NUMERICAL EVALUATOR OF DYNAMIC RESPONSES OF SINGLE/MULTIPLE DEGREE(S) OF FREEDOM STRUCTURES

BY: KEVIN YONGCHUL KIM

REPORT

Submitted in partial fulfillment of the requirements

For the Degree of Masters of Science in Civil and Environmental Engineering

University of California, Davis

Davis, CA

FACULTY ADVISOR: DR. KEN LOH

MS Plan II Committee: Dr. John E. Bolander

MS Plan II Committee: Dr. Rob Y. H. Chai

Contents

Abstract

The report outlines the procedure of two sets of MATLAB based programs written that takes in user inputs of structural properties, (i.e. mass, stiffness, damping ratio) subjects the structure to external forces, (i.e. user-defined functions, ground motion data) and outputs displacements of the given structure over a specified time range by means of numerical analysis.

The first set of programs evaluates a single degree of freedom (SDOF) structure's dynamic response, while the second set of programs evaluates a multiple degree of freedom (MDOF) structure's dynamic response.

Motivation

The primary purpose of these programs is to serve as a complementary learning module for students in graduate or undergraduate upper-division courses in structural dynamics. For example, students will be able to see how increasing or decreasing mass, stiffness, and damping of a SDOF or MDOF structure affects its response. The programs are not limited to free vibration responses as it also allows for user-inputs of forcing functions; therefore, students can observe the effects that different forcing functions (i.e. step-force, impulse loads, harmonic loads) have on an assigned structure.

The programs are also allowed to handle ground motion data to output dynamic responses. In turn, students can input ground motion data of historic earthquakes and observe how a user-defined model may have responded under the given earthquake.

At its most basic, these programs serve to educate students on an input-output basis much like that of a black box; however, students interested in programming or numerical analysis of dynamic response are encouraged to explore the various subroutines included in the program, and this report will provide a general summary of those subroutines.

It is assumed throughout the report that the reader has a basic understanding of matrix algebra and structural dynamics—and therefore should refer to textbooks or outside sources should the brief summaries in theory prove to be insufficient.

1. Single Degree of Freedom (SDOF) Modeling and Response

1.1 Background

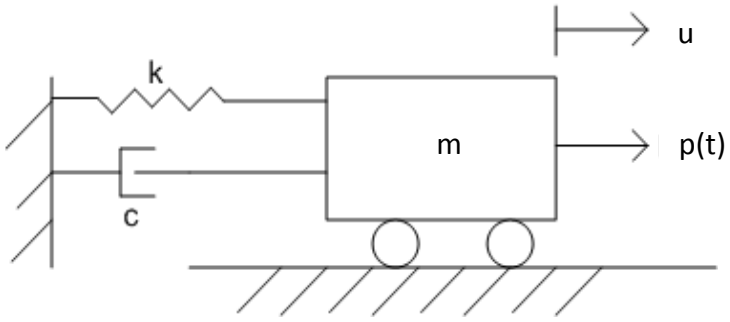

Figure 1: A SDOF structure represented as classic mass-spring-damper model

A single degree of freedom structure's general equation of motion can be represented as a mass-spring-damper model as seen in Figure 1, and its equation of motion can be written as a second order differential equation as follows:

$$m\ddot{u}(t) + c\dot{u}(t) + ku(t) = p(t) \quad (1)$$

Where m is the mass of the model, c is the viscous damping coefficient, k is the stiffness of the spring, u(t), u̇(t), ü(t) represent displacement, velocity, and acceleration at a given time t respectively, while p(t) represents the external force applied to the system at a given time t. For convenience, the dependence

on t will be assumed from herein, and the terms u̇ and ü will be interchangeable with terms *v* and *a* respectively.

As the program uses numerical integration to calculate the dynamic responses, Equation (1) is modified as follows:

$$m\ddot{u}_i + c\dot{u}_i + ku_i = p_i \tag{2}$$

where the subscript *i* indicate the value of the variable associated with it at a given time t_i. The indices t_i and t_{i+1} are separated out by a time step Δt (or dt).

1.2 User_Input.m

The set of programs starts off with the user opening up a file titled User_Input.m via MATLAB. The user can then define the mass, stiffness and damping ratio ξ (xi) of the structure, impose initial displacement and initial velocity of the given structure, and select the time increment and time range for which the system's response will be numerically integrated to. The user then selects from one of three numerical integration techniques coded into the program, which are the Central Difference Method, Newmark's Method, and the Runge-Kutta Method, all of which will be explained further in detail in the following sections.

1.2.1 Variable: Ground_motion

Lastly, the user must identify what type of forces the modelled structure will undergo; this is categorized into two broad categories. They must enter either 1 for actual ground motion, or 2 for well-defined forcing function. The motivation for this is that each option will output a different structure and different animation schemes to better illustrate the dynamic responses to students/users of this program. As the report goes along, the ramifications this option has on the output of the program will be made more evident. See Figure 2 below for a sample screenshot of a user input.

```
%% User_input.m--attain output by running Project_Begin.m

%% structrual properties
m=0.2533; %kip-sec^2/in %mass of SDOF structure
k=10; %kips/in %stiffness of SDOF structure
xi=0.05; %damping ratio

%% initial conditions
u0=0; %initial displacement (in)
v0=0; %initial velocity (in/s)

%% numerical integration
dt=.01; %time increment (s)
tr=35; %time range (s)

Analysis_Type='RungeKutta'; %Enter 'CDM','Newmark',or 'RungeKutta'
NM_Type='N/A'; %Enter 'average','linear', or'N/A' for non-Newmark analysis

Ground_Motion=1; %Enter 1 if feeding recorded ground motion into prog.
%ForcingFunction;%Enter 2 if inputting theoretical forcing functions.

%% Head into ForcingFunction.m to assign a forcing fcn. or ground motion
```

Figure 2: All variables regarding the structure that the user can define are available at User_Input.m

1.3 ForcingFunction.m

After inputting structural properties and initial conditions into User_Input.m, the user will then create or select a forcing function that the SDOF structure will undergo.

Suppose the user is interested in seeing how the defined structure responds under a smooth sinusoidal forcing function that follows the formula,

$$p(t) = p_0 \sin(w_f t) \tag{3}$$

for when $t \leq tr$, and $p(t)=0$ when $t>tr$. Where p_0 is the amplitude of the forcing function, w_f is the forcing function frequency, and tr the time range of the forcing function.

```
%% SAMPLE FORCING FUNCTION INPUT #1: Subject user-created structure to a
%hypothetical sinusoidal forcing function

%% User Input
ptr=0.6; %forcing function time range [s]
po=10; %forcing function amplitude [kips]
wf=pi/0.6; %forcing function frequency [rad/s]
%% END User Input

p=zeros(1,length(t)); %initialize p vector,
for i=1:length(t)
    if t(i)<=ptr %during the interval the forcing function is applied...
        p(i)=po*sin(wf*t(i)); % the forcing function is this...
        % ^^same forcing function from Chopra example 5.1 (206)
    else
        p(i)=0; %outside of the applied force range, the force=zero.
    end
end
```

Figure 3: A sinusoidal forcing function written in MATLAB

All the user would have to do in this case is to enter the forcing function time range, its amplitude, and its frequency, and then a p vector at time step Δt will be generated. If the student would like to incorporate different types of forcing functions such as ramp-force or a step force, they must ideally have a working knowledge of the way MATLAB works with vectors and for-loops in order to modify the code above.

However, suppose the student would like to see how the structure would respond under an actual recorded earthquake. Then the user should save the ground motion data into the same directory as the program (for demonstration, ground motions from the 1940 El Centro earthquake is already saved into the directory as **El_Centro_Ground_Motion.m**) and input the name of the file as exemplified in the following figure below.

```
%% SAMPLE FORCING FUNCTION INPUT #2: Subject user-created structure to the
% %El Centro Earthquake

%%BEGIN User Input
run('El_Centro_Ground_Motion') %feed Data downloaded from NCEES website.
%%END User Input

b=a'; %transpose the ground history data...
zz=b(:); %so that the data turns into neat column vector;
zz(length(zz))=[]; %omit last value in data;0 added to square the matrix
acc=[0;zz]; %@ t=0, acceleration is 0, motion data starts at t=0.02; add 0
el_t=0:0.02:31.18; %time vector goes from 0 to 31.18s with time step .02s;
x=el_t'; %El Centro Time vector converted to column vector
y=acc; %El Centro acc vector is already a column vector.
xi=t'; %convert user input time vector into column vector
yi=interp1q(x,y,xi); %linearly interpol. to attain acc pts spaced dt apart.
yi(isnan(yi))=0; %set values after final data in ground motion acc=0.
g=32.2*12; %gravity in inches/s^2
ag=yi*g; %ground motion acc. a(t);
p=-m*ag; %external force due to ground motion in SDOF system
p=p'; %revert p vector into row vector to match the code consistency.
```

Figure 4: Ground Motion data is converted into a force p vector.

If output is only of interest to the user, then he or she needs to simply type the name of the file into the run command.

