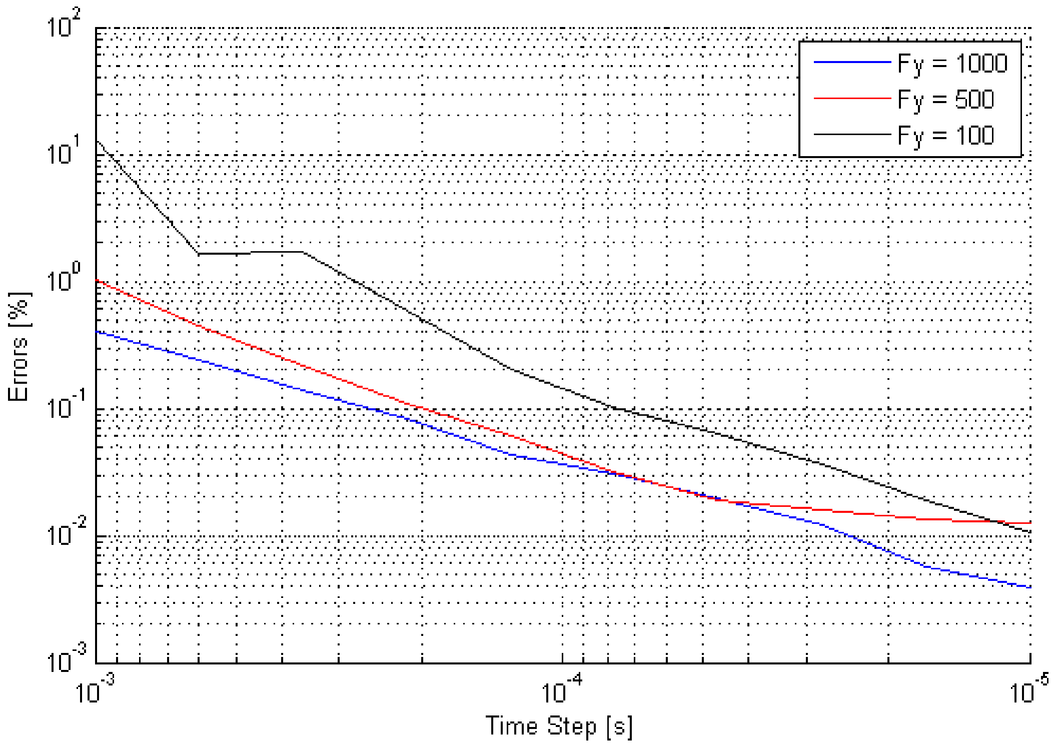


FIGURE 4.15: DEMONSTRATION OF THE EFFECT OF YIELD FORCE ON ERROR FOR NEWMARK-Β

The ODE solution times are given in Table 4.2. From this table, it is clear that increased non-linear behaviour has no real impact on solution time for the ODE solver. The ODE solution does not take any longer because it is always finding the turning points very accurately, the additional non-linear contribution and thus dependence on this accuracy has no impact. For reference and comparison, the time taken for the Newmark-β solution at a few time step values are also given in Table 4.3.

TABLE 4.2: SOLUTION TIMES FOR THE ODE ALGORITHM

F_y	**Solution Time [s]**
1000	1.2
500	1.1
100	1.1

TABLE 4.3: SOLUTION TIMES FOR THE NEWMARK-β ALGORITHM

Time Step [s]	Solution Time [s]
10^{-3}	0.4
10^{-4}	3.5
10^{-5}	34.5

4.2 Chapter Summary

The first result of note is that all test cases verify the consistency of the ODE algorithm with the Newmark-β algorithm. As the time steps used in the Newmark-β algorithm were decreased the difference between the two algorithm's results always tended towards zero. This verifies that the ODE algorithm is trustworthy.

The error in the Newmark-β solution depends not only on the model but also on the input acceleration. This result means that a time step that gave an adequate result for a given model and acceleration input is not guaranteed to give a similarly adequate result for a different acceleration input on the same model. For the analyst this means that a separate convergence analysis must be done for each input acceleration used on a model.

