6.3 ELEMENT BEHAVIOR

6.3.1 Column behavior

The response of the column impacted by BHA in the loaded wall is shown in Figure 6.8, where the horizontal displacement of the control point is plotted against the maximum moment recorded in column G1.4 during the pushover analysis in Figure 6.8 for both BHA impact cases. The maximum moment in the column occurred at the 6th story and the top story for the mid-height and roof-level BHA impact cases, respectively. Maximum column shear against the same displacement is plotted in Figure 6.9. The column yield moment is approximately 250,000 kip-in: it is attained at a relatively small displacement of 0.2 feet in the mid-height BHA impact case. In contrast, the column does not fail in moment under the BHA roof-level impact. Column shear force, shown in Figure 6.9, shows that the columns attains its shear strength (of approximately 10,000 kips) at approximately the same time when it yields in bending in the mid-height BHA impact case. This suggests that the column likely fails locally. The column yields in the case when BHA impacts at roof level, but just barely.

6.3.2 Beam behavior

The highest-loaded beam in the impacted wall was selected to investigate its failure mode. The relationship between beam moment and the horizontal displacement of the control point is shown in Figure 6.10, while the beam shear force is plotted in Figure 6.11. The maximum moments and shears for the mid-height and roof-level BHA impact cases occurred in the beam in between J and K on the 6th story and in between K and L on the top story on the loaded side of the event shield, respectively. The beam yield moment is approximately 6000 kip-inches. As can be seen in Figure 6.10, the beam yielded in the mid-height BHA impact case but remained elastic in the roof-level BHA impact case. Beam shear force, shown in Figure 6.11, shows that the columns attains its shear strength (of approximately 16,000 kips) slightly before it yields in bending in the mid-height BHA impact case. Shear strength of the beam was never exceeded in the roof-level BHA impact case, even though it was assumed the beam shear reinforcement was only the minimum required by ACI Code (4GR60 #4 hoops every 6” on center)