The bulk of code following the user input, creates a ground acceleration vector a_g *(or* $\ddot{u_g}$*)* spaced Δt by interpolating between the ground motion data, then creates a *p* vector by multiplying the acceleration vector by (–)*m.* Recall that the equation of motion for an SDOF system subjected to ground motion can be written as the following.

$$m\ddot{u}_i + c\dot{u}_i + ku_i = -m\ddot{u}_{gi} \tag{4}$$

1.4 Project_Begin.m

After assigning structural properties and analysis method under **User_Input.m**, then creating a force vector *p* under **ForcingFunction.m**, the user can then run the file **Project_Begin.m** in the directory to output plots regarding displacement vs. time, structural response animation, and ground motion if applicable.

Project_Begin.m starts off by first creating a time vector *t* that starts from 0, increases by a time step dt all the way to the time range *tr* that the user has input under **User_Input.m**. It also assigns the following variables as such:

$$\omega_n = \sqrt{\frac{k}{m}} \tag{5}$$

$$c = 2m\omega_n\xi \tag{6}$$

$$T_n = \frac{2\pi}{\omega_n} \tag{7}$$

where ω_n is the natural frequency of the structure in radians per second, *c* is the damping coefficient from Figure 1 (in kips*s/in), and T_n is the natural period in seconds.

Depending on what the user selected as the analysis type and whether or not ground motion will be fed into the **ForcingFunction.m**—**Project_Begin.m** will run one of the six scripts written into the directory listed as below:

1. **Central_Difference_Method.m**
2. **Newmark_Method.m**
3. **Runge_Kutta_Method.m**
4. **Central_Difference_Method_Ground_Motion.m**
5. **Newmark_Method_Ground_Motion.m**
6. **Runge_Kutta_Method_Ground_Motion.m**

The first three scripts will output a spring model animation while the latter three scripts will output an animation of a model that resembles a water tower undergoing earthquake motion. The differences between these two sets of output will be made more evident as the report goes along. The process in how displacement data is calculated is identical between files 1 and 4, files 2 and 5, and files 3 and 6.

1.5 Central Difference Method (CDM)

1.5.1 Overview

The Central Difference Method is based on a finite difference approximation of the time derivatives of displacement such as velocity and acceleration. (Chopra 207). Taking constant time step dt, the expressions for velocity and acceleration at time *i* are:

$$\dot{u}_i = \frac{u_{i+1} - u_{i-1}}{2\Delta t} \tag{1.5.1.1}$$

$$\ddot{u}_i = \frac{u_{i+1} - 2u_i + u_{i-1}}{(\Delta t)^2} \tag{1.5.1.2}$$

By putting these two variables into Equation (2), and manipulating the equation around, the displacement at the next time step can be expressed as:

$$u_{i+1} = \frac{\hat{p}_i}{\hat{k}} \tag{1.5.1.3}$$

Where

$$\hat{p}_i = p_i - \left[\frac{m}{(\Delta t)^2} - \frac{c}{2\Delta t}\right] u_{i-1} - \left[k - \frac{2m}{(\Delta t)^2}\right] u_i \tag{1.5.1.4}$$

And

$$\hat{k} = \frac{m}{(\Delta t)^2} + \frac{c}{2\Delta t} \tag{1.5.1.5}$$

Refer to either the Chopra text of Section 5.3 or a structural dynamics text for the full derivation.

1.5.2 Programming Iteration on MATLAB

Once $\hat{k}$ and $\hat{p}_i$ are defined, a simple for-loop in MATLAB will calculate all the displacements of u_i at time step t_i shown in the following:

```
u(1)=u0; %replace the first u entry as user input initial displacement
a= (m/dt^2)-(c/(2*dt));
b= k-((2*m)/(dt)^2);
for i=2:length(t)
    phat(i)=p(i)-a*u(i-1)-b*u(i); %Equation 1.5.1.4
    u(i+1)=phat(i)/kh;            %Equation 1.5.1.3
end
```

Figure 5: CDM can be written in MATLAB as an elegant for-loop

1.6 Newmark's Method

1.6.1 Overview

Newmarks' method is a family of time-stepping methods that is entirely focused on acceleration at a given time step. In short, the method is still based off the equilibrium equation (2), but all the iterative steps hinge on attaining the values of acceleration at a given point for the iterations to continue. For the full derivation and explanation of Newmark's Method, refer to Chopra's Section 5.4 or any dynamics textbook, as the report will focus more on the iterative steps.

1.6.2 Iteration Overview

The iteration first starts off with the user selecting either '*average*' or '*linear*' under the **User_Input.m** file which determines the coefficients of γ, and β.

(1) Average acceleration method ($\gamma = \frac{1}{2}, \beta = \frac{1}{4}$)

(2) Linear acceleration method ($\gamma = \frac{1}{2}, \beta = \frac{1}{6}$)

Afterwards, initial calculations provide the initial acceleration and coefficients as follows.

$$\ddot{u}_0 = \frac{p_0 - c\dot{u}_0 - ku_0}{m} \tag{1.6.2.1}$$

$$\hat{k} = k + \frac{\gamma}{\beta \Delta t} c + \frac{1}{\beta (\Delta t)^2} m \tag{1.6.2.2}$$

$$a = \frac{1}{\beta \Delta t} m + \frac{\gamma}{\beta} c \tag{1.6.2.3}$$

$$b = \frac{1}{2\beta} m + \Delta t \left(\frac{\gamma}{2\beta} - 1 \right) c \tag{1.6.2.4}$$

At this point, changes in position over displacement, velocity, and acceleration are calculated per given iteration.

$$\Delta p_i = p_{i+1} - p_i \tag{1.6.2.5}$$

$$\Delta \hat{p}_i = \Delta p_i + a \dot{u}_i + b \ddot{u}_i \tag{1.6.2.6}$$

$$\Delta u_i = \frac{\Delta \hat{p}_i}{\hat{k}} \tag{1.6.2.7}$$

$$\Delta \dot{u}_i = \frac{\gamma}{\beta \Delta t} \Delta u_i - \frac{\gamma}{\beta} \dot{u}_i + \Delta t \left(1 - \frac{\gamma}{2\beta} \right) \ddot{u}_i \tag{1.6.2.8}$$

$$\Delta \ddot{u}_i = \frac{1}{\beta (\Delta t)^2} \Delta u_i - \frac{1}{\beta \Delta t} \dot{u}_i - \frac{1}{2\beta} \ddot{u}_i \tag{1.6.2.9}$$

After changes in displacement, velocity, and acceleration are found, the next step's dynamic responses are simply added from the previous steps as follows.

$$u_{i+1} = u_i + \Delta u_i \tag{1.6.2.10}$$

$$\dot{u}_{i+1} = \dot{u}_i + \Delta \dot{u}_i \tag{1.6.2.11}$$

$$\ddot{u}_{i+1} = \ddot{u}_i + \Delta \ddot{u}_i \tag{1.6.2.12}$$

1.6.3 Programming Iteration on MATLAB

After initial variables are coded into MATLAB following formulas (1.6.2.1)~(1.6.2.4), initial displacement and velocity are placed into the beginning of the position and velocity vector. The program then enters into an iterative loop as exemplified in the following figure.

```
%% Initial conditions
u(1)=u0; %replace the first u entry as initial displacement
v(1)=v0; %replace the first velocity entry as initial velocity
acc_n(1)=a0; %replace the first acceleration entry as initial acceleration
%^^where a0 is Eqn 1.6.2.1

% Iterative Steps
for i=1:length(t)-1
    dp(i)=p(i+1)-p(i); %Eqn 1.6.2.5
    dph(i)=dp(i)+a*v(i)+b*acc_n(i);%Eqn 1.6.2.6
    du(i)=dph(i)/kh; %Eqn 1.6.2.7
    dv(i)=(g/B/dt)*du(i)-(g/B)*v(i)+dt*(1-(g/2/B))*acc_n(i);%1.6.2.8
    da(i)=1/(B*dt^2)*du(i)-(1/(B*dt))*v(i)-(1/(2*B))*acc_n(i);%1.6.2.9

    u(i+1)=u(i)+du(i);%1.6.2.10
    v(i+1)=v(i)+dv(i);%1.6.2.11
    acc_n(i+1)=acc_n(i)+da(i);%1.6.2.12
end
```

Figure 6:Newmark's Method's iterative steps can be written in MATLAB as follows

1.7 Runge-Kutta Method

1.7.1 Overview

In brief, the Runge-Kutta 4th Order Method uses weighted average of four individual slopes to get a slopes for displacement and velocity to increase by for a given time step. Certain slopes call for values of forcing function a half step in between the values of entries in the *p* vector; by assuming a small time increment, these force values at half steps are assumed to be the average of the two *p* entries.