The convergence analysis must be done thoroughly for the Newmark-β algorithm. This was highlighted by the false minimums in error that arose for successive Newmark-β time steps. This combined with the previous point requiring that a convergence study be done for every input acceleration used on a model adds substantial time to the Newmark-β analysis. The total time for a Newmark-β analysis is therefore much greater than the time taken for a single adequately accurate result.

5 SUMMARY AND CONCLUSIONS

This thesis explores an alternative approach to dynamic non-linear seismic analysis of building models. This approach centres on there being more efficient methods of numerical integration than Newmark-β which is typically used.

The reliability of the ODE solution is verified with several test cases against a constant average acceleration Newmark-β scheme. The results from Newmark-β scheme always tended towards the ODE solution as the time step was reduced. This shows the ODE solution to be trustworthy.

The main outcome of this thesis is that using more efficient alternatives to Newmark-β can indeed give substantial performance improvements for non-linear analysis of building models. The results show that when the non-linear behaviour has a significant impact on the result the alternative method presented here is much faster. Additionally the ODE solution presented in this thesis gave solutions that were far more accurate than the Newmark-β solution gave with reasonable time steps.

The most computationally intensive part of the algorithm is parallelisable and so performance increases can be had on multi threaded systems. Alternatively, for running multiple test cases, each could be run as a single thread with many being run simultaneously. Either option would work with the later likely being faster as the sequential parts of the code would also be run in parallel. Development would also be much simpler as coding with multiple threads is complex.

6 Future Work

This thesis shows that for dynamic non-linear seismic analysis there are significant performance gains to be had in using efficient numerical integration. In order to take advantage of this the methods presented here would need to be extended.

To make this solving method more useful it needs to support more element types. More hysteretic models should also be supported. Adding more element types should be simple and require little or no changes to the inner algorithm. Changes in the hysteretic models however would require continuous derivations of those models. These would then have to be handled inside the inner algorithm. As long as a continuous derivation of the required model exists this should be achievable.

There are possibly more performance increases to be had by using an alternative ODE solver. In this thesis four solutions are benchmarked to find the most efficient but there exist many others as well as tweaks that can be done to the solvers used here. Although it is unlikely that a very large performance increase is to be had it is likely still worth investigating.

There are also other non-linear effects that can be modelled in dynamic seismic analysis. These include soil interaction and P-Delta effects (Lindeburg & Baradar, 2001). The P-Delta effects are related to the vertical load on nodes having shifted from the original position causing a destabilising moment.

Finally the whole program needs to be ported to a compiled language such as C or FORTRAN. This will result in a much more efficient program in terms of both CPU usage and memory usage. The algorithm also lends itself to parallelisation to allow for efficient use of systems with many threads.

7 References

Amestoy, P. R., Duff, I. S., L'Excellent, J. Y., & Li, X. S. (2001). Analysis, Tuning and Comparison of Two General Sparse Solvers for Distributed Memory Computers. *ACM Trans. Math. Softw. , 27* (4), 388-421.

Berkman, O., Galil, Z., Schieber, U., & Vishkin, U. (1989). Highly parallelizable problems. *Annual ACM Symposium on Theory of Computing* , 309-319.

Bruneau, M., Uang, C.-M., & Whittaker, A. (1998). *Ductile design of steel structures.* New York, NY, USA: McGraw-Hill.

Carr, A. J. (2008). *Ruaumoko Manual* (Vol. 5). Christchurch, New Zealand: University of Canterbury.

Clough, R. W., & Penzien, J. (1975). *Dynamics of Structures.* New York: Mcgraw-Hill College.

Dormand, J. R., & Prince, P. J. (1980). A family of embedded Runge-Kutta formulae. *Journal of Computational and Applied Mathematics , 6* (1), 19-26.

Farid, F. O. (1995). Criteria for invertibility of diagonally dominant matrices. *Linear Algebra and its Applications , 215*, 63-93.