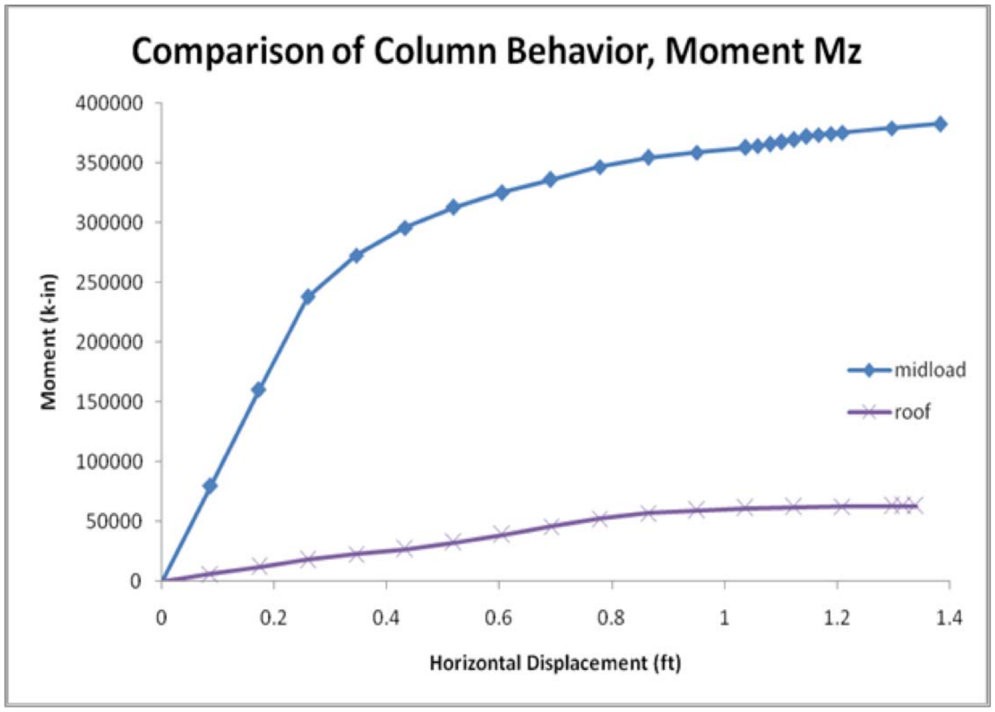

Figure 6.8 Column moment

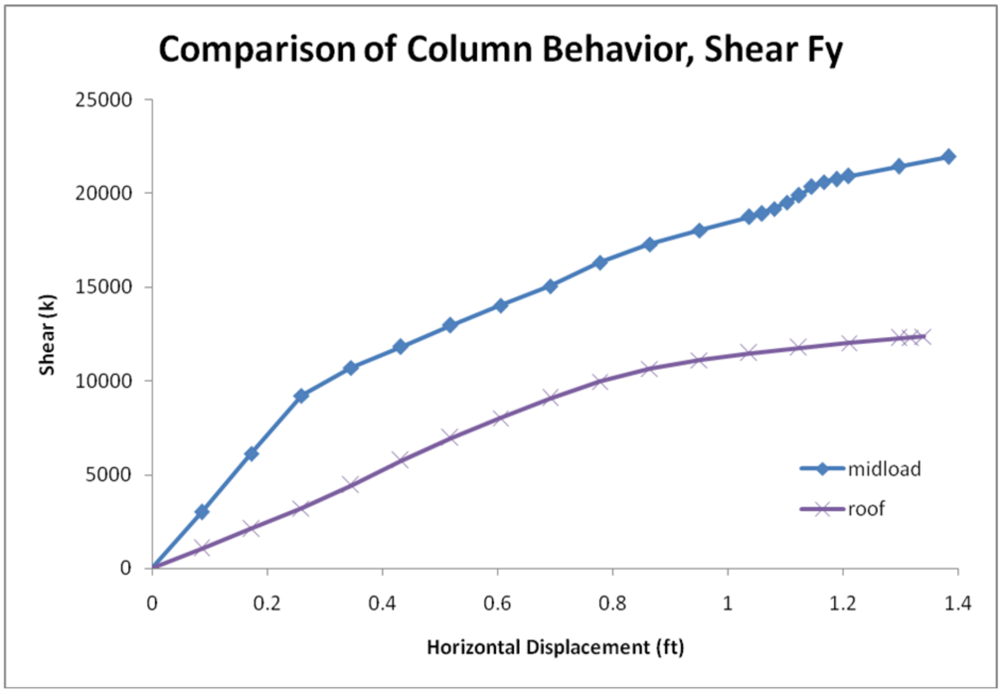

Figure 6.9 Column shear

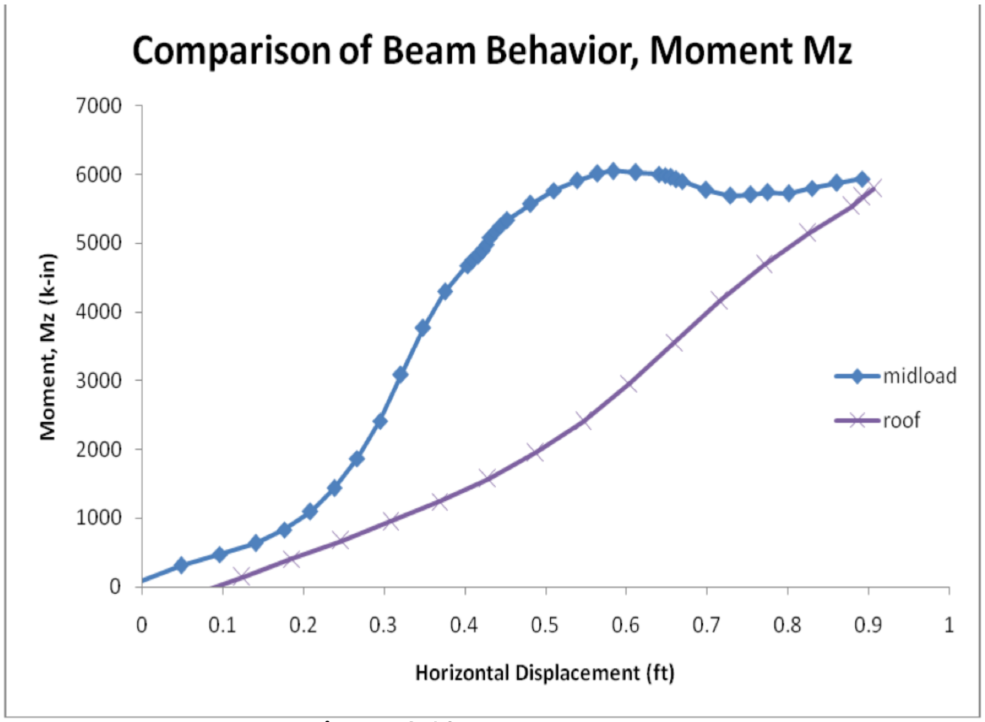

Figure 6.10 Beam moment

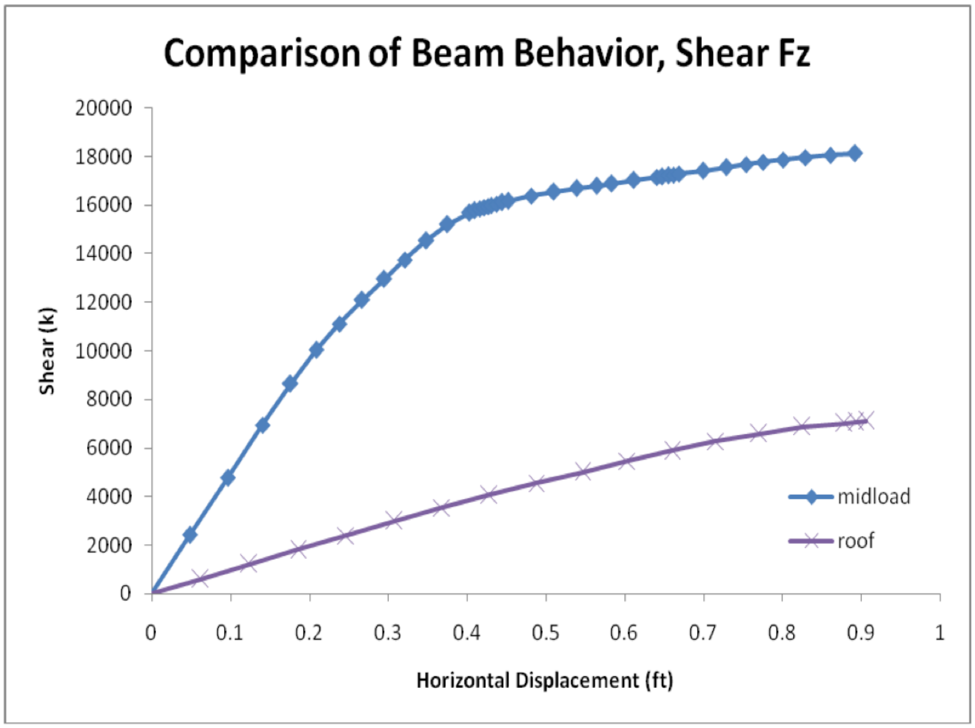

Figure 6.11 Beam shear

7 Conclusions

Due to the large amount of physical, social and economic damage that would occur if a nuclear power plant’s security was breached, protective structures, both physical and operational, need to be in place to mitigate a number of threats, including airplane crashes or deliberate airplane impact. This study focused on the ability of a decoupled external events shield using waffle slab construction to resist the impact from a BHA. The external event shield structure was designed as a separate structure that envelopes an internal citadel structure that houses the nuclear reactor and the power conversion units. In this study, the shield structure was designed as a hangar structure comprising four side walls and a roof diaphragm. The walls and the roof are designed as gills comprised of orthogonal beam and/or column elements, following a honeycomb principle to locally absorb and then distribute the impact of the aircraft. The analyses conducted in this study validated these behavior assumptions. A three-dimensional finite element structural model of the external event shield structure was created using the OpenSees finite element analysis framework to capture the nonlinear response of the external event shield elements. A displacement controlled pushover analysis was performed using a force pattern corresponding to the impact of a hypothetical aircraft similar to Boeing 747 (BHA) flying horizontally into the structure. Two impact locations were investigated: one at mid-height and the other at the roof level of the impacted wall. Both deformed shapes and element forces and deformations were examined at critical locations to determine structural performance.

Based on the results of the analyses conducted in this study, it can be concluded that the external events shield structure can be designed to withstand a BHA impact load, thus preventing unacceptable damage to the inner citadel structure. The mid-height BHA impact case was found to be more critical than the roof level BHA impact case. In the mid-height BHA impact case the loaded wall responded inelastically, with beams and columns in the zone corresponding to the load pattern extent (approximately 100 feet) yielding in bending and approaching or exceeding their shear strength. The rest of the external event shield structure remained elastic.

Maximum displacement of the impact point was 3.7 feet, and the maximum interstory drift ratio in the impacted wall was 5.8%. Under roof level BHA impact, the external events shield structure as a whole stayed elastic, with only slight localize yielding in bending at the point of impact. Under this load case, the maximum observed displacement and interstory drift ratio were 0.95 ft and 0.9%, respectively. Generally, the column behavior was controlled by bending while the beam behavior was controlled by shear. Given the beam and column sections adopted for the external event structure in this study, the waffle slab type of construction of the external events shield allowed the loaded wall to resist and transfer resultant forces from the airplane crash to the rest of the external events shield.

8 Future Work

Future research is necessary to more realistically design and model the decoupled external shield structure protective structures for base isolated nuclear power plants. This research should focus on the following three areas:

1. Design and model refinement. Iterative design of beam and column sections for the external event shield structure should be continued until an optimal solution is found. Both design and model should be refined to account for the participation of the external façade wall and the roof slab in beam and column bending. Finally, additional pushover load cases to model eccentric BHA impact in horizontal flight as well impact of a descending BHA should be done.
2. Dynamic effects: The external event shield structure should be examined for the dynamic effects of both BHA impact and earthquake loads. Such dynamic analysis is particularly important with regards to the BHA impact load to determine what local effects may occur on the structural element level. This includes penetration and scabbing as well as assessing if the connected beams on the loaded wall, and subsequently the rest of the external events shield, will have enough time to react to the aircraft load or if the elements will, in essence, be blind to the load since the impact occurs so quickly. Using more sophisticated elements or models may therefore become necessary. To validate the dynamic characteristics of the model, a shaking table test of a reduced-scale model may be necessary.
3. External event shield structure configuration and geometry: Further research may need to be performed on different external events shield geometry, such as cylindrical or dome-like shapes. However, to fully determine the feasibility of an external events shield, the cost of the protective structure needs to be ascertained. Finally, research should be performed on alternative protection strategies, such as partially or fully burying the citadel, to find an optimal solution.

9 Acknowledgements

This project was given financial support by US Department of Energy, Office of Nuclear Energy through the 2008 Global Nuclear Energy Partnership (GNEP) program. I would also like to gratefully acknowledge the rest of this UC Berkeley GNEP project team including Professor Bozidar Stojadinovic, Professor Per Peterson, Dr. Robert Budnitz, Mr. Eric Keldrauk, Mr. Mike Mieler, Mr. Edward Blandford, and Mr. Mike Laufer.

10 References

Beck, P.W. Nuclear Energy in the Twenty-First Century: Examination of a Contentious Subject. *Annual Reviews Energy Environment*, 24, 113-37, 1999.

Bulson, P.S. *Explosive loading of engineering structures: A history of research and a review of recent developments.* E &FN SPON, London, 1997.

Conrath, E.J., Krauthammer, T., Marchand, K.A., and Mlakar, P.F. *ASCE Structural design for physical security: State of the practice*. ASCE, 1999.

Guirgis, S. and Guirguis, E. An energy approach study of the penetration of concrete by rigid missiles. Nucl. Eng. Des. (2009), doi: 10.1016j.nucengdes.2008.11.016.

Krauthammer, T. *Modern Protective Structures*. CRC Press, 2008.

Nazaroff, W. Lecture #21: Nuclear Power. *UC Berkeley: CE 107 Climate – Change Mitigation Reading Materials*, Spring 2008.

OpenSees. "OpenSees" <http://opensees.berkeley.edu/index.php>. (Accessed May 01, 2009).

Petrangeli, E.G. and Forasassi, G. Large Airplane Crash on a Nuclear Plant. *Proceedings of ICAPP 2007*, Paper 7081, 2007.

Quan, X., Itoh, M., Cowler M., Katayama M., Birnbaum, N., Gerber, B., and Fairlie, G. Applications of a coupled multi-solver approach in evaluating damage of reinforced concrete walls from shock and impact. *SMiRT18-J04-4*, Beijing, China, 2005.