1.7.2 Iteration steps

The iteration steps are based off the definition that the slope of displacement over time, du/dt is the velocity v, and by setting the slope of velocity dv/dt, equal to acceleration in Equation 1 and isolating the acceleration variable onto one side. Then the slopes for du and dv are determined by averaging four slopes as follows. These slopes are then added onto the previous u and v to attain the next u's and v's.

$$du_{1i} = \Delta t(v_{i-1}) \tag{1.7.2.1}$$

$$d_{v1i} = \Delta t\left[\frac{p_{i-1} - cv_{i-1} - ku_{i-1}}{m}\right] \tag{1.7.2.2}$$

$$du_{2i} = \Delta t\left[v_{i-1} + \frac{dv_{1i}}{2}\right] \tag{1.7.2.3}$$

$$dv_{2i} = \frac{\Delta t}{m}\left[\frac{(p_i + p_{i-1})}{2} - c\left(v_{i-1} + \frac{dv_{1i}}{2}\right) - k\left(u_{i-1} + \frac{du_{1i}}{2}\right)\right] \tag{1.7.2.4}$$

$$du_{3i} = \Delta t\left[v_{i-1} + \frac{dv_{2i}}{2}\right] \tag{1.7.2.5}$$

$$dv_{3i} = \frac{\Delta t}{m}\left[\frac{(p_i + p_{i-1})}{2} - c\left(v_{i-1} + \frac{dv_{2i}}{2}\right) - k\left(u_{i-1} + \frac{du_{2i}}{2}\right)\right] \tag{1.7.2.6}$$

$$du_{4i} = \Delta t[v_{i-1} + dv_{3i}] \tag{1.7.2.7}$$

$$dv_{4i} = \frac{\Delta t}{m}[p_i - c(v_{i-1} + dv_{3i}) - k(u_{i-1} + du_{3i})] \tag{1.7.2.8}$$

$$du_i = \frac{[du_{1i} + du_{2i} + du_{3i} + du_{4i}]}{6} \tag{1.7.2.9}$$

$$dv_i = \frac{[dv_{1i} + dv_{2i} + dv_{3i} + dv_{4i}]}{6} \tag{1.7.2.10}$$

$$u_i = u_{i-1} + du_i \tag{1.7.2.11}$$

$$v_i = v_{i-1} + dv_i \tag{1.7.2.12}$$

1.7.3 Programming Iterations on MATLAB

As with Newmark's method, the initial conditions are set, and a for-loop is run; the resulting iteration loop on MATLAB is as follows in the figure below.

```
%% Runge-Kutta 4th Order ODE Iterations
%initial conditions
v(1)=v0; %set 1st pt. of velocity vector as initial velocity
u(1)=u0; %set 1st pt. of displacement vector as initial displacement

rkc=c/m; %normalize damping coefficient by mass.
rkk=k/m; %normalize stiffness coefficient by m

for i=2:length(t) %for every time step dt...
du1(i)=dt*v(i-1); %Eqn 1.7.2.1
dv1(i)=dt*((p(i-1))/m-rkc*v(i-1)-rkk*u(i-1)); %Eqn 1.7.2.2

du2(i)=dt*(v(i-1)+(dv1(i)/2)); %Eqn 1.7.2.3
dv2(i)=dt*((.5*(p(i)+p(i-1)))/m-rkc*(v(i-1)+dv1(i)/2)-rkk*(u(i-1)+du1(i)/2));
%^Eqn 1.7.2.4

du3(i)=dt*(v(i-1)+dv2(i)/2); %Eqn 1.7.2.5
dv3(i)=dt*((.5*(p(i)+p(i-1)))/m-rkc*(v(i-1)+dv2(i)/2)-rkk*(u(i-1)+du2(i)/2));
%^Eqn 1.7.2.6

du4(i)=dt*(v(i-1)+dv3(i)); %Eqn 1.7.2.7
dv4(i)=dt*(((p(i)))/m-rkc*(v(i-1)+dv3(i))-rkk*(u(i-1)+du3(i))); %Eqn 1.7.2.8

du(i)=(du1(i)+2*du2(i)+2*du3(i)+du4(i))/6; %Eqn 1.7.2.9
dv(i)=(dv1(i)+2*dv2(i)+2*dv3(i)+dv4(i))/6; %Eqn 1.7.2.10

 u(i)=u(i-1)+du(i); %Eqn 1.7.2.11
 v(i)=v(i-1)+dv(i); %Eqn 1.7.2.12
end
```

1.8 Plots and Animations

Once numerical analysis is applied by means of Central Difference Method, Newmark's Method, or the Runge Kutta Method, a full displacement vector **[u]** is created that matches the length of the time vector **[t]**. At this point, the program is ready for output, and a full-screen figure comprised of three subplots will appear as seen in the following figures below.

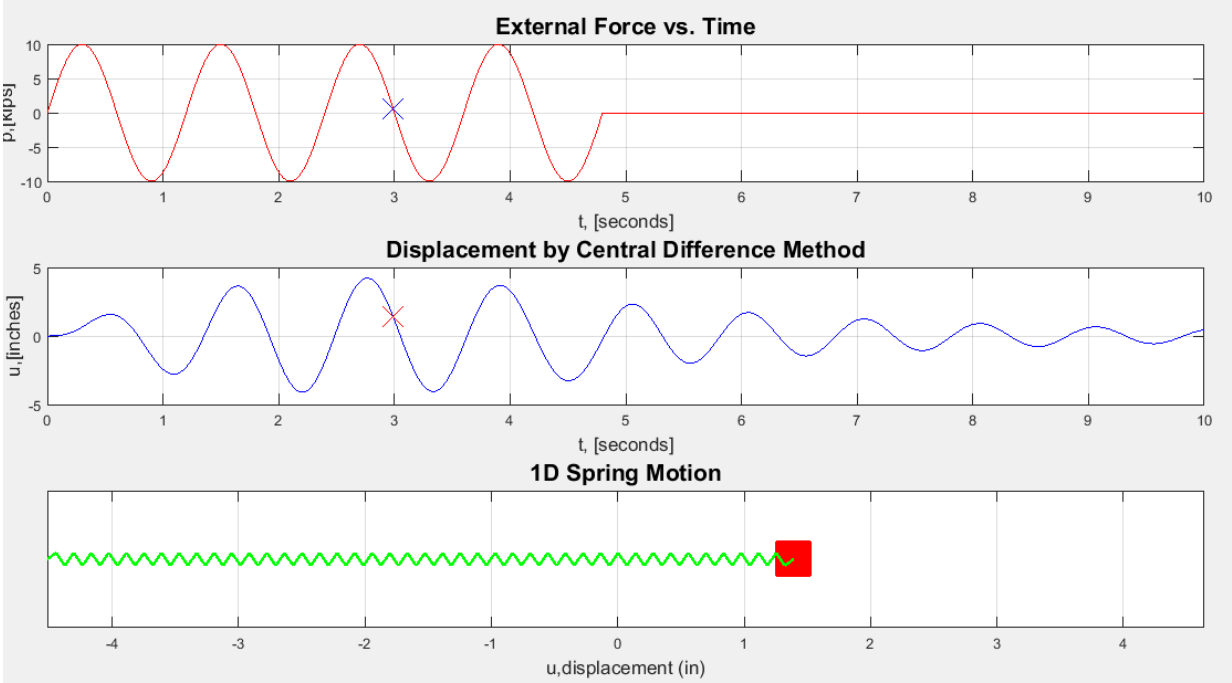

Figure 7: Set of Plots when smooth theoretical forcing function is chosen.

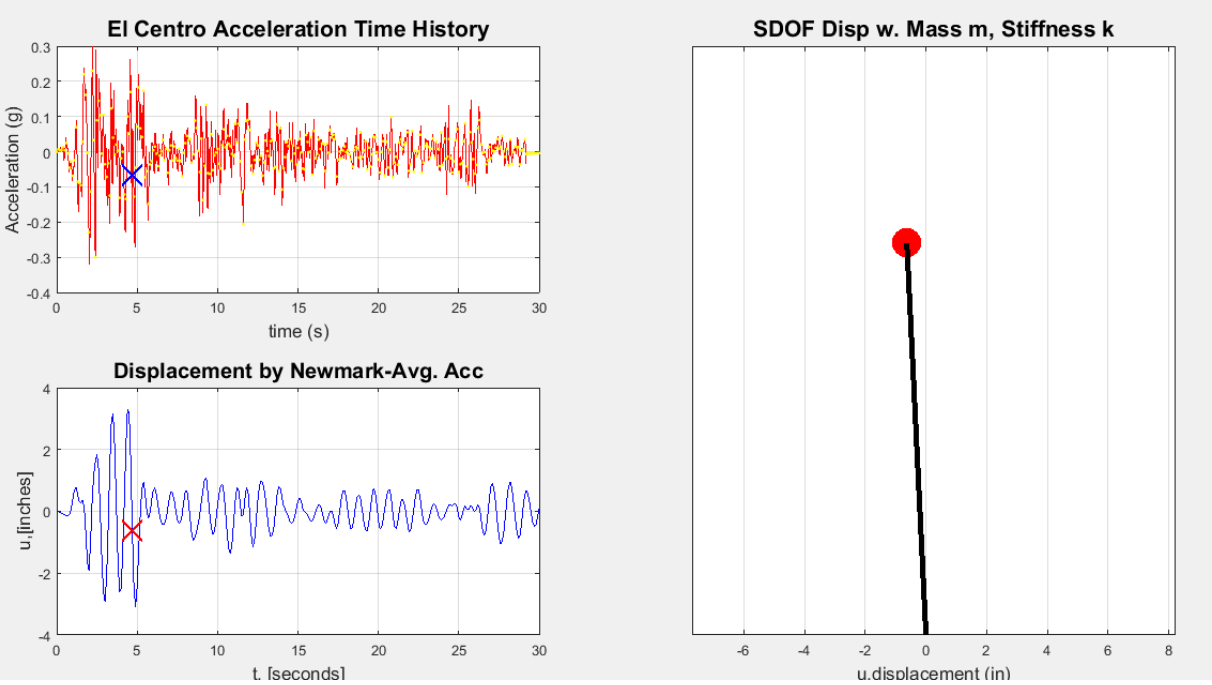

Figure 8: Set of plots when Earthquake Ground motion data is chosen.