Hann, C. E., Chase, J. G., & Shaw, G. M. (2005). Efficient implementation of non-linear valve law and ventricular interaction dynamics in the minimal cardiac model. *Computer Methods and Programs in Biomedicine* (80), 65-74.

Hosea, M. E., & Shampine, L. F. (1994). Efficiency Comparisons of Methods for Integrating ODEs. *Computers & Mathematics with Applications , 28* (6), 45-55.

Krawinkler, H., & Gupta, A. (1998). Story Drift Demands for Steel Moment Frame Structures in Different Seismic Regions. *Proc. 6th US National Conference on Earthquake Engineering.* Seattle, Washington.

Lindeburg, M. R., & Baradar, M. (2001). *Seismic Design of Building Structures: A Professional's Introduction to Earthquake Forces and Design Details* (8th ed. ed.). Belmont, CA: Professional Publications, Inc.

Lu, L.-Y., Chung, L.-L., Wu, L.-Y., & Lin, G.-L. (2006). Dynamic analysis of stuctures with frictional devices using discrete-time state-space formulation. *Computers and Structures* (84), 1049-1071.

Mathworks. (2010, May 10). *Left or right matrix division*. Retrieved May 10, 2010, from http://www.mathworks.com/access/helpdesk/help/techdoc/ref/mldivide.html

MathWorks. (2010, May 10). *Solve initial value problems for ordinary differential equations*. Retrieved May 10, 2010, from The Mathworks: http://www.mathworks.com/access/helpdesk/help/techdoc/ref/ode23tb.html

Newmark, N. M. (1959). A Method of Computation for Structural Dynamics. *J. Engrg. Mech. Div. ASCE* , 67-94.

Palermo, D., & Vecchio, F. J. (2004). Compression Field Modeling of Reinforced Concrete Subjected to Reversed Loading: Formulation. *ACI Structural Journal* (100-S64), 155-164.

Ramberg, W., & Osgood, W. R. (1943). Description of stress-strain curves by three parameters. *Technical Note No. 902* .

Shampine, L. F., & Reichelt, M. W. (1997). The Matlab ODE Suite. *SIAM Journal of Scientific Computing , 18* (1), 1-22.

Shapine, L. F., & Gordon, M. K. (1975). *Computer Solution of Ordinary Differential Equations: the Initial Value Problem.* San Francisco: W. H. Freeman & Co Ltd.

APPENDIX A: MANUAL

A.1 MESHING

There are two approaches to creating the mesh with boundary conditions. The first is to use the provided meshing code. This is a quick and easy way to create large building models quickly, the steps for this are:

1. Choose the number of: floors, bays, elements per column. Define the width of each bay, the height of each floor and the additional load applied to each floor if any. For example:

```
% mesh variables
floors = 3;
bays = 3;
elms_per_col = 1;                       % [m]
bay_width = 9;                          % [m]
floor_height = 4;                       % [m]
floor_load = [4e3; 4e3; 5e3];           % [kg]
```

2. Define the cross section(s) used in the building. This is done with the `el_cross_section` object. It needs the cross sectional area, `A`, and the second moment of area, `I`. For example:

```
% W24x68 cross section
crossW24x68 = el_cross_section;
crossW24x68.A = 0.01;                   % [m^2]
crossW24x68.I = 0.001;                  % [m^4]
```

3. Define the material(s) for use in the model. Similarly with the cross sections this is done with the `el_material` object. This object requires the Young's modulus, `E`, the yield force, `Fy`, and the density, `rho`. It can optionally be given a name by storing a string in the name variable, `name`. For example:

```
% W14x257 material
matW14x257 = el_material;
matW14x257.name = 'W14x257';
matW14x257.E = 5e9;             % [Pa]
```

```
matW14x257.Fy = 1e4;                    % [N]
matW14x257.rho = 5e3;          % [N/m]
```