Riera, J.D. On the stress analysis of structures subjected to aircraft impact forces. *Nuclear Engineering and Design*, 8, 415-426, 1968.

Smith, P.D. and Hetherington, J.G. *Blast and ballistic loading of structures.* Butterworth-Heinemann Ltd, Oxford, 1994.

World Nuclear Association. Nuclear Power in the World Today. *http://www.world-nuclear.org/info/inf01.html*, accessed April 26, 2009.

Appendix A. OpenSees Model Explanatory Notes

Table of Contents

A.1 Model start script (Model.GoAll)

This script sources in the scripts that define geometry, build elements, define recorders and apply gravity load, and apply the pushover load. This was done so that if there was a problem with one of the scripts, the user would be able to pinpoint where the error occurred more easily.

A.2 Defining geometry (WaffleNodes)

The goal of this script was to automate node generation in a way that would help the user easily define where the node was given its node number designation. It should be noted that in this model, the horizontal plane is defined by the XZ plane and the vertical planes are defined by the XY and YZ planes. Also, all units in this model correspond to pounds and feet.

The external events shield geometry is defined with the following variables: ShellWidth, ShellLength, ShellHeight, LCol, LBeam, and LGird. Beams are designated as the beam members that lie parallel to the X axis and therefore related to ShellWidth, which is the dimension of the external events shield along the X axis. Relatedly, girders are designated as the beam members that lie parallel to the Z axis and therefore are related to ShellHeight, the dimension of the external events shield along the Z axis. Approximate masses are then defined for each node taking into account self weight of beams, columns and external wall, as well as superimposed dead loads from storage equipment and live load. This is where the column and beam sections are defined using the following variables: HCol, BCol, HBeam, BBeam, HGird, and BGird. Masses are defined here instead of the gravity script to allow for easy nodal assignation in the for loops.

To allow for number of nodes to adjust with external events shield dimensions, a number of for loops were designed to automate node number and mass definition. Node numbers have six digits, the first two inform the user which X grid line the node is on, the next two which Y grid line the node is on, and the last two which Z grid line the node is on, where 010101 would be at (0,0,0). Therefore, node number 21008 is located at X gridline 2, Y gridline 10, and Z gridline 8. In this model that means node number 21008 is located at (7.5, 144, 96).

The rest of the script deals with defining the roof diaphragm where RoofGrid.Nodes is sourced in, fixing the base of the structure, and defining the pushover load nodes. The variable that defines whether the load is applied at the middle of the wall or at the roof level is ‘roof’. If ‘roof’ is set to 1, then the load is applied at the roof level, if it is set at 0, then the pushover load is applied at the middle of the wall.

A.3 Building elements (WaffleNodes.ElementBuild)

This script defines material, section, and element properties and then builds the elements (columns, beams, girders) and is largely adapted from OpenSees Example 7 by Silvia Mazzoni and Frank McKenna. Most of the defined variables are used many times in the OpenSees examples so the user should refer to the examples manual or user manual for more discussion on those terms. However, there is a ‘matfactor’ variable because originally, material strengths were varied to determine external events shield performance. Concrete and steel materials were assumed to be Concrete01 and Steel01. Corotational geometric transformation was used for higher accuracy. Nonlinear beam column elements were used to capture nonlinear behavior. The element assignments are also automated using for loops although no categorization like that used for node numbering was employed. Defining the roof grid elements is sourced in from RoofGrid.ElementBuild.

A.4 Applying gravity load (WaffleNodes.GravityLoad)

This script defines recorders as well as applies the gravity load. For this model, recorders were used for nodal displacements, support reactions, and element forces. Nodal displacement recorders were defined for all nodes on the loaded wall as well as a line of nodes along the roof at the middle of the roof along the X and Z axis, and one vertical line of nodes for all other walls (please see following figure). Gravity loads consider only the self-weight of the beams, columns, and external wall and defined in units of lb/ft to be applied along the elements uniformly. Defining roof gravity loads is sourced in from RoofGrid.Gravity.

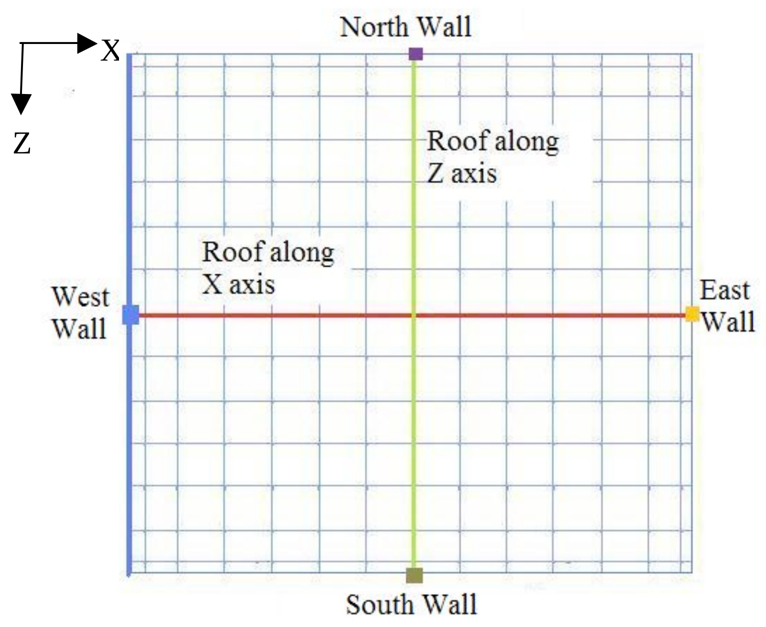

Figure A.1 Recorder positions on external events shield cross section

A.5 Applying pushover load (WaffleNodes.Analysis.PushDisplacementControl)

This script defines the application of the pushover load. The pushover analysis for this model is displacement controlled. Therefore, OpenSees tracks displacement of a control node, in this case, the node which the fuselage load is applied, 'MidPushLoad'. The maximum displacement of this node and the displacement increment is defined with variables Dmax and Dincr, respectively. Because the model is so large, constraints was defined as Transformation, RCM numberer and SparseSPD system were used. The pushover loads used were defined as fractions of the overall load such that when initially summed would equal one. This was done so that the 'pseudo-time' would always equal the base shear resultant from load application rather than a load factor. If analysis fails initially, a number of other methods for convergence are used including Newton with initial tangent, Broyden, and Newton with line search. This is adapted from Mazzoni and McKenna's Example 7 so please refer to the OpenSees user and examples manual for further discussion.

A.6 Roof grid scripts

Similar to the WaffleNodes scripts, the roof grid nodes automate nodal number and mass definition as well as element building and gravity load definition with RoofGrid.Nodes, RoofGrid.ElementBuild, and RoofGrid.Gravity respectively. The roof grid geometry is taken from the WaffleNodes script, defining the grid to be equally spaced at LBeam, with sections using HCol and BCol. The grid nodal mass takes into account self weight and live load.

A.7 Other scripts

A.7a Eigenvalue analysis (AnalysisOptn_eigen)

This script was adapted from Jeffrey Hunt's eigenvalue analysis. This script is sourced in after gravity load is applied to determine the first five mode periods of the structure.

A.7b Building RC section (BuildRCrectSection)

This script is taken from the OpenSees examples written by Mazzoni and McKenna. This builds the reinforced concrete section with rectangular patches given the defined section geometry, material properties, and steel reinforcement.

A.7c Display scripts (DisplayModel3D, DisplayPlane)

This script is taken from the OpenSees examples written by Mazzoni and McKenna. This displays a 3D perspective of the model as well as views of the XY, YZ, and XZ planes.