The first subplot either plots a forcing function that the user has input, or the ground motion the user has chosen. The second subplot will plot plot **[u]** against **[t**], with the title displaying what numerical analysis method is being used. The third plot will either plot a 1D spring extending or compressing (if the user provides custom forcing function) or an arbitrary 1D structure with mass *m* undergoing displacements under earthquake motion (if the user fed ground history into program).

Note the crosshairs present in both Figures 7 and 8; these crosshairs move along their respective paths in sync with the 1D motion that is being animated in the third subplot. For example, in Figure 7, one can see that the screenshot is snapped when time is ~3 seconds, and at that moment, the spring has been displaced from its initial position ~1.4 inches to the right.

1.9 SDOF Analysis Flow Chart

Below shows a general outline of the process the folder SDOF goes through.

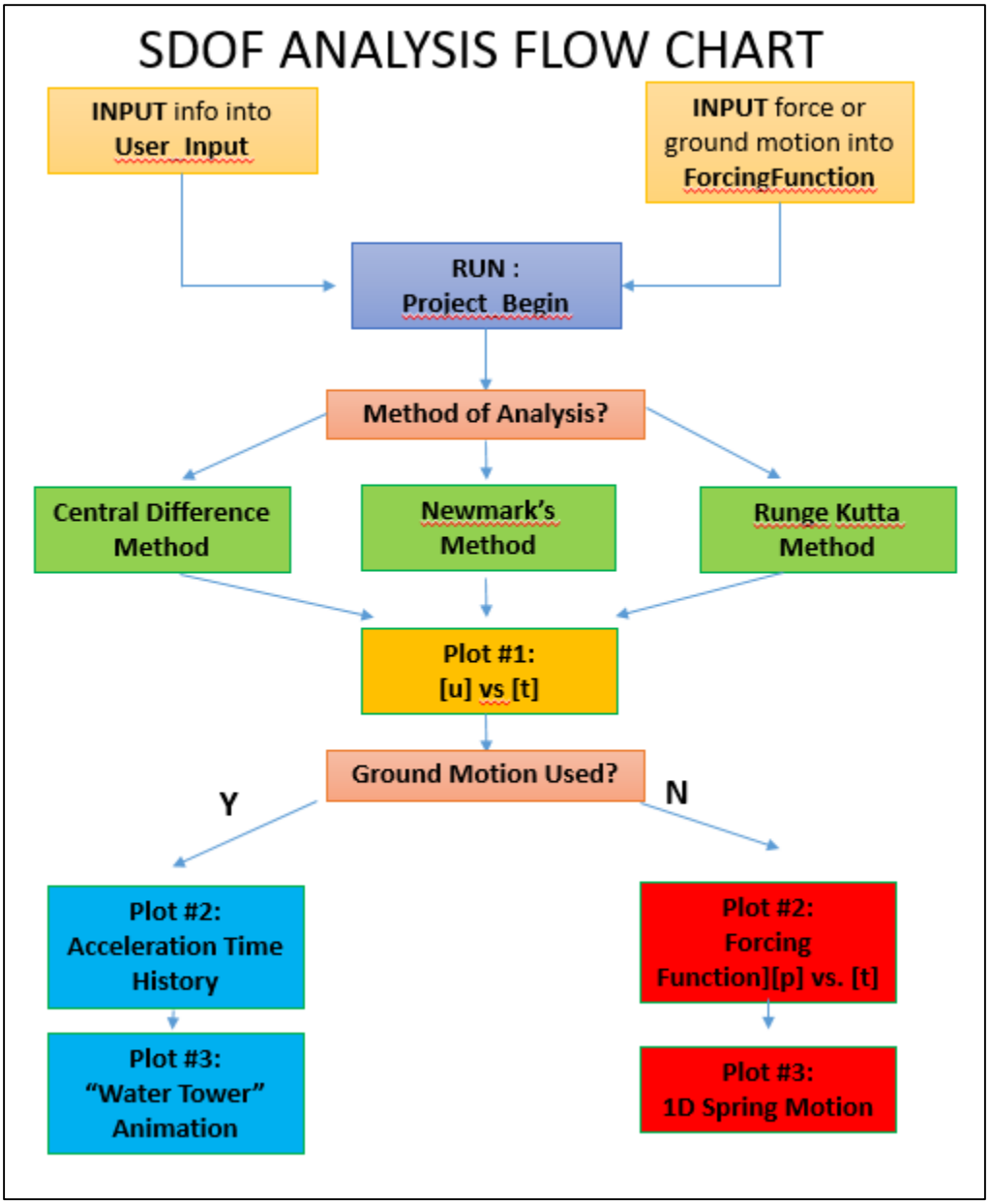

Figure 9: Flowchart of SDOF Analysis

2. Multiple Degree of Freedom (MDOF) Modeling and Response

2.1 Background/Overview

Dynamic analysis of structures with multiple degrees of freedom (MDOF) is still grounded upon the concepts based in the equations of motions of SDOF systems as stated in Section 1.1; the equation of motion for an MDOF system can be expressed as follows:

$$[\mathbf{m}]\{\ddot{\mathbf{u}}\} + [\mathbf{c}]\{\dot{\mathbf{u}}\} + [\mathbf{k}]\{\mathbf{u}\} = \{\mathbf{p}(\mathrm{t})\} or - [\boldsymbol{m}]\{\boldsymbol{i}\}\{\ddot{\boldsymbol{u}}_{\boldsymbol{g}}(\boldsymbol{t})\} \quad (2.1.1)$$

Equation 2.1.1 simply modifies Equation (1) where variables ***m, c, k*** now turn into *n*-by-*n* matrices where *n* is the number of degrees of freedom assigned to the structure. All the variables ***u, u',u" u_g"*** are now column vectors. Vector {i} is the influence vector and used in the case where ground motion data is used to analyze the dynamic responses.

As Equation 2.1.1 is written, the matrices are coupled, meaning they will create an *n* amount of equations where each equation may contain several $\boldsymbol{u}_i$ variables. As the number of degrees of freedom increases, it proves to be advantageous to transform these matrices to create uncoupled matrices.

One method of doing this—as the program presented here does—is by expressing the displacements in terms of natural vibration modes φ_n of the undamped system and multiplying them by the modal coordinates $\mathbf{q}_n(t)$ to approximate nodal displacements. In equation form, this can be expressed as (Chopra, 647).:

$$\boldsymbol{u}(\boldsymbol{t}) \cong \sum_{\boldsymbol{n=1}}^{\boldsymbol{J}} \boldsymbol{\varphi_n q_n}(\boldsymbol{t}) = \boldsymbol{\varphi q}(\boldsymbol{t}) \quad (2.1.2)$$

Where **J** can be selected from 1 to *n* to increase accuracy.

By applying transformations using the property of modal orthogonality (Chopra, Section 12.4), Equation (2.1.1) can be rewritten as:

$$[\mathbf{M}][\ddot{\mathbf{q}}] + [\mathbf{C}][\dot{\mathbf{q}}] + [\mathbf{K}][\mathbf{q}] = [\mathbf{P}(\mathrm{t})] \tag{2.1.3}$$

where

$$\mathbf{M} = \boldsymbol{\varphi}^T\mathbf{M}\boldsymbol{\varphi} \qquad \mathbf{C} = \boldsymbol{\varphi}^T\mathbf{c}\boldsymbol{\varphi} \qquad \mathbf{K} = \boldsymbol{\varphi}^T\mathbf{k}\boldsymbol{\varphi} \qquad \mathbf{P}(\mathrm{t}) = \boldsymbol{\varphi}^T\mathbf{p}(\mathrm{t}) \tag{2.1.4}$$

where **M** and **K** are uncoupled (diagonal) matrices.