4. The mesh generation script can now be run to create the mesh variables. This script takes inputs of: the number of floors, number of bays, elements per column, bay width, floor height, column material, beam material(s), column cross section, beam cross section(s), floor load(s). The beam materials and cross sections can be either single objects or a vector of objects equal in length to the number of floors. If a vector is given each object is used for each floor starting at the 1st. Similarly the floor loads can be a scalar applied to all floors or a vector so that a different load can be applied to each floor. The outputs are a vector of elements, `els`, a matrix of nodes, x, a vector and matrix of boundary conditions, `bcn` and `bcv`, a vector `load_nodes` and a corresponding vector of load forces, `load_force`. This is best illustrated in an example:

```
% set up beam vectors
mat_beam = [matW33x118 matW30x116 matW24x68];
cross_beam = [crossW33x118 crossW30x116 crossW24x68];

% run mesh generator
[els, x, bcn, bcv, load_nodes, load_force] = mesh_gen(...
   floors,bays,elms_per_col, bay_width, floor_height, ...
   matW14x257, mat_beam, crossW14x257, cross_beam, ...
   floor_load);
```

5. Optionally this generated mesh can be verified visually by creating a plot using the plot script, `plot_mesh`. If a name was given to any materials this will show up in the plot. For example:

```
plot_mesh( els, x,bcn,bcv );
```

The alternative is to create the element vector, node list and boundary conditions manually. This is useful for very simple examples using only a few elements, the steps are outlined here:

1. Create a vector of nodes where the first column consists of x coordinates and the second column has the corresponding y coordinates. For example, the single element sticking out of the ground:

```
% nodelist
x = [0 0;                                   % ground node
       0 1];                                % 1st floor node
```

2. As when using the meshing script, create the cross section and material objects. The single element example is continued here:

```
% define the material
mat1 = el_material;
mat1.E = 1E9;
mat1.rho = 5e4;

% define the cross section
cross1 = el_cross_section;
cross1.A = .5;
cross1.I = .01;
```

3. Create a vector of each node and a corresponding matrix of boundary conditions. The matrix should contain the boundary conditions for each node in the order of the vector. The first column is the type of condition for the first degree of freedom and the corresponding value, then the second degree of freedom in the third column and so forth. For the single element example:

```
% boundary conditions
bcn = [1 2]';
bcv = [1 0 1 0 1 0;
       2 1 2 0 2 0];
```

4. The element(s) can now be defined. If more than one element is required then create a vector of element objects. The element object requires some inputs on creation. These inputs are the material, cross section, a vector of the nodes this element connects and the node list matrix. In this case an element "vector" in MATLAB is created for the single element case:

```
els(1) = element(mat1,cross1,[1 2],x);
```

At this point both methods have almost all the data needed to being the analysis with either the Newmark-β or ODE algorithm. The final pieces of data to be defined are the input acceleration vector, a corresponding time step between each of the data values and the Ramberg-Osgood constant. These can be defined as follows:

```
% earthquake data
load eq_R
f = accel;
dt_data = 0.02;

% ramberg-osgood constant
r = 5;
```

A.2 Solving with Newmark-β

The first step is to enable hysteresis on each element. This is done by creating the Ramberg-Osgood hysteresis object inside each element object. On creation this object requires the Ramberg-Osgood constant, `r`. For example:

```
for a = 1:length(els);
       els(a).hyst = hyst_rambergosgood(r);
end
```

Now the solver time step needs to be defined and the solver can be run:

```
dt_solver = 1e-5;
[D,T] = const_accel_newmark_hyst(els, size(x,1)*3, ...
                      f, dt_data, dt_solver, bcn, bcv, ...
                    @(M,K) caughey(M,K,zeta,bcn,bcv), ...
                      load_nodes, load_force);
```

The inputs to the `const_accel_newmark_hyst` function are; `els`, a vector of element objects; `n`, the number of degrees of freedom in the system; `f`, a vector of acceleration data; `dt_data`, the time step in the acceleration data; `dt_solver`, the time step that the Newmark-β solution should use; `bcn` and `bcv`, the vector and matrix defining the boundary conditions as defined earlier; `damping_fn(M,K)`, a function that takes arguments of `M` and `K`, the mass and stiffness matrices, and returns the damping matrix; `load_nodes` and `load_force`, vectors of the nodes to have load forces applied to and the corresponding forces as produced by the meshing script discussed earlier.