Appendix B. OpenSees Model Script

Table of Contents

B.1 Model start script (Model.GoAll)

```
puts "Assigning nodes and masses"
source {C:\Users\Joy\Desktop\MEng\NewModel\WaffleNodes.tcl}

puts "------------------------------------------------------------"
puts "Building elements"
source {C:\Users\Joy\Desktop\MEng\NewModel\WaffleNodes.ElementBuild.tcl}

puts "------------------------------------------------------------"
#Assigning Gravity loads
puts "Applying gravity loads"
source {C:\Users\Joy\Desktop\MEng\NewModel\WaffleNodes.GravityLoad.tcl}

#puts "------------------------------------------------------------"
puts "Performing Eigenvalue Analysis"
source {C:\Users\Joy\Desktop\MEng\NewModel\AnalysisOptn_eigen.tcl}
set eigFID [open NewModel/M1EigenVal.out w]
puts $eigFID [eigen  generalized    5]
close $eigFID
analyze  1  0.0001

puts "------------------------------------------------------------"
puts "Performing pushover analysis"
source
{C:\Users\Joy\Desktop\MEng\NewModel\WaffleNodes.Analysis.PushDisplacementC
ontrol.tcl}
```

B.2 Defining geometry (WaffleNodes)

```
# SET UP ----------------------------------------------------------
wipe;                  # clear memory of all past model definitions
model BasicBuilder -ndm 3 -ndf 6; # Define the model builder,
ndm=#dimension, ndf=#dofs
file mkdir NewModel;
set geometry [open NewModel/geometry.out w]
set masses [open NewModel/masses.out w]
puts $geometry "Node Number   X(ft) Y(ft) Z(ft)"

source {C:\Users\Joy\Desktop\MEng\NewModel\DisplayModel3D.tcl}
source {C:\Users\Joy\Desktop\MEng\NewModel\DisplayPlane.tcl}

# define GEOMETRY -------------------------------------------------
# define structure-geometry paramters
# Note:  All lengths are in feet!!  All weights are in lbs!  all time is
in s!

set ShellWidth 192;  # about 60 m
set ShellLength 192; # about 60 m
set ShellHeight 144; # about 45 m

set LCol 16;           # column height (parallel to Y axis), about 5 m
set LBeam 16;          # beam length (parallel to X axis), about 5 m
set LGird 16;          # girder length (parallel to Z axis), about 5 m

set NStory [expr $ShellHeight/$LCol];
set NBayX [expr $ShellLength/$LBeam + 2];          # +4 b/c need to
account for internal frame
set NBayZ [expr $ShellWidth/$LGird + 2];           # +4 b/c need to
account for internal frame

# Define MASSES ---------------------------------------------------
set g 32.2; #gravity, ft/s^2
set SDL 100.; #superimposed dead load, psf
set LL 20.; #Live load, psf

set ConcreteWeight 150.; #pcf
set SteelWeight 490.; #pcf
set wallt 2.; #wall thickness, ft
set concwallt 1.3; #concrete wall thickness, ft
set WallWeight [expr $concwallt*$LCol*$ConcreteWeight + [expr $wallt-
$concwallt]*$LCol*$SteelWeight]; #Wall load, lb/ft

#Column and Beam self weight, [lb]
set HCol 15.; # ft
set BCol 4.; # ft
set HBeam 4.; # ft
set BBeam $HCol; # ft
set HGird $HBeam; # ft
set BGird $BBeam; # ft
set ColBeamWallWeight [expr
$HCol*$BCol*$LCol*$ConcreteWeight+$HBeam*$BBeam*$LBeam*$ConcreteWeight +
$WallWeight*$LBeam];

#Bay loads, [lb]
set BayArea [expr $LBeam*$BBeam];
set BaySDL [expr $BayArea*$SDL];
set BayLL [expr $BayArea*$LL];

set DLFact 1.;
set LLFact 1.;
set NodeWeight [expr $ColBeamWallWeight*$DLFact + $DLFact*$BaySDL +
$LLFact*$BayLL];
set NodeMass [expr $NodeWeight/$g];

puts $masses "Nodal Mass is $NodeMass lb/g"
puts $masses "Node Number Px(lb-s^2/ft) Py(lb-s^2/ft) Pz(lb-s^2/ft)"

# Build MODEL ------------------------------------------------------------
# define nodes along axes parallel to X axis
set NodeN 10102;  #node number is Xgrid, Ygrid, Zgrid

for {set k 0} {$k <=1} {incr k 1} {
      if {$k==0} {
            set Z [expr $BBeam/2.];
      } else {
            set Z [expr $ShellWidth - $BBeam/2.]
      }
      set X 0.0;
      for {set i 0} {$i<=$NBayX} {incr i 1} {
            set Y 0.0;
            if {$i==1
                  || $i== [expr $NBayX - 1]
            } then {
                  if {$i==1} {
                        set X [expr $BBeam/2.];
                  } else {
                        set X [expr $X - $BBeam/2.];
                  }
                  for {set j 0} {$j<=$NStory} {incr j 1} {
                        node $NodeN $X $Y $Z
                        puts $geometry "$NodeN $X $Y $Z"
                        mass $NodeN [expr $NodeMass/4.] 0. [expr
$NodeMass/4.] 0. 0. 0.
                        puts $masses "$NodeN [expr $NodeMass/4.] 0 [expr
$NodeMass/4.]"
                        set Y [expr $Y + $LCol];
                        set NodeN [expr $NodeN + 100];
                  }
                  if {$i==1} {
                        set X [expr $X - $BBeam/2. + $LBeam];
                  } else {
                        set X [expr $X + $BBeam/2.];
                        }
                } else {
                        for {set j 0} {$j<=$NStory} {incr j 1} {
                                node $NodeN $X $Y $Z
                                puts $geometry "$NodeN $X $Y $Z"
                                mass $NodeN $NodeMass 0. $NodeMass 0. 0. 0.
                                puts $masses "$NodeN $NodeMass 0 $NodeMass"
                                set Y [expr $Y + $LCol];
                                set NodeN [expr $NodeN + 100];
                        }
                set X [expr $X + $LBeam];

                }
                set NodeN [expr $NodeN - $NStory*100 - 100 + 10000];
        }
set NodeN [expr $NodeN - $NBayX*10000 - 10000 +$NBayZ - 2];
}

#define Nodes along grid lines parallel to Z axis
set NodeN 20101;  #node number is Xgrid, Ygrid, Zgrid

for {set i 0} {$i <=1} {incr i 1} {
        if {$i==0} {
                set X [expr $BBeam/2.];
        } else {
                set X [expr $ShellLength - $BBeam/2.]
        }
        set Z 0.0;
        for {set k 0} {$k<=$NBayZ} {incr k 1} {
                set Y 0.0;
                if {$k==1
                        || $k== [expr $NBayZ - 1]
                } then {
                } else {
                        for {set j 0} {$j<=$NStory} {incr j 1} {
                                node $NodeN $X $Y $Z
                                puts $geometry "$NodeN $X $Y $Z"
                                mass $NodeN $NodeMass 0. $NodeMass 0. 0. 0.
                                puts $masses "$NodeN $NodeMass 0 $NodeMass"
                                set Y [expr $Y + $LCol];
                                set NodeN [expr $NodeN + 100];
                        }
                set Z [expr $Z + $LBeam];
                set NodeN [expr $NodeN - $NStory*100 - 100];
                }
        set NodeN [expr $NodeN + 1];
        }
set NodeN [expr $NodeN - $NBayZ - 1 +$NBayX*10000 - 20000];
}

# Define Roof Diaphragm -------------------------------------
source {C:\Users\Joy\Desktop\MEng\NewModel\RoofGrid.Nodes.tcl}  #if want
roof grid
```

```
# Fix supports
fixY 0.0 1 1 1 1 1 1   #Fix all nodes at base of structure

# Define Pushover Load Nodes
set roof 1;      # 0 for mid level load, 1 for roof level load
if {$roof == 0} {
      set MidPushLoad 20608;  #set middle pushover load to be middle of
building at mid level
      set PushHeight 80.;    #for mid level load
} else {
      set MidPushLoad 21008;            #set middle pushover load to be
middle of building at roof level
      set PushHeight $ShellHeight;      #for roof level load
}

set LeftPushLoad 1;
set RightPushLoad 3;
set MidZ [expr $ShellWidth/2.];         #set Z location of middle load
set WingL [expr 52.0 + 2.0/3.0];        #set distance between middle load
and left/right loads
node $LeftPushLoad [expr $BBeam/2.] $PushHeight [expr $MidZ - $WingL];
#node $MidpushLoad [expr $BBeam/2.] $PushHeight $MidZ];  #if there's not
already a node at this location specify a location
node $RightPushLoad [expr $BBeam/2.] $PushHeight [expr $MidZ + $WingL];
puts $geometry "LeftPushLoad $LeftPushLoad [expr $BBeam/2.] $PushHeight
[expr $MidZ - $WingL]"
puts $geometry "MidPushLoad $MidPushLoad [expr $BBeam/2.] $PushHeight
$MidZ"
puts $geometry "RightPushLoad $RightPushLoad [expr $BBeam/2.] $PushHeight
[expr $MidZ + $WingL]"

close $geometry
close $masses
puts "Nodes and masses assigned"

# Define DISPLAY ---------------------------------------------------------
set  xPixels 1200;     # height of graphical window in pixels
set  yPixels 800;# height of graphical window in pixels
set  xLoc1 10;    # horizontal location of graphical window (0=upper left-
most corner)
set  yLoc1 10;    # vertical location of graphical window (0=upper left-
most corner)
set dAmp 2;# scaling factor for viewing deformed shape, it depends on the
dimensions of the model
DisplayModel3D NodeNumbers $dAmp $xLoc1 $yLoc1  $xPixels $yPixels
```