2.2 User_Input_MDOF.m

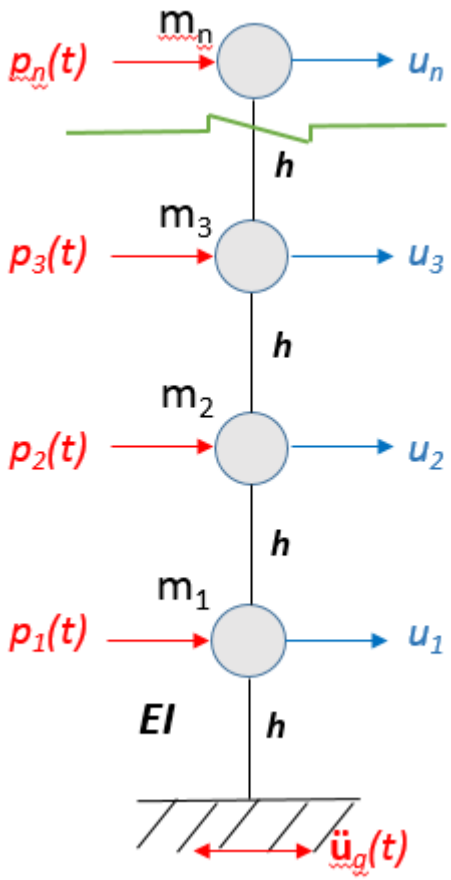

Figure 10: Chimney User can input as structural model

Under **User_Input_MDOF.m**, the user is given a reinforced-concrete chimney idealized as a lumped mass cantilever as shown in Figure 10 above. The user can modify the height of the chimney, the amount of nodes the chimney is idealized as, the stiffness of the columns, masses of each nodes, and the damping ratio of the first two modes. As was the case in the User Input file for SDOF system, the

user decides the time step increment and the time range, and must list the type of numerical analysis, and whether or not ground motion is being fed into the Forcing Function file. A screen capture of sample user-input is displayed in the following figure.

```
%% SAMPLE USER INPUT #1--RC 5-story Chimney problem Chopra Example 15.1
% EI=5.469*10^10; %kip-ft^2;
h=120; %feet; height per floor

m=208.6; %kip-sec^2/ft; mass of floor
m=[m;m;m;m;m/2]; %mass of every node

u0=[0;0;0;0;0]; %initial displacements per floor of chimney--in feet
v0=[0;0;0;0;0]; %initial velocity ft/s

xi=0.00; %damping ratio of first two modes assuming Rayleigh Damping

dt=0.0001; %enter time step increment [s]
tr=10; %enter time range [s]

Ground_Motion=2; %Enter 1 if feeding ground motion into Forcing Function
                 %Enter 2 if inputting theoretical forcing functions.
Analysis_Type='Newmark'; %Enter 'CDM','Newmark'
```

Figure 11: Sample User Input 5 DOF chimney

The file also contains two additional sample files, one for a three-story chimney, and another where the user can create an n-story chimney with the identical *m*, and no initial displacements and velocity simply typing the number of degrees of freedom the structure is to be modeled as.

2.3 MDOF_ForcingFunction.m

Just as was the case in the SDOF **ForcingFunction.m**, after assigning structural and analysis properties, the user must either create a forcing function or assign a ground motion. The steps are near identical as outlined in Section 1.3 of this report, with the only change being instead of creating a single column vector **[p]** that matches the time column vector **[t]** , stacks of column vectors listing ***p*** at all nodes at a given time ***t*** is created, where the length of the stack matches the length of the time vector **[t]**. The

figure below shows how a step force of 1000 kips applied at the top of the 5-story chimney may be input into **MDOF_ForcingFunction.m**.

```
%% SAMPLE FORCE INPUT #1:5-Story Chimney with step force applied at top
po=[0;0;0;0;0]; %initial force -kips [assume 0 at t=0 of step force]
p=[0;0;0;0;1000]; %kips; force at every node--@t>0 of step force
%^as it's written, the top story will have a step force of 1000 kips
%applied to it.

p_v=zeros(nn,1,(length(t))); %create a stack of column vectors
p_v(:,1,1)=po; %assign its initial column vectors the initial force vector
for i=2:length(t)
    p_v(:,1,i)=p; %assign a steady force of p vector for the subsequent
    %force column vectors
end
```

Figure 12: A stack of p vectors, p_v is assigned for this step force of 1000 kips applied at the top of the 5-dof chimney.

If the user wishes to input ground motion as the forcing function, the exact same line of codes as exemplified in Figure 4 is used to attain interpolated values of ground acceleration that matches the size of the time vector **[t]**, with the only difference being that the last two lines of Figure 4 (attaining a single p column vector) will be replaced by the following to attain stacks of force vectors.

```
influ=ones(nn,1); %since only a chimney is being modeled.
p_v=zeros(nn,1,(length(t))); %create a stack of column vectors
for i=1:length(t)
    p_v(:,1,i)=-m*influ*ag(i);
end
```

Figure 13: p_v for EQ motion

Note that the influence vector {i} in this case is just a column vector of 1's that match the length of the number of degrees of freedom, since a chimney's horizontal displacement is only of interest in this file.

2.4 MDOF_Run.m

Similar to the **Project_Begin.m** file, after the user inputs structural properties and analysis method under **User_Input_MDOF.m**, then creating a stack of force vector *p_v* under **MDOF_ForcingFunction.m**,

the user can then run the file **MDOF_Run.m** in the directory to output plots regarding displacement vs. time, the chimney's displacement response animation, and ground motion if applicable.

MDOF_Run.m creates quite a few more intermediary variables and matrices than its SDOF counterpart as variables in SDOF file becomes either matrices, column vectors, or stacks of column vectors.

The first step in the script is to capture the amount of degrees of freedom the user has input by counting the size of the mass vector *m* the user has created. The file also creates a time vector *t* that starts from 0, increases by a time step dt all the way to the time range *tr* that the user has input under **User_Input_MDOF.m**.

2.4.1 Mass Matrix [m]

The *matrix* **[*m*]** gets created next, by putting the values of the mass vector in the user input along the trace of the matrix. Note, this may not always be the case if the structure's degrees of freedom were not aligned like that of a chimney; read Chopra's Section 9.2.4:Inertia Forces for explanations.

2.4.2 Statically condensed $[\widehat{k}]$

The program then proceeds to create a statically condensed $[\widehat{k}]$ matrix. To discuss in brief, (as static condensation is covered in most Matrix Analysis textbook), the $[\widehat{k}]$ matrix is created by first assembling a 2X2 element stiffness matrix into a global *k* matrix and then partitioning them into four quadrants. The assembled k matrix can be written as seen in (2.4.2.1),

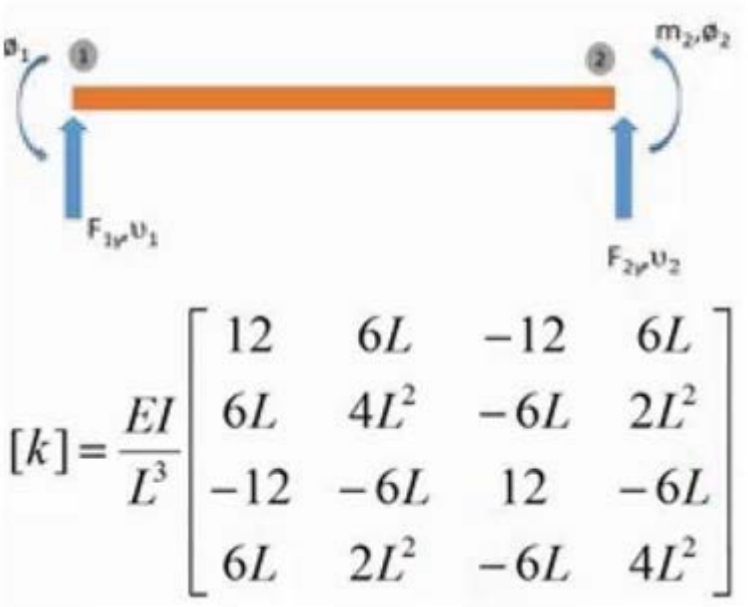

Figure 14: Element Stiffness Matrix taking flexure and shear into account.

$$k_{assembled} = \begin{bmatrix} k_{TT} & k_{TO} \\ k_{OT} & k_{OO} \end{bmatrix} \tag{2.4.2.1}$$

where the translational degrees of freedom are numbered first, followed by the rotational degrees of freedom. From here on the statically condensed stiffness matrix, $[\widehat{k}]$ that conveniently only deals with translational degrees of freedom, can be defined as:

$$[\widehat{k}] = [k_{TT}] - [k_{TO}][k_{OO}]^{-1}[k_{OT}] \tag{2.4.2.2}$$

For convenience in the report, as well as the actual program, $[\widehat{k}]$ is written interchangeably with [k].

2.4.3 Modal Frequencies ω and Mode Shapes φ

After the condensed **[k]** matrix is attained, the program follows the procedure as noted in Chopra's Section 10.2 to find modal frequencies $\omega_i$by method of Generalized Eigenvalue Problem as follows:

$$\det[\boldsymbol{k} - {\omega_n}^2 \boldsymbol{m}] = 0 \tag{2.4.3.1}$$

where ${\omega_n}^2$ serves as the eigenvalues for the problem. Once the eigenvalues are solved for using MATLAB's built-in function, these values are square rooted to obtain ω_n. There will be an *n* amount of

ω_n values, and these are sorted from least to greatest, where the least value corresponds to the first mode shape, and the last mode corresponds to the final mode shape.

At this stage, the script goes to find a corresponding mode shape (eigenvectors) for every modal frequency, which in equation form, can be expressed as:

$$[\boldsymbol{k} - {\omega_n}^2 \boldsymbol{m}]\varphi_n = \boldsymbol{0} \tag{2.4.3.2}$$

Once the mode shapes are found for given modal frequency, each mode shape vector φ_n is scaled so that its first entry is set to 1, and then stacked column by column as one 2D $[\boldsymbol{\varphi_n}]$ =$[\varphi_1, \varphi_2 \ldots \varphi_{dof}]$.