The outputs consist of a matrix of displacements, `D`, for each degree of freedom for each time step and a corresponding time vector, `T`. It is important to note that during the course of the `const_accel_newmark_hyst` function the element objects will have stored data and changed their local tangent stiffness. These elements must therefore be redefined before any successive runs.

A.3 Solving with the ODE algorithm

Instead of enabling hysteresis on an element by element basis the `ode_hyst` function takes the Ramberg-Osgood constant as an argument so it can be called immediately:

```
    [D, T] = ode_hyst( size(x,1)*3, els, f, dt_data, bcn, ...
                bcv, @(M,K) caughey(M,K,zeta,bcn,bcv), Fy, ...
                r, load_nodes, load_force);
```

The inputs for the `ode_hyst` function are similar to the `const_accel_newmark_hyst` function with only two notable exceptions. The yield force, `Fy`, is optional. If it is given a value this value will be used for all elements, if it is set to zero the yield force will be found on a per element basis as with the `const_accel_newmark_hyst` function. Secondly, the Ramberg-Osgood constant, `r`, is given globally here.

A.4 Post processing

The displacements for any degree of freedom are easily plotted from the displacement matrices returned. To produce the benchmark results as in Figures 4.1 to 4.13 a function called `resultPlot` is used. The function requires a vector of solver time step values used in the Newmark function, a corresponding vector of errors, a corresponding vectors of time taken for those solutions and a scalar of the time taken for the ODE solution. For example:

```
    resultPlot( dt_solver, errors, NewmarkTime, OdeTime )
```

This function returns nothing. It creates the plot on the current active figure.

APPENDIX B: TEST CASE CODE

B.1 SINGLE ELEMENT

```
% Single element sticking up out of the ground
x = [0 0;      % ground node
     0 3];     % 1st floor node

% Hysteresis data
r = 5;
Fy = 1e4;

% acceleration data
% f = -(0:0.1:2);
% f = [f -f zeros(1,length(f)*3) f];
load eq_R.mat
f = -accel';

% Time step data
dt_data = 0.02;
dt_solver = logspace(-2,-4,20);

% damping coefficients
zeta = 0.05*ones(1,size(x,1)*3);

% define the material
mat1 = el_material;
mat1.E = 1E9;
mat1.Fy = Fy;
mat1.rho = 5e4/9.81;

% define the cross sections
cross_col = el_cross_section;
cross_col.A = .5;
cross_col.I = .01;

% define elements
els(1) = element(mat1,cross_col,[1 2],x);

bcn = [1 2]';
bcv = [1 0 1 0 1 0;
       2 1 2 0 2 0];

%% Ode run
disp('Starting secant ODE solver');tic
[ D2, T] = ode_hyst( size(x,1)*3, els, f, dt_data,...
    bcn, bcv,  @(M,K) caughey(M,K,zeta,bcn,bcv), Fy, ...
      r, 0, 0, 'ode113' );
OdeTime=toc;

%% Newmark runs
errors = dt_solver*0; NewmarkTime = errors;

for idt = 1:length(dt_solver),
    % define elements
    els(1) = element(mat1,cross_col,[1 2],x);
    % Use rambergosgood hysteresis
    for a = 1:length(els),
        els(a).hyst = hyst_rambergosgood(r);
    end

    disp('Start Newmark solver');tic
    [D1, T1] = newmark_hyst(els,size(x,1)*3,f, ...
              dt_data,dt_solver(idt),bcn,bcv, ...
                     @(M,K) caughey(M,K, zeta, bcn, bcv ));
    timea = toc;

    % calculate error
    errors(idt) = errorCalc(T1,D1(4,:),T,D2(:,1))*100;
    NewmarkTime(idt) = timea;
end

%% plot the result
figure(1);clf;
resultPlot( dt_solver,errors,NewmarkTime,OdeTime )
```