B.3 Building elements (WaffleNodes.ElementBuild)

```
# Define Elements and Sections -------------------------------------------
# Adapted from # Example 7. 3D RC Frame and LibRCMaterials by Silvia
Mazzoni & Frank McKenna, 2006

source {C:\Users\Joy\Desktop\MEng\NewModel\BuildRCrectSection.tcl}
set Ubig 1.e10;              # a really large number

# Material Parameters ----------------------------------------------------
# confined and unconfined CONCRETE
# nominal concrete compressive strength
set matfactor 1.;
set fc          [expr -$matfactor*8.*1000.*144.];       # CONCRETE
Compressive Strength, psf   (+Tension, -Compression)
#set fc         [expr -6.*1000.*144.];
#set fc         [expr -8.*1000.*144.];
set Ec          [expr 57000*sqrt(-$fc/144.)*144.];# Concrete Elastic
Modulus, psf
set nu 0.2;
set Gc [expr $Ec/2./[expr 1+$nu]];      # Torsional stiffness Modulus, psf

# confined concrete
set Kfc 1.3;            # ratio of confined to unconfined concrete strength
set Kres 0.2;           # ratio of residual/ultimate to maximum stress
set fc1C [expr $Kfc*$fc];  # CONFINED concrete (mander model), maximum
                             stress, psf
set eps1C [expr 2.*$fc1C/$Ec];     # strain at maximum stress
set fc2C [expr $Kres*$fc1C];       # ultimate stress, psf
set eps2C  [expr 20*$eps1C];       # strain at ultimate stress
set lambda 0.1;              # ratio between unloading slope at $eps2 and
                                   initial slope $Ec
# unconfined concrete
set fc1U  $fc;               # UNCONFINED concrete (todeschini parabolic
                                   model), maximum stress, psf
set eps1U -0.003;            # strain at maximum strength of unconfined
                                   concrete
set fc2U [expr $Kres*$fc1U];       # ultimate stress, psf
set eps2U -0.01;             # strain at ultimate stress

# Assume no tensile-strength properties

set IDconcCore 1;
set IDconcCover 2;
uniaxialMaterial Concrete01 $IDconcCore $fc1C $eps1C $fc2C $eps2C;   #
Core concrete (confined)
uniaxialMaterial Concrete01 $IDconcCover $fc1U $eps1U $fc2U $eps2U;  #
Cover concrete (unconfined)

# REINFORCING STEEL parameters
set Fy [expr $matfactor*60.*1000.*144.];      # STEEL yield stress, psf
set Es [expr 29000.*1000.*144.];         # modulus of steel, psf
set Bs 0.01;                # strain-hardening ratio
set IDSteel 3;
uniaxialMaterial Steel01 $IDSteel  $Fy $Es $Bs

#Define Fiber Sections ---------------------------------------------------
set SectionType FiberSection ;

# Section tags
set ColSecTag 1;
set BeamSecTag 2;
set GirdSecTag 3;
set ColSecTagFiber 4;
set BeamSecTagFiber 5;
set GirdSecTagFiber 6;
set SecTagTorsion 70 ;

# Column and Beam geometry
# From RectModelNodesTest, take HCol, BCol, HBeam, BBeam, HGird, BGird
set cover [expr 2.5/12.];    # rectangular-RC-Column cover, ft
set numBarsTopCol 50;       # number of long-reinf bars on top layer
set numBarsBotCol 50;       # number of long-reinf bars on bottom layer
set numBarsIntCol 40;       # TOTAL number of reinf bars on the
                                  intermediate layers
set barAreaTopCol [expr 4./144.]; # long-reinf bar area, ft^2
set barAreaBotCol [expr 4./144.]; # long-reinf bar area, ft^2
set barAreaIntCol [expr 4./144.]; # long-reinf bar area, ft^2

set numBarsTopBeam 45;      # number of long-reinf bars on top layer
set numBarsBotBeam 45;      # number of long-reinf bars on bottom layer
set numBarsIntBeam 24;      # TOTAL number of reinforcing bars on the
                                  intermediate layers
set barAreaTopBeam [expr 4./144.];# long-reinf bar area, ft^2
set barAreaBotBeam [expr 4./144.];# long-reinf bar area, ft^2
set barAreaIntBeam [expr 4./144.];# long-reinf bar area, ft^2

set numBarsTopGird $numBarsTopBeam;  # long-reinf bars on top layer
set numBarsBotGird $numBarsBotBeam;  # long-reinf bars on bottom layer
set numBarsIntGird $numBarsIntBeam;  # TOTAL number of reinforcing bars on
                                        the intermediate layers
set barAreaTopGird [expr 1./144.];# long-reinf bar area, ft^2
set barAreaBotGird [expr 1./144.];# long-reinf bar area, ft^2
set barAreaIntGird [expr 1./144.];# long-reinf bar area, ft^2

set nfCoreY 15;        # # of fibers in the core patch in the y direction
set nfCoreZ 15;        # # of fibers in the core patch in the z direction
set nfCoverY 15;       # # of fibers in the cover patches with long sides
                            in the y direction
set nfCoverZ 15;       # # of fibers in the cover patches with long sides
                            in the z direction

# rectangular section with one layer of steel evenly distributed around
the perimeter and a confined core.
```

```
# BuildRCrectSection $id $HSec $BSec $coverH $coverB $coreID $coverID
$steelID $numBarsTop $barAreaTop $numBarsBot $barAreaBot $numBarsIntTot
$barAreaInt $nfCoreY $nfCoreZ $nfCoverY $nfCoverZ
BuildRCrectSection $ColSecTagFiber $HCol $BCol $cover $cover $IDconcCore
$IDconcCover $IDSteel $numBarsTopCol $barAreaTopCol $numBarsBotCol
$barAreaBotCol $numBarsIntCol $barAreaIntCol  $nfCoreY $nfCoreZ $nfCoverY
$nfCoverZ

BuildRCrectSection $BeamSecTagFiber $HBeam $BBeam $cover $cover
$IDconcCore  $IDconcCover $IDSteel $numBarsTopBeam $barAreaTopBeam
$numBarsBotBeam $barAreaBotBeam $numBarsIntBeam $barAreaIntBeam  $nfCoreY
$nfCoreZ $nfCoverY $nfCoverZ

BuildRCrectSection $GirdSecTagFiber $HGird $BGird $cover $cover
$IDconcCore  $IDconcCover $IDSteel $numBarsTopGird $barAreaTopGird
$numBarsBotGird $barAreaBotGird $numBarsIntGird $barAreaIntGird  $nfCoreY
$nfCoreZ $nfCoverY $nfCoverZ

# assign torsional Stiffness for 3D Model
uniaxialMaterial Elastic $SecTagTorsion $Ubig
section Aggregator $ColSecTag $SecTagTorsion T -section $ColSecTagFiber
section Aggregator $BeamSecTag $SecTagTorsion T -section $BeamSecTagFiber
section Aggregator $GirdSecTag $SecTagTorsion T -section $GirdSecTagFiber

# Define geometric transformation
set IDColTransfX 1; # columns with long side along global X axis (HCol is
along local y axis)
set IDColTransfZ 2; # columns with long side along global Z axis (HCol is
along local y axis)
set IDBeamTransf 3; # beams (parallel to X axis)
set IDGirdTransf 4; # girders (parallel to Z axis)


geomTransf Corotational $IDColTransfX 0 0 1
geomTransf Corotational $IDColTransfZ 1 0 0
geomTransf Corotational $IDBeamTransf 0 0 1
geomTransf Corotational $IDGirdTransf -1 0 0

# Element connectivity
set elements [open NewModel/elements.out w];
puts $elements "Element Number   Node i   Node j"
set numIntgrPts 5;     #Number of integration points
set Nodei 10102;
set ElemNum 1;

# columns
puts $elements "Columns along global X axis"
for {set k 0} {$k<=1} {incr k 1} {
      for {set i 0} {$i<=$NBayX} {incr i 1} {
            for {set j 1} {$j<=$NStory} {incr j 1} {
                  if { $i==1
                        || $i== [expr $NBayX - 1]
                  } then {
                        set Nodej [expr $Nodei + 100];
                        set Nodei $Nodej;
                    } else {
                        set Nodej [expr $Nodei + 100];
                        element nonlinearBeamColumn $ElemNum $Nodei $Nodej
$numIntgrPts $ColSecTag $IDColTransfZ;
                        puts $elements "$ElemNum $Nodei $Nodej"
                        set Nodei $Nodej;
                        set ElemNum [expr $ElemNum + 1];
                    }
                }
                set Nodei [expr $Nodei - [expr $NStory*100] + 10000];
        }
set Nodei [expr $Nodei + $NBayZ - 2 - $NBayX*10000 - 10000];
}

set Nodei 20101;
puts $elements "Columns along global Z axis"
for {set i 0} {$i<=1} {incr i 1} {
        for {set k 0} {$k<=$NBayZ} {incr k 1} {
                for {set j 1} {$j<=$NStory} {incr j 1} {
                    if { $k==1
                        || $k== [expr $NBayZ - 1]
                    } then {
                        set Nodej [expr $Nodei + 100];
                        set Nodei $Nodej;
                    } else {
                        set Nodej [expr $Nodei + 100];
                        element nonlinearBeamColumn $ElemNum $Nodei $Nodej
$numIntgrPts $ColSecTag $IDColTransfX;
                        puts $elements "$ElemNum $Nodei $Nodej"
                        set Nodei $Nodej;
                        set ElemNum [expr $ElemNum + 1];
                    }
                }
                set Nodei [expr $Nodei - [expr $NStory*100] + 1];
        }
set Nodei [expr $Nodei - $NBayZ - 1 + $NBayX*10000 - 20000];
}

set NColTot [expr $ElemNum - 1];

# Beams (parallel to X axis)
set Nodei 10202
puts $elements "Beams (parallel to X axis)"
for {set j 1} {$j<=$NStory} {incr j 1} {
        for {set k 1} {$k <= 2} {incr k 1} {
                for {set i 1} {$i<=$NBayX} {incr i 1} {
                    set Nodej [expr $Nodei + 10000];
                    element nonlinearBeamColumn $ElemNum $Nodei $Nodej
$numIntgrPts $BeamSecTag $IDBeamTransf;
                    puts $elements "$ElemNum $Nodei $Nodej"
                    set ElemNum [expr $ElemNum + 1];
```

```
                        set Nodei $Nodej;
                  }
            set Nodei [expr $Nodei - $NBayX*10000 + $NBayZ - 2];
            }
      set Nodei [expr $Nodei - $NBayZ*2 + 4 + 100];
      }

      set NBeamsTot [expr $ElemNum - 1 - $NColTot];

      # Girders (parallel to Z axis)
      if {$roof == 1} {
            set level $NStory;      #j == NStory for roof level airplane load, 5
      for midlevel airplane load
      } else {
            set level 5;
      }

      set Nodei 20201
      puts $elements "Girders (parallel to Z axis)"
      for {set j 1} {$j<=$NStory} {incr j 1} {
            for {set i 1} {$i <= 2} {incr i 1} {
                  if {
                    $j == $level
                    && $i == 1
                  } then {
                        for {set k 1} {$k <= [expr $NBayZ+2]} {incr k 1} {
                              if {$k == 4} {
                                    set Nodej $LeftPushLoad;
                                    element nonlinearBeamColumn $ElemNum $Nodei
      $Nodej $numIntgrPts $GirdSecTag $IDGirdTransf;
                                    puts $elements "$ElemNum $Nodei $Nodej"
                                    set ElemNum [expr $ElemNum + 1];
                              } elseif {$k == 5} {
                                    set Nodej [expr $Nodei + 1];
                                    set Nodei $LeftPushLoad;
                                    element nonlinearBeamColumn $ElemNum $Nodei
      $Nodej $numIntgrPts $GirdSecTag $IDGirdTransf;
                                    puts $elements "$ElemNum $Nodei $Nodej"
                                    set ElemNum [expr $ElemNum + 1];
                                    set Nodei $Nodej;
                              } elseif {$k == [expr $NBayZ - 2]} {
                                    set Nodej $RightPushLoad;
                                    element nonlinearBeamColumn $ElemNum $Nodei
      $Nodej $numIntgrPts $GirdSecTag $IDGirdTransf;
                                    puts $elements "$ElemNum $Nodei $Nodej"
                                    set ElemNum [expr $ElemNum + 1];
                                    set Nodej [expr $Nodei + 1];
                                    set Nodei $RightPushLoad;
                              } elseif {$k == [expr $NBayZ-1]} {
                                    element nonlinearBeamColumn $ElemNum $Nodei
      $Nodej $numIntgrPts $GirdSecTag $IDGirdTransf;
                                    puts $elements "$ElemNum $Nodei $Nodej"
                                    set ElemNum [expr $ElemNum + 1];
```

```
                                                set Nodei $Nodej;
                                        } else {
                                                set Nodej [expr $Nodei + 1];
                                                element nonlinearBeamColumn $ElemNum $Nodei
$Nodej $numIntgrPts $GirdSecTag $IDGirdTransf;
                                                puts $elements "$ElemNum $Nodei $Nodej"
                                                set ElemNum [expr $ElemNum + 1];
                                                set Nodei $Nodej;
                                        }
                                }
                        } else {
                                for {set k 1} {$k<=$NBayZ} {incr k 1} {
                                        set Nodej [expr $Nodei + 1];
                                        element nonlinearBeamColumn $ElemNum $Nodei $Nodej
$numIntgrPts $GirdSecTag $IDGirdTransf;
                                        puts $elements "$ElemNum $Nodei $Nodej"
                                        set ElemNum [expr $ElemNum + 1];
                                        set Nodei $Nodej;
                                }
                        }
                set Nodei [expr $Nodei - $NBayZ + $NBayX*10000 - 20000];
                }
        set Nodei [expr $Nodei - $NBayX*20000 + 40000 + 100];
}

set NGirdTot [expr $ElemNum - 1 - $NColTot - $NBeamsTot];

#If have roof grid and not rigid diaphragm at roof
source {C:\Users\Joy\Desktop\MEng\NewModel\RoofGrid.ElementBuild.tcl}

set NElemTot [expr $ElemNum - 1];

set NColFloor [expr $NColTot/$NStory];
set NBeamsFloor [expr $NBeamsTot/$NStory];
set NGirdFloor [expr [expr $NGirdTot - 2]/$NStory];

puts $elements "Total number of elements is $NElemTot"
puts $elements "Total number of columns is $NColTot, per floor is
$NColFloor"
puts $elements "Total number of Beams is $NBeamsTot, per floor is
$NBeamsFloor"
puts $elements "Total number of girders is $NGirdTot, per floor is
$NGirdFloor"

close $elements
puts "Elements built"
```