2.4.4 Damping Matrix [c]

Rayleigh Damping is used as written in Chopra's Section 11.4.1. It assumes that the damping matrix **[c]** is given by:

$$\boldsymbol{c} = a_o \boldsymbol{m} + a_1 \boldsymbol{k} \tag{2.4.4}$$

where a_o and a_1 are given by:

$$a_o = \xi \frac{2\omega_1 \omega_2}{\omega_1 + \omega_2} \qquad a_1 = \xi \frac{2}{\omega_1 + \omega_2} \tag{2.4.5}$$

This assumes that the first two modes dominate the other two and that the damping ratio is assumed to be the same, which is reasonable based on experimental data (Chopra 493).

2.4.5 Uncoupling to [M], [K], [C]

After the modal shapes, modal frequencies, the mass matrix, the stiffness matrix, and the damping matrix are all found, the script takes the triple product of these product as stated in equation (2.1.4).

2.4.6 Analysis Method Selection

Depending on what the user selected as the analysis type and whether or not ground motion will be fed into the **MDOF_ForcingFunction.m**—**MDOF_Run.m** will run one of the four scripts written into the directory listed as below:

1. **MDOF_Newmark.m**
2. **MDOF_Central_Difference_Method.m**
3. **MDOF_Newmark_Ground_Motion.m**
4. **MDOF_Central_Difference_Method_Ground_Motion.**

All four files will output plot displacement vs. time, and animation of a chimney undergoing displacement vs. time. The first two files however, will plot an additional plot showing modal coordinates vs. time, while the latter two files will plot an additional plot showing acceleration time history of the earthquake chosen. The differences between these two sets of outputs will be made more evident as the report goes along.

2.5 Numerical Integration of MDOF System

The numerical integration of MDOF system is simply an extension of the numerical integration methods established for SDOF system in Sections 1.5~1.7. The main idea is that responses that were scalar quantities in the SDOF case become matrix equations, and therefore solving for a set of variables at times involve matrix inversions. And although each of the methods do iterate to attain $\boldsymbol{[u_i]}$ at each time step just like in the SDOF case, since the iterations are based off the modal equilibrium equation of (2.1.3), the bulk of the work that gets iterated on each time step at an analogous level to SDOF's *u*, *u*', and *u*" are actually **[q], [q'],** and **[q"]**.

2.5.1 MDOF Newmark's Method

The concepts established in Section 1.6 can be readily extended to MDOF systems, where the scalar equations of incremental equilibrium all now become matrix equations. The average acceleration method is used for the coefficients of γ and β as the method is unconditionally stable, as well as the fact that the linear acceleration method requires a very small time step for the solution to not "blow up"

at the expense of processing speed, as this program incorporates all five mode shapes while generally two is sufficient.

$$(q_n)_0 = \frac{\varphi_n^T \boldsymbol{m u_0}}{\varphi_n^T \boldsymbol{m} \varphi_n} \qquad (\dot{q}_n)_0 = \frac{\varphi_n^T \boldsymbol{m \dot{u}_0}}{\varphi_n^T \boldsymbol{m} \varphi_n} \tag{2.5.1.1}$$

$$\boldsymbol{P_0} = \boldsymbol{\varphi^T p}_0 \tag{2.5.1.2}$$

$$\ddot{\boldsymbol{q}}_0 = \boldsymbol{M^{-1}}[\boldsymbol{P_0} - \boldsymbol{C}\dot{\boldsymbol{q}}_\mathbf{0} - \boldsymbol{K q_0}] \tag{2.5.1.3}$$

$$\widehat{\boldsymbol{K}} = \boldsymbol{K} + \frac{\gamma}{\beta \Delta t}\boldsymbol{C} + \frac{1}{\beta(\Delta t)^2}\boldsymbol{M} \tag{2.5.1.4}$$

$$\boldsymbol{a} = \frac{1}{\beta \Delta t}\boldsymbol{M} + \frac{\gamma}{\beta}\boldsymbol{C} \tag{2.5.1.5}$$

$$\boldsymbol{b} = \frac{1}{2\beta}\boldsymbol{M} + \Delta t\left(\frac{\gamma}{2\beta} - 1\right)\boldsymbol{C} \tag{2.5.1.6}$$

At this point, changes in modal responses of coordinates, and its first and second derivatives are all calculated per each time step in an iteration.

$$\boldsymbol{P_i} = \boldsymbol{\varphi^T p}_i \tag{2.5.1.7}$$

$$\Delta \boldsymbol{P}_i = \boldsymbol{P}_{i+1} - \boldsymbol{P}_i \tag{2.5.1.8}$$

$$\Delta \widehat{\boldsymbol{P}}_i = \Delta \boldsymbol{P}_i + \boldsymbol{a}\dot{\boldsymbol{q}}_\boldsymbol{i} + \boldsymbol{b}\ddot{\boldsymbol{q}}_\boldsymbol{i} \tag{2.5.1.9}$$

$$\Delta \boldsymbol{q_i} = \widehat{\boldsymbol{K}}^{\boldsymbol{-1}} \Delta \widehat{\boldsymbol{P}}_i \tag{2.5.1.10}$$

$$\Delta \dot{\boldsymbol{q}}_\boldsymbol{i} = \frac{\gamma}{\beta \Delta t}\Delta \boldsymbol{q_i} - \frac{\gamma}{\beta}\dot{\boldsymbol{q}}_\boldsymbol{i} + \Delta t\left(1 - \frac{\gamma}{2\beta}\right)\ddot{\boldsymbol{q}}_\boldsymbol{i} \tag{2.5.1.11}$$

$$\Delta \ddot{\boldsymbol{q}}_\boldsymbol{i} = \frac{1}{\beta(\Delta t)^2}\Delta \boldsymbol{q_i} - \frac{1}{\beta \Delta t}\dot{\boldsymbol{q}}_\boldsymbol{i} - \frac{1}{2\beta}\ddot{\boldsymbol{q}}_\boldsymbol{i} \tag{2.5.1.12}$$

After changes in modal coordinates and its two derivatives are found, the next step's dynamic responses are simply added from the previous steps as follows.

$$\boldsymbol{q}_{i+1} = \boldsymbol{q}_i + \Delta \boldsymbol{q}_i \tag{2.5.1.13}$$

$$\dot{\boldsymbol{q}}_{i+1} = \dot{\boldsymbol{q}}_i + \Delta \dot{\boldsymbol{q}}_i \tag{2.5.1.14}$$

$$\ddot{\boldsymbol{q}}_{i+1} = \ddot{\boldsymbol{q}}_i + \Delta\ddot{\boldsymbol{q}}_i \tag{2.5.1.15}$$

Finally, the modal coordinates are transformed into real space by multiplying it by the $\boldsymbol{\varphi}$ vector to attain actual displacements of all the nodes over time.

$$\boldsymbol{u}_{i+1} = \boldsymbol{\varphi}\boldsymbol{q}_{i+1} \tag{2.5.1.16}$$

2.5.1.1 Programming Iteration on Matlab

After initial variables and matrices are coded into MATLAB following formulas (2.5.1.1)~(2.5.1.6), the program then enters into an iterative loop as exemplified in the following figure to create stacks of column vectors of q and u vs. the time vector t.

```
%% Newmark's Method Iterative Process.
for i=1:length(t)-1

    dP_v(:,:,i)=P_v(:,:,i+1)-P_v(:,:,i); %Eqn 2.5.1.8
    dP_hat(:,:,i)=dP_v(:,:,i)+a*dq(:,:,i)+b*aq(:,:,i); %Eqn 2.5.1.9

    del_q(:,:,i)=Kh^(-1)*dP_hat(:,:,i);
    ^%Eqn 2.5.10
    del_dq(:,:,i)=(y/(beta*dt))*del_q(:,:,i)-(y/beta)*dq(:,:,i)+dt*(1-(y/(2*beta)))*aq(:,:,i);
    %^Eqn 2.5.1.11
    del_aq(:,:,i)=(1/(beta*dt^2))*del_q(:,:,i)-(1/(beta*dt))*dq(:,:,i)-(1/(2*beta))*aq(:,:,i);
    %^Eqn 2.5.1.12

    q(:,:,i+1)=q(:,:,i)+del_q(:,:,i); %Eqn 2.5.1.13
    dq(:,:,i+1)=dq(:,:,i)+del_dq(:,:,i); %Eqn 2.5.1.14
    aq(:,:,i+1)=aq(:,:,i)+del_aq(:,:,i); %Eqn 2.5.1.15

    u(:,:,i+1)=phi*q(:,:,i+1); %Eqn 2.5.1.16
end
```

Figure 15: Newmark's Method coded in MATLAB

2.5.2 Central Difference Method (CDM)

As developed in Section 1.5, the Central Difference Method is particularly easy to implement into an MDOF model. Again the scalar equations developed in Section 1.5 all become matrix equations, and additional features must arise from needing to transform initial conditions on

nodal displacements to modal coordinates and then transforming them back to nodal displacements at the next time step.

The solutions will "blow up" giving meaningless results if dt>0.1T_J where T_J is the natural period of the last mode. The program by default includes all the modes, and if the solution is to blow up, the code will trigger an error in the program, where it prompts the user to lower the dt value below a fully calculated number of 0.1*T_J.