B.2 Two Elements

```
% Two elements sticking up out of the ground
x = [0 0;      % ground node
     0 3;      % 1st floor node
     0 6];     % 2nd floor node

% Hysteresis data
r = 5;
Fy = 1e4;

% acceleration data
% f = -(0:0.1:2);
% f = [f -f zeros(1,length(f)*3) f];
load eq_R.mat
f = -accel';

% Time step data
dt_data = 0.02;
dt_solver = logspace(-2,-4,20);

% damping coefficients
zeta = 0.05*ones(1,size(x,1)*3);

% define the material
mat1 = el_material;
mat1.E = 1E9;
mat1.Fy = Fy;
mat1.rho = 5e4/9.81;

% define the cross sections
cross_col = el_cross_section;
cross_col.A = .5;
cross_col.I = .01;

% define elements
els(1) = element(mat1,cross_col,[1 2],x);
els(2) = element(mat1,cross_col,[2 3],x);

bcn = [1 2 3]';
bcv = [1 0 1 0 1 0;
       2 1 2 0 2 0;
       2 1 2 0 2 0];

%% Ode run
disp('Starting secant ODE solver');tic
[ D2, T, V, F ] = ode_hyst( size(x,1)*3, els, f, dt_data,...
    bcn, bcv,  @(M,K) caughey(M,K,zeta,bcn,bcv), Fy, r, 0, 0, 'ode113' );
OdeTime=toc;

%% Newmark runs
errors = dt_solver*0; NewmarkTime = errors;

for idt = 1:length(dt_solver),

    % (re)define elements
    els(1) = element(mat1,cross_col,[1 2],x);
    els(2) = element(mat1,cross_col,[2 3],x);
```

```
      % Use rambergosgood hysteresis
     for a = 1:length(els),
         els(a).hyst = hyst_rambergosgood(r);
     end

     disp('Starting newmark solver’);tic
     [D1, T1] = const_accel_newmark_hyst(els, ...
                      size(x,1)*3,f, dt_data, ...
                      dt_solver(idt), bcn, bcv, ...
                      @(M,K) caughey(M,K, zeta, bcn, bcv ));
     timea = toc;

     % calculate error
     errors(idt) = max([errorCalc(T1,D1(4,:),T,D2(:,1)) ...
                             errorCalc(T1,D1(7,:),T,D2(:,4))])*100;
     NewmarkTime(idt) = timea;
     fprintf('Error for this run: %.3f%%\n',errors(idt));
 end

 %% plot the result
 figure(1);
 resultPlot( dt_solver,errors,NewmarkTime,OdeTime )
```