B.4 Applying gravity load (WaffleNodes.GravityLoad)

```
# Gravity Loads and recorders
# Adapted from Example 7 by Silvia Mazzoni & Frank McKenna, 2006
file mkdir NewModel/SupportReactions;
file mkdir NewModel/LeftWallDisplacements;
file mkdir NewModel/OtherDisplacements;
file mkdir NewModel/OtherDisplacements/Plan
file mkdir NewModel/OtherDisplacements/ElevationX
file mkdir NewModel/OtherDisplacements/ElevationZ
file mkdir NewModel/ElementForces

set girderloads [open NewModel/girderloads.out w]
set BeamLoads [open NewModel/beamloads.out w]
set weight [open NewModel/weight.out w]

# Define RECORDERS -------------------------------------------------------
# displacement of left wall ---------------------------------------------

set NodeN 20201
for {set k 0} {$k <= $NBayZ} {incr k 1} {
if {$k==1
      || $k==[expr $NBayZ - 1]
} then {
set NodeN [expr $NodeN + 1];
} else {
      for {set j 1} {$j <= $NStory} {incr j 1} {
            recorder Node -file
NewModel/LeftWallDisplacements/DFree$NodeN.out -time -node $NodeN -dof 1 2
3 disp;                 # displacements of free node
            set NodeN [expr $NodeN + 100];
      }
set NodeN [expr $NodeN - $NStory*100 + 1];
}
}

#displacement of loaded nodes ----------------------------------------
recorder Node -file NewModel/OtherDisplacements/DFree$LeftPushLoad.out -
time -node $LeftPushLoad -dof 1 2 3 disp;
#recorder Node -file NewModel/DFree$MidPushLoad.out -time -node
$MidPushLoad -dof 1 2 3 disp;
recorder Node -file NewModel/OtherDisplacements/DFree$RightPushLoad.out -
time -node $RightPushLoad -dof 1 2 3 disp;

# plan section -----------------------------------------------------
# displacement of side walls along X axis
set NodeN 10602
for {set k 1} {$k <= 2} {incr k 1} {
      for {set i 0} {$i <= $NBayX} {incr i 1} {
            if {$i==1
                  || $i==[expr $NBayX - 1]
            } then {
                  set NodeN [expr $NodeN + 10000];
            } else {
                  recorder Node -file
NewModel/OtherDisplacements/Plan/DFree$NodeN.out -time -node $NodeN -dof 1
2 3 disp;             # displacements of free node
                  set NodeN [expr $NodeN + 10000];
            }
      }
set NodeN 10614
}
# displacement of rear wall
set NodeN 140601
for {set k 0} {$k <= $NBayZ} {incr k 1} {
      if {$k==1
            || $k==[expr $NBayZ - 1]
      } then {
      } else {
            recorder Node -file
NewModel/OtherDisplacements/Plan/DFree$NodeN.out -time -node $NodeN -dof 1
2 3 disp;             # displacements of free node
      }
      set NodeN [expr $NodeN + 1];
}

# elevation section z axis thr. middle of building -----------------------
# side walls
set NodeN 80202
for {set k 1} {$k <= 2} {incr k 1} {
      for {set j 1} {$j <= $NStory} {incr j 1} {
            recorder Node -file
NewModel/OtherDisplacements/ElevationZ/DFree$NodeN.out -time -node $NodeN
-dof 1 2 3 disp;            # displacements of free node
            set NodeN [expr $NodeN + 100];
      }
set NodeN 80214
}

# roof
set NodeN 81001
for {set k 0} {$k <= $NBayZ} {incr k 1} {
      if {$k==1
            || $k==[expr $NBayZ - 1]
      } then {
            set NodeN [expr $NodeN + 1];
      } else {
            recorder Node -file
NewModel/OtherDisplacements/ElevationZ/DFree$NodeN.out -time -node $NodeN
-dof 1 2 3 disp;            # displacements of free node
            set NodeN [expr $NodeN + 1];
      }
}
# elevation section X axis thr. middle of building -----------------------
# side walls
```

```
set NodeN 140208
for {set j 1} {$j <= $NStory} {incr j 1} {
      recorder Node -file 
NewModel/OtherDisplacements/ElevationX/DFree$NodeN.out -time -node $NodeN 
-dof 1 2 3 disp;              # displacements of free node
      set NodeN [expr $NodeN + 100];
}


# roof
set NodeN 11008
for {set i 0} {$i <= $NBayX} {incr i 1} {
      if {$i==1 
            || $i==[expr $NBayX - 1]
      } then {
            set NodeN [expr $NodeN + 10000];
      } else {
            recorder Node -file 
NewModel/OtherDisplacements/ElevationX/DFree$NodeN.out -time -node $NodeN 
-dof 1 2 3 disp;              # displacements of free node
            set NodeN [expr $NodeN + 10000];
      }
}

# record element forces near airplane load
set ElemN 289;
for {set j 1} {$j<=$NStory} {incr j 1} {
      recorder Element -file NewModel/ElementForces/Force$ElemN.out -time 
-ele $ElemN localForce;                   # local element forces Fx Fy Fz Mx 
My Mz
      set ElemN [expr $ElemN + 1];
}

if {$roof == 0} {
      set ElemN 833;
} elseif {$roof == 1} {
      set ElemN 945;
}

for {set k 1} {$k<= $NBayZ} {incr k 1} {
      recorder Element -file NewModel/ElementForces/Force$ElemN.out -time 
-ele $ElemN localForce;                   # local element forces Fx Fy Fz Mx 
My Mz
      set ElemN [expr $ElemN + 1];
}

# Support Reactions -------------------------------------------------------
#set NodeN 20101
#for {set i 1} {$i <= 2} {incr i 1} {
#for {set k 0} {$k <= $NBayZ} {incr k 1} {
#     if {$k==1 
#           || $k==[expr $NBayZ - 1]
#     } then {
```

```
#           set NodeN [expr $NodeN + 1];
#     } else {
#           recorder Node -file NewModel/SupportReactions/RBase$NodeN.out
-time -node $NodeN   -dof 1 2 3 reaction;          # support reaction
#           set NodeN [expr $NodeN + 1];
#     }
#}
#set NodeN 140101
#}

#set NodeN 10102
#for {set k 1} {$k <= 2} {incr k 1} {
#for {set i 0} {$i <= $NBayX} {incr i 1} {
#     if {$i==1
#           || $k==[expr $NBayX - 1]
#     } then {
#           set NodeN [expr $NodeN + 10000];
#     } else {
#           recorder Node -file NewModel/SupportReactions/RBase$NodeN.out
-time -node $NodeN   -dof 1 2 3 reaction;          # support reaction
#           set NodeN [expr $NodeN + 10000];
#     }
#}
#set NodeN 10114
#}

# Define GRAVITY --------------------------------------------------
# From RectModelNodes, get g = 32.2 ft/s^2, DLFact, LLFact
# Weights:  SDL (psf), LL (psf), ConcreteWeight (pcf), SteelWeight (pcf),
WallWeight
# Configuration:  slabt (ft), wallt (ft), concwallt (ft)
# From RectModelNodes, take HCol, BCol, HBeam, BBeam, HGird, BGird
# From RectModel.ElementBuildTest: NElemTot, NColTot, NBeamsTot, NExtGird,
NIntGird, NColFloor, NBeamsFloor, NExtGirdFloor, NIntGirdFloor

set QBeam [expr $WallWeight*$DLFact +
$ConcreteWeight*$HBeam*$BBeam*$DLFact]; #beam load (lb/ft)
set QGird [expr $WallWeight*$DLFact +
$ConcreteWeight*$HGird*$BGird*$DLFact]; # Exterior girder Load (lb/ft)
set QCol [expr $ConcreteWeight*$HCol*$BCol*$DLFact]; #Column load (lb/ft)

set WeightBeam [expr $QBeam*$LBeam];
set WeightGird [expr $QGird*$LGird];
set WeightCol [expr $QCol*$LCol];

set RoofWeight [expr $NColFloor*$WeightCol/2. + $NBeamsFloor*$WeightBeam +
$NGirdFloor*$WeightGird];
set BottomFloorWeight [expr $NColFloor*$WeightCol/2.];
set FloorWeight [expr $NColFloor*$WeightCol + $NBeamsFloor*$WeightBeam +
$NGirdFloor*$WeightGird];

set TotalWeight [expr $BottomFloorWeight +$RoofWeight + [expr $NStory -
1]*$FloorWeight];
```

```
set TotalMass [expr $TotalWeight/$g];

# define GRAVITY LOADS ----------------------------------------------------

pattern Plain 101 Linear {
#columns
for {set i 1} {$i <= $NColTot} {incr i 1} {
      eleLoad -ele $i -type -beamUniform 0. 0. -$QCol;
}

#Beams
set BeamNumber [expr $NColTot + 1]
for {set i 1} {$i <= $NBeamsTot} {incr i 1} {
      eleLoad -ele $BeamNumber -type -beamUniform -$QBeam 0.;
      puts $BeamLoads "$BeamNumber $QBeam lb/ft"
      set BeamNumber [expr $BeamNumber + 1];
}

#Girders
set GirdNumber $BeamNumber;
for {set i 1} {$i <= $NGirdTot} {incr i 1} {
      eleLoad -ele $GirdNumber -type -beamUniform -$QGird 0.;
      puts $girderloads "$i $QGird lb/ft"
      set GirdNumber [expr $GirdNumber + 1];
}

#if have Roof Grid and not rigid diaphragm
source {C:\Users\Joy\Desktop\MEng\NewModel\RoofGrid.Gravity.tcl}
}

puts $weight "Total Weight is $TotalWeight"
close $weight

# Apply gravity load
# Gravity-analysis parameters -- load-controlled static analysis
set Tol 1.0e-8;              # convergence tolerance for test
variable constraintsTypeGravity Transformation;          # default;
#if {  [info exists RigidDiaphragm] == 1} {
#     if {$RigidDiaphragm=="ON"} {
#           variable constraintsTypeGravity Transformation;     #  large
model: try Transformation
#     };    # if rigid diaphragm is on
#};   # if rigid diaphragm exists
constraints $constraintsTypeGravity ;                # how it handles
boundary conditions
numberer RCM;                # renumber dof's to minimize band-width
(optimization), if you want to
#system BandGeneral; #if elastic
system SparseSPD ;           # how to store and solve the system of
equations in the analysis (large model: try UmfPack)
test EnergyIncr $Tol 6 ;           # determine if convergence has been
achieved at the end of an iteration step
```

```
algorithm Newton;            # use Newton's solution algorithm: updates
tangent stiffness at every iteration
set NstepGravity 10;         # apply gravity in 10 steps
set DGravity [expr 1./$NstepGravity];   # first load increment;
integrator LoadControl $DGravity; # determine the next time step for an
analysis
analysis Static;             # define type of analysis static or transient
analyze $NstepGravity;       # apply gravity


# maintain constant gravity loads and reset time to zero
loadConst -time 0.0
set Tol 1.0e-6;              # reduce tolerance after gravity loads
puts "Model Built"
close $girderloads
close $BeamLoads
```

B.5 Applying pushover load (WaffleNodes.Analysis.PushDisplacementControl)

```
# STATIC PUSHOVER ANALYSIS -----------------------------------------------

set IDctrlNode $MidPushLoad;
      # node where displacement is read for displacement control
set IDctrlDOF 1;
      # degree of freedom of displacement read for displacement contro
set Dmax [expr 0.15*$ShellHeight];
      # maximum displacement of pushover. push to 1% drift.
set Dincr [expr 0.0006*$ShellHeight];
      # displacement increment for pushover.

# analysis parameters
set Tol 1.e-8;           # Convergence Test: tolerance
set maxNumIter 15;       # Convergence Test: maximum number of iterations
that will be performed before "failure to converge" is returned
set printFlag 0;         # Convergence Test: flag used to print
information on convergence (optional)        # 1: print information on
each step;

constraints Transformation;
numberer RCM;
#system BandGeneral; #if elastic
system SparseSPD ;
test EnergyIncr $Tol $maxNumIter $printFlag;
algorithm Newton;      #Linear, Newton, or ModifiedNewton?

integrator DisplacementControl  $IDctrlNode $IDctrlDOF $Dincr
analysis Static

# create load pattern
set Wingload [expr 32022.*1000.]; #lbs
set Cabinload [expr 84423.*1000.]; #lbs
set WingloadFract [expr $Wingload/[expr $Cabinload + 2*$Wingload]];
set CabinloadFract [expr $Cabinload/[expr $Cabinload + 2*$Wingload]];

pattern Plain 200 Linear {;
      load $LeftPushLoad $WingloadFract 0. 0. 0. 0. 0.
      load $MidPushLoad $CabinloadFract 0. 0. 0. 0. 0.
      load $RightPushLoad $WingloadFract 0. 0. 0. 0. 0.
}

# Define DISPLAY ---------------------------------------------------------
set  xPixels 1200;     # height of graphical window in pixels
set  yPixels 800;# height of graphical window in pixels
set  xLoc1 10;   # horizontal location of graphical window (0=upper left-
most corner)
set  yLoc1 10;   # vertical location of graphical window (0=upper left-
most corner)
set ViewScale 1; # scaling factor for viewing deformed shape, it depends
on the dimensions of the model
```

```
DisplayModel3D DeformedShape $ViewScale $xLoc1 $yLoc1  $xPixels $yPixels

set fmt1 "%s Pushover analysis: CtrlNode %.3i, dof %.1i, Disp=%.4f %s";
      # format for screen/file output of DONE/PROBLEM analysis

#  ---------------------------------    perform Static Pushover Analysis
set Nsteps [expr int($Dmax/$Dincr)];        # number of pushover analysis
steps
set ok [analyze $Nsteps];                  # this will return zero if no
convergence problems were encountered

# ---------------------------------- in case of convergence problems
if {$ok != 0} {
      # if analysis fails, we try some other stuff, performance is slower
inside this loop
      set Dstep 0.0;
      set ok 0
      while {$Dstep <= 1.0 && $ok == 0} {
            set controlDisp [nodeDisp $IDctrlNode $IDctrlDOF ]
            set Dstep [expr $controlDisp/$Dmax]
            set ok [analyze 1];                          # this will
return zero if no convergence problems were encountered
            if {$ok != 0} {;                  # reduce step size if still
fails to converge
                  set Nk 4;              # reduce step size
                  set DincrReduced [expr $Dincr/$Nk];
                  integrator DisplacementControl  $IDctrlNode $IDctrlDOF
$DincrReduced
                  for {set ik 1} {$ik <=$Nk} {incr ik 1} {
                        set ok [analyze 1];                          #
this will return zero if no convergence problems were encountered
                        if {$ok != 0} {
                              # if analysis fails, we try some other stuff
                              # performance is slower inside this loop
      global maxNumIterStatic;         # max no. of iterations performed
before "failure to converge" is ret'd
                              puts "Trying Newton with Initial Tangent .."
                              test NormDispIncr   $Tol 2000 0
                              algorithm Newton -initial
                              set ok [analyze 1]
                              test EnergyIncr $Tol $maxNumIter $printFlag;
                              algorithm Newton;      #Linear, Newton, or
ModifiedNewton?
                        }
                        if {$ok != 0} {
                              puts "Trying Broyden .."
                              algorithm Broyden 8
                              set ok [analyze 1 ]
                              algorithm Newton;      #Linear, Newton, or
ModifiedNewton?
                        }
                        if {$ok != 0} {
                              puts "Trying NewtonWithLineSearch .."
```

```
                                        algorithm NewtonLineSearch 0.8
                                        set ok [analyze 1]
                                        algorithm Newton;       #Linear, Newton, or
ModifiedNewton?
                                }
                                if {$ok != 0} {;                        # stop if still
fails to converge
                                        puts [format $fmt1 "PROBLEM" $IDctrlNode
$IDctrlDOF [nodeDisp $IDctrlNode $IDctrlDOF] "inch"]
                                        return -1
                                }; # end if
                        }; # end for
                        integrator DisplacementControl  $IDctrlNode $IDctrlDOF
$Dincr;     # bring back to original increment
                }; # end if
        };     # end while loop
};      # end if ok !0
# ------------------------------------------------------------------------

if {$ok != 0 } {
        puts [format $fmt1 "PROBLEM" $IDctrlNode $IDctrlDOF [nodeDisp
$IDctrlNode $IDctrlDOF] "ft"]
} else {
        puts [format $fmt1 "DONE"  $IDctrlNode $IDctrlDOF [nodeDisp
$IDctrlNode $IDctrlDOF] "ft"]
}
```

B.6 Roof grid scripts

B.6a Defining geometry (RoofGrid.Nodes)

```
#Building Roof Grid - waffle slab structure

# define GEOMETRY -------------------------------------------
# from RectModelNodes file get ShellWidth, ShellLength, ShellHeight
# gridlines are 16 ft apart (approximately 5 m)

set griddist $LBeam;

# define MASSES ---------------------------------------------
# from RectModelNodes get g, LL (psf)
# ConcreteWeight (pcf), SteelWeight (pcf)
# assume SDL = 0;

set gridt $HBeam;
set gridDL [expr $griddist*$griddist*$gridt*$ConcreteWeight];  #lbs
set gridLL [expr $griddist*$griddist*$LL];                      #lbs

set Hgrid $HCol;                             #ft
set Bgrid $BCol;                                                #ft
set gridWeight [expr $griddist*$Hgrid*$Bgrid*$ConcreteWeight]; #lbs

set totalgridWeight [expr $gridWeight + $DLFact*$gridDL +$LLFact*$gridLL];
#lbs
set gridMass [expr $totalgridWeight/$g];
      #lbs