The following equations are the iterative process as described in Chopra's Section 15.2.1.

$$(q_n)_0 = \frac{\varphi_n^T \boldsymbol{m u_0}}{\varphi_n^T \boldsymbol{m} \varphi_n} \qquad (\dot{q}_n)_0 = \frac{\varphi_n^T \boldsymbol{m \dot{u}_0}}{\varphi_n^T \boldsymbol{m} \varphi_n} \tag{2.5.2.1}$$

$$\boldsymbol{P_0} = \boldsymbol{\varphi^T p}_0 \tag{2.5.2.2}$$

$$\ddot{\boldsymbol{q}}_0 = \boldsymbol{M^{-1}}[\boldsymbol{P_0} - \boldsymbol{C}\dot{\boldsymbol{q}}_0 - \boldsymbol{K q_0}] \tag{2.5.2.3}$$

$$\boldsymbol{q}_{-1} = \boldsymbol{q_0} - \Delta t \dot{\boldsymbol{q}}_0 + \frac{(\Delta t)^2}{2} \ddot{\boldsymbol{q}}_0 \tag{2.5.2.4}$$

$$\widehat{\boldsymbol{K}} = \frac{\boldsymbol{M}}{(\Delta t)^2} + \frac{\boldsymbol{C}}{2\Delta t} \tag{2.5.2.5}$$

$$\boldsymbol{a} = \frac{1}{(\Delta t)^2}\boldsymbol{M} - \frac{1}{2\Delta t}\boldsymbol{C} \tag{2.5.2.6}$$

$$\boldsymbol{b} = \boldsymbol{K} - \frac{2}{(\Delta t)^2}\boldsymbol{M} \tag{2.5.2.7}$$

At this point, changes in modal coordinates are calculated per each time step in an iterative process.

$$\boldsymbol{P_i} = \boldsymbol{\varphi^T p}_i \tag{2.5.2.8}$$

$$\widehat{\boldsymbol{P}}_i = \boldsymbol{P}_i - \boldsymbol{a q_i} - \boldsymbol{b q_i} \tag{2.5.2.9}$$

$$\boldsymbol{q_i} = \widehat{\boldsymbol{K}}^{-1}\widehat{\boldsymbol{P}}_i \tag{2.5.2.10}$$

Finally, the modal coordinates are transformed into real space by multiplying it by the $\boldsymbol{\varphi}$ vector to attain actual displacements of all the nodes over time.

$$\boldsymbol{u}_{i+1} = \boldsymbol{\varphi}\boldsymbol{q}_{i+1} \qquad (2.5.2.11)$$

2.5.2.1 Programming Iteration on Matlab

After initial variables and matrices are coded into MATLAB following formulas (2.5.2.1)~(2.5.2.7), the program then enters into an iterative loop as exemplified in the following figure to create stacks of column vectors of q and u vs. the time vector t.

```
%% CDM iteration steps
for i=2:length(t)
    P_hat(:,:,i)=P_v(:,:,i)-a*q(:,:,i-1)-b*q(:,:,i);  % Eqn 2.5.2.9
    q(:,:,i+1)=Kh^(-1)*P_hat(:,:,i);  % Eqn 2.5.2.10
    u(:,:,i+1)=phi*q(:,:,i+1);  % Eqn 2.5.2.11
end
```

Figure 16:CDM iteration in concise lines of code.

2.6 MDOF Plots and Animations

Once numerical analysis is applied by means of Newmark's Method or the Central Difference Method, modal coordinate vectors $\mathbf{[q]}_I$, as well as displacement vectors $\mathbf{[u]}_I$ are created for every degree of freedom that matches the length of the time vector $\mathbf{[t]}_i$. At this point, the program is ready for output, and a full-screen figure comprised of three subplots will appear as seen in the following figures below.

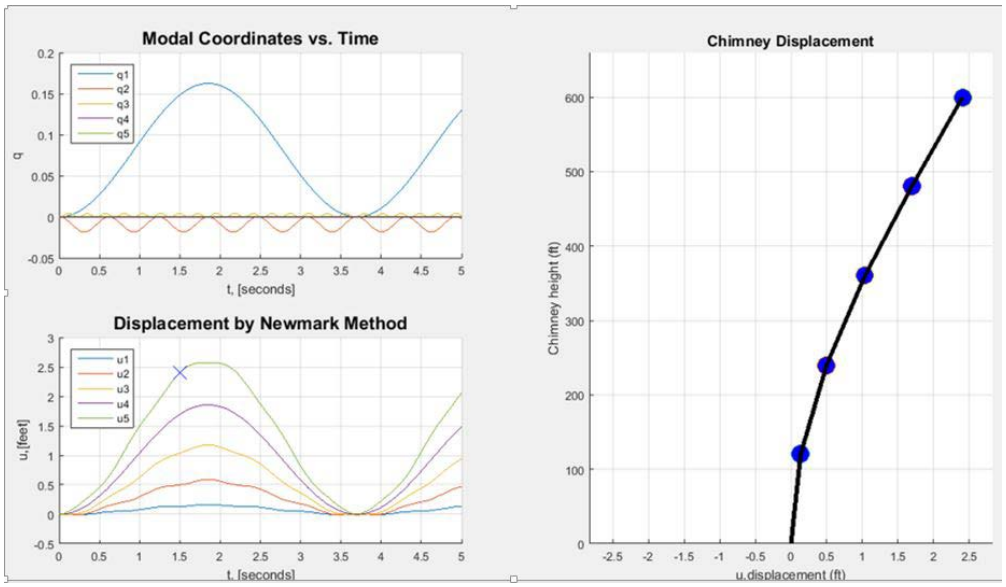

Figure 17:Set of Plots when Newmark's Method is chosen and a step force of 1000k at top of chimney is input

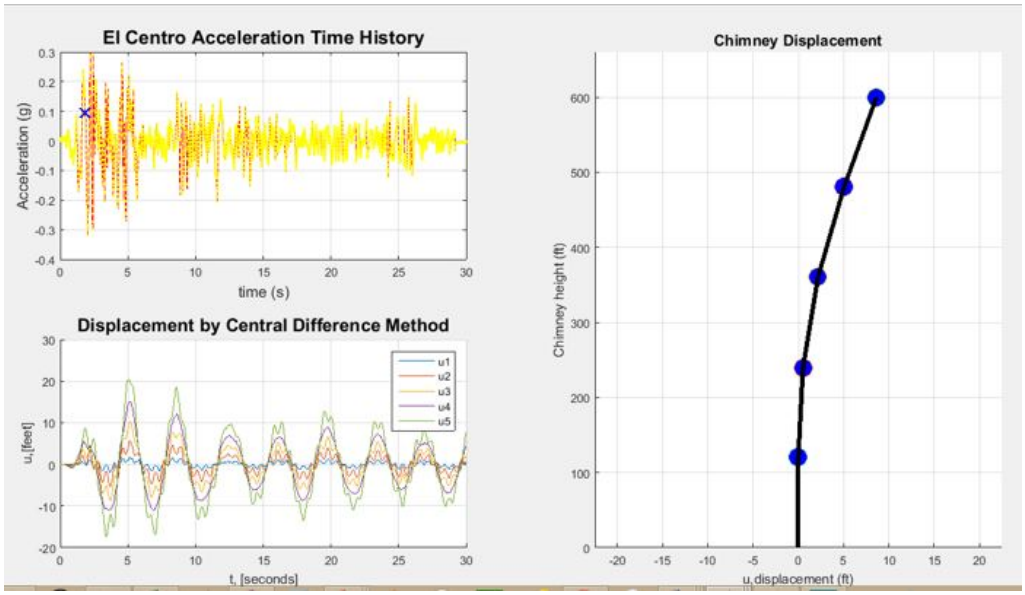

Figure 18: Set of Plots when CDM is chosen, and El Centro's EQ motion is selected as the Forcing Function.

If a theoretical forcing function is fed into **MDOF_ForcingFunction.m**, the program's first subplot will feature a plot showing modal coordinates vs. time as seen in Figure 17. The user or student will be clearly able to see the exponential decay in contributions from mode *n*, as the number of modes included in the analysis increases.

If ground motion is fed into the Forcing Function file, the first subplot in the output will feature an acceleration time history plot as seen in Figure 18.

Both Figures 17, and 18's remaining two plots will display a displacement vs. time graph with the title displaying what numerical analysis method is being used, and an animation of the chimney that the user has specified under **User_Input_MDOF.m**

Note that the crosshairs present in both figures' plots are in sync with the chimney's translational motion that is being animated in the third subplot. For example, in Figure 17, one can see that the screenshot is snapped when time is ~1.5 seconds, and at that moment, the top of the chimney—denoted as u_5 --has displaced 2.5 feet to the right.

2.7 Power Spectral Density—Fast Fourier Transform

As a final optional figure, the user can choose to plot out the power spectral density function of the displacement of the 1st degree of freedom of the chimney. At the top of User_Input.m file, the user can define the variable *FFT* as anything or just simply comment it out if spectral density is not of interest as a dynamic analysis.

Suppose the user/student wished to find the Spectral Density function incurred by a free-vibration response where the initial displacement of the top of the chimney is set to a certain value.

The Power Spectral Density script titled **FFT_PSD.m** would plot the following figure.