B.3 Single Floor

```
% Single floor
x = [0 0; 10 0;    % ground nodes
     0 3; 10 3]; % 1st floor nodes

% Hysteresis data
r = 5;
Fy = 1e4;

% acceleration data
% f = -(0:0.1:2);
% f = [f -f zeros(1,length(f)*3) f];
load eq_R.mat
f = -accel';

% Time step data
dt_data = 0.02;
dt_solver = logspace(-2,-4,20);

% damping coefficients
zeta = 0.05*ones(1,size(x,1)*3);

% define the material
mat1 = el_material;
mat1.E = 1E9;
mat1.Fy = Fy;
mat1.rho = 5e4/9.81;

% define the cross sections
cross_col = el_cross_section;
cross_col.A = .5;
cross_col.I = .01;

bcn = [1 2 3 4]';
bcv = [1 0 1 0 1 0;
       1 0 1 0 1 0;
       2 1 2 0 2 0;
       2 1 2 0 2 0];

% define elements
els(1) = element(mat1,cross_col,[1 3],x);
els(2) = element(mat1,cross_col,[2 4],x);
   els(3) = element(mat1,cross_col,[3 4],x);

%% Ode run
disp('Starting secant ODE solver');tic
[ D2, T, V, F ] = ode_hyst( size(x,1)*3, els, f, dt_data,...
    bcn, bcv,  @(M,K) caughey(M,K,zeta,bcn,bcv), 0, r, 0, 0, 'ode113' );
OdeTime=toc;

%% Newmark runs
errors = zeros(length(dt_solver),1); NewmarkTime = errors;

for idt = 1:length(dt_solver),

    % redefine elements
    els(1) = element(mat1,cross_col,[1 3],x);
    els(2) = element(mat1,cross_col,[2 4],x);
    els(3) = element(mat1,cross_col,[3 4],x);
    % Use rambergosgood hysteresis
    for a = 1:length(els),
        els(a).hyst = hyst_rambergosgood(r);
    end

    disp('Starting newmark solver’);tic
    [D1, T1] = const_accel_newmark_hyst(els,size(x,1)*3, ...
                     f, dt_data, dt_solver(idt), bcn, bcv, ...
                     @(M,K) caughey(M,K, zeta, bcn, bcv ));
    timea = toc;

    % calculate error
    errors(idt) = max([errorCalc(T1,D1(7,:),T,D2(:,1)) ...
                            errorCalc(T1,D1(10,:),T,D2(:,4))])*100;
    NewmarkTime(idt) = timea;
    fprintf('Error for this run: %.3f%%\n',errors(idt));
end

%% plot the result
figure(1);clf;
resultPlot( dt_solver,errors,NewmarkTime,OdeTime )
```