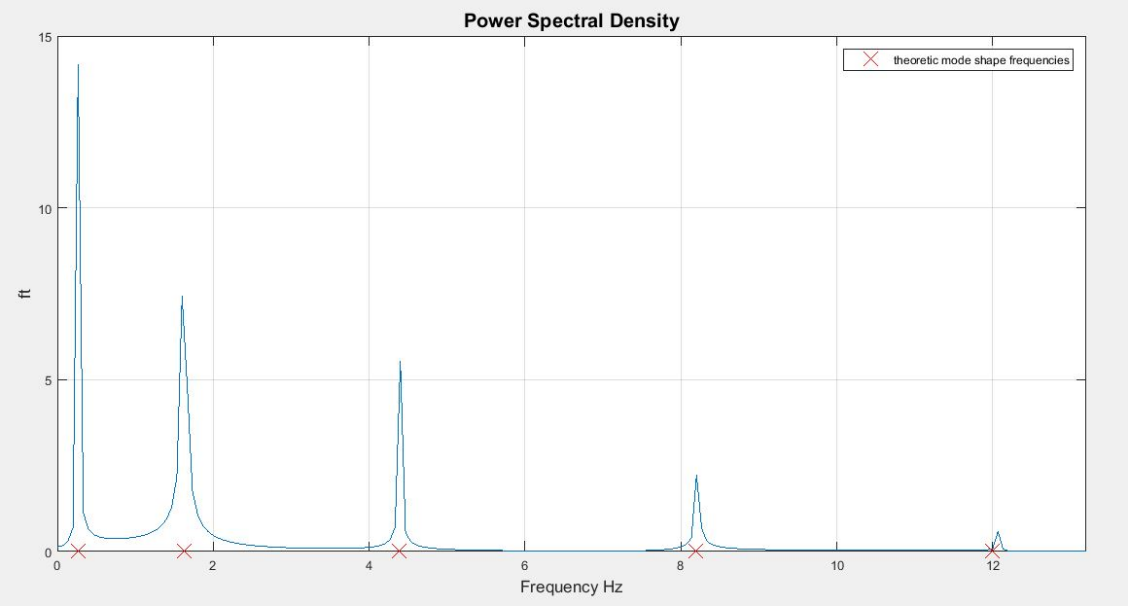

Figure 19: Power Spectral Density Chart of Free Vibration Response with 'X' marking theoretical modal frequencies

As the 5-story chimney is allowed to freely vibrate without a forcing function, there are virtually no disturbances, and therefore, the Spectral Density Chart nearly captures all the theoretical modal frequencies perfectly, as the peaks are aligned to the red x's (denoting the theoretic modal frequencies). with great accuracy.

Suppose the user now inputs ground motion, the plot would then look as follows.

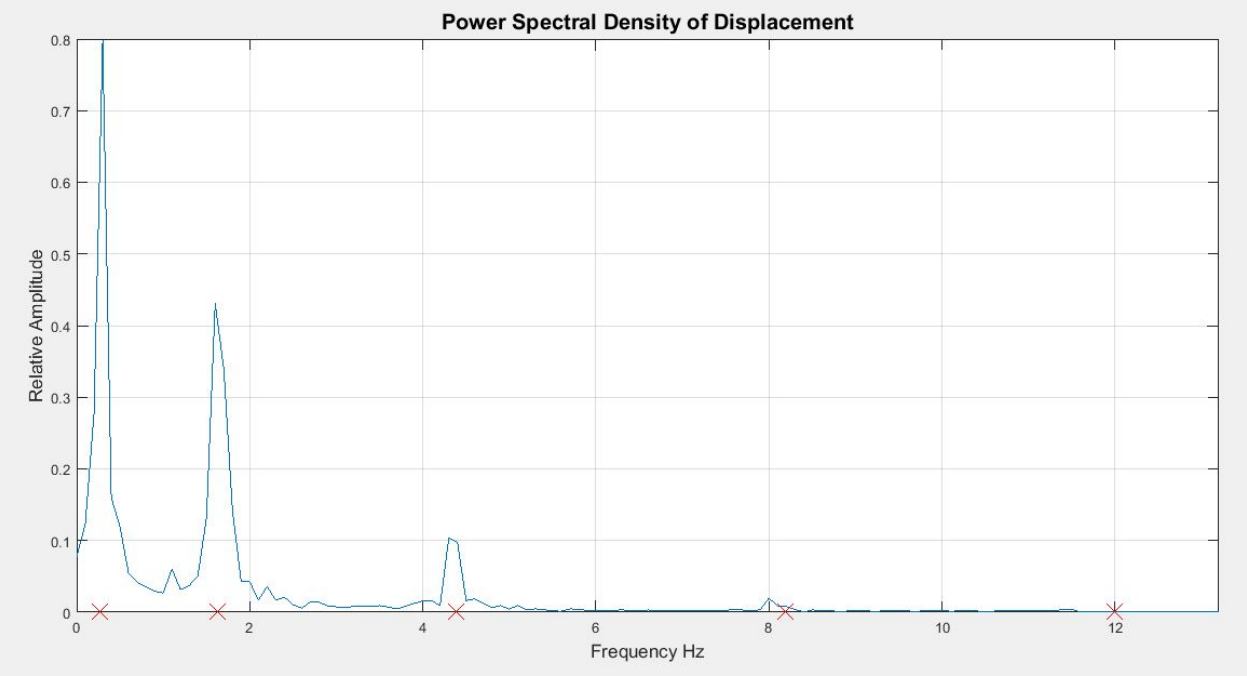

Figure 20: FFT plotted after displacement due to El Centro EQ is fed into the fft function.

As expected, the earthquake's erratic motion does not result in purely five peaks as there are too many disturbances; however, the Spectral Density Chart still is able to capture the first four modes with great accuracy, as the distinct peaks align themselves to the theoretic modal frequencies.

2.8 MDOF Analysis Flow Chart

Below shows a general outline of the process the folder MDOF goes through.

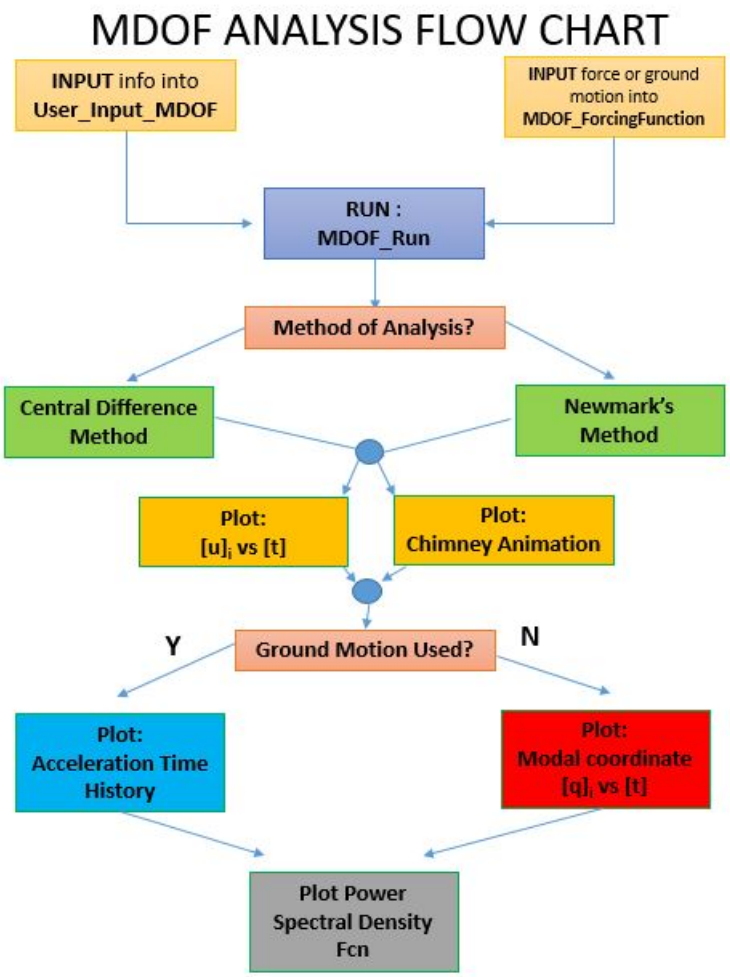

Figure 21: Flowchart of MDOF Analysis

3. Conclusion

Procedures regarding two sets of MATLAB based programs—(SDOF, MDOF)—that take user inputs of structural properties and forcing functions to output dynamic responses through numerical analysis have been greatly detailed and outlined in this document.

After reading through this report, a person with basic understanding of MATLAB and structural dynamics should be able to enter their own inputs for structural properties and create forcing functions to produce outputs in response data in order to extend and clarify his or her knowledge in structural dynamics.

References

Chopra, A. (2007). *Dynamics of Structures: Theory and Applications to Earthquake Engineering* (Third ed.). Englewood Cliffs, New Jersey: Prentice Hall.

Dorran, D. (2012, June 14). Using Matlab's fft function. Retrieved May 18, 2015, from https://www.youtube.com/watch?v=dM1y6zfQkDU

Ian, T. (2012, May 17). FFT Tutorial. Retrieved May 18, 2015, from https://www.youtube.com/watch?v=zKKGA30bHG0

Spiridonov, A. (2011, April 29). Runge Kutta 4th order method for ODE2. Retrieved May 18, 2015, from https://www.youtube.com/watch?v=smfX0Jt_f0I