B.4 SAC-3 Building Model

```
% acceleration data
% f = -(0:0.1:2);
% f = [f -f zeros(1,length(f)*3) f];
load eq_R.mat
f = -accel';

% Time step data
dt_data = 0.02;
dt_solver = logspace(-3,-5,20);

% hysteresis variables
r = 5;

% mesh variables
floors = 3;
bays = 3;
elms_per_col = 1;
bay_width = 9.144;                      % 30 feet in meters
floor_height = 3.9624;                  % 13 feet in meters
floor_load = [4706e3; 4706e3; 5094e3];  % loading to be applied [kN]

%-----------------------------------------------------------------------------
%%                           Set up cross-sections
%-----------------------------------------------------------------------------
% W24x68
crossW24x68 = el_cross_section;
crossW24x68.A = 0.012967716;        % 20.1 in^2 in m^2 (from [2])
crossW24x68.I = 0.000761703509;     % 1830 in^4 in m^4 (from [2])

% W30x116
crossW30x116 = el_cross_section;
crossW30x116.A = 0.022064472;       % 34.2 in^2 in m^2 (from [2])
crossW30x116.I = 0.00205202093;     % 4930 in^4 in m^4 (from [2])

% W14x257
crossW14x257 = el_cross_section;
crossW14x257.A = 0.048774096;       % 75.6 in^2 in m^2 (from [2])
crossW14x257.I = 0.00141518685;     % 3400 in^4 in m^4 (from [2])

% W33x118
crossW33x118 = el_cross_section;
crossW33x118.A = 0.022387052;       % 34.7 in^2 in m^2 (from [2])
crossW33x118.I = 0.00245576541;     % 5900 in^4 in m^4 (from [2])


%-----------------------------------------------------------------------------
%%                            Set up materials
%-----------------------------------------------------------------------------
E  = 34473785000;           % 5000ksi in Pa (from [1])
Sigma_y = 1/10*317158822e-2;     % 46ksi in Pa (from [1]) (reduced by 10^2
for now)
g = 9.81;

% W14x257
matW14x257 = el_material;
matW14x257.name = 'W14x257';
```

```
matW14x257.E = E;
matW14x257.Fy = Sigma_y*crossW14x257.A;
matW14x257.rho = 382.458133*g;      % 257 lb/ft in N/m (from [2])

% W24x68
matW24x68 = el_material;
matW24x68.name = 'W24x68';
matW24x68.E = E;
matW24x68.Fy = Sigma_y*crossW24x68.A;
matW24x68.rho = 101.195148*g;       % 68 lb/ft in N/m (from [2])

% W30x116
matW30x116 = el_material;
matW30x116.name = 'W30x116';
matW30x116.E = E;
matW30x116.Fy = Sigma_y*crossW30x116.A;
matW30x116.rho = 172.627017*g;      % 116 lb/ft in N/m (from [2])

% W33x118
matW33x118 = el_material;
matW33x118.name = 'W33x118';
matW33x118.E = E;
matW33x118.Fy = Sigma_y*crossW33x118.A;
matW33x118.rho = 175.603345*g;      % 118 lb/ft in N/m (from [2])

%--------------------------------------------------------------------------
%%                            Generate Mesh
%--------------------------------------------------------------------------
% set up beam vectors
mat_beam = [matW33x118 matW30x116 matW24x68];
cross_beam = [crossW33x118 crossW30x116 crossW24x68];
% mat_beam = matW14x257;
% cross_beam = crossW14x257;

% run mesh generator
[els, x, bcn, bcv, load_nodes, load_force] = mesh_gen(...
    floors,bays,elms_per_col, bay_width, floor_height, matW14x257, ...
    mat_beam, crossW14x257, cross_beam, floor_load);

% confirm the mesh is correct using a plot
figure(1);plot_mesh( els, x,bcn,bcv );%title('SAC-3 Mesh');

% Damping vector:
zeta = repmat(0.05,size(x,1)*3,1);


%--------------------------------------------------------------------------
%%                               ODE Solve
%--------------------------------------------------------------------------

% redefine elements (old ones have hysteresis degredation)
[els, nodelist, bcn, bcv, load_nodes, load_force] = mesh_gen(...
    floors,bays,elms_per_col, bay_width, floor_height, matW14x257, ...
    mat_beam, crossW14x257, cross_beam, floor_load);

disp('Starting secant ODE solver');tic
[ D, T, V, F ] = ode_hyst( size(x,1)*3, els, f, dt_data,...
    bcn, bcv,  @(M,K) caughey(M,K,zeta,bcn,bcv), 0, r, ...
    load_nodes, load_force,'ode113');
```

```
OdeTime=toc;

%-----------------------------------------------------------------------
%%                              Solve with Newmark
%-----------------------------------------------------------------------
errors = zeros(length(dt_solver),1);NewmarkTime=errors;
for idt=1:length(dt_solver)
    % redefine elements (old ones have hysteresis degredation)
    [els, nodelist, bcn, bcv, load_nodes, load_force] = mesh_gen(...
        floors,bays,elms_per_col, bay_width, floor_height, matW14x257, ...
        mat_beam, crossW14x257, cross_beam, floor_load);

    % Enable Ramberg-Osgood hysteresis on all elements
    % coefficient (r) defined at the top of this script
    for a = 1:length(els);
        % Use rambergosgood
        els(a).hyst = hyst_rambergosgood(r);
    end
    disp('Solving with Newmark');tic;
    [D1,T1] = const_accel_newmark_hyst(els, size(x,1)*3, f, ...
                dt_data, dt_solver(idt), bcn, bcv, ...
                @(M,K) caughey(M,K,zeta,bcn,bcv), load_nodes, load_force);
    timea = toc;
    errors(idt) = max([errorCalc(T,D(:,end-2),T1,D1(end-2,:)) ...
                   errorCalc(T,D(:,1),T1,D1(4,:))])*100;
    NewmarkTime(idt) = timea;
    fprintf('Error for this run: %.3f%%\n',errors(idt));
end

%-----------------------------------------------------------------------
%%                              Display the results
%-----------------------------------------------------------------------
figure(1);clf;
resultPlot( dt_solver,errors,NewmarkTime,OdeTime )

% References:
% [1] - Seismic Energy Dissipation of Buildings Engineered Cladding Systems
%           by Quan Viet Nguyen
% [2] -
% http://www.engineeringtoolbox.com/american-wide-flange-steel-beams-
d_1319.html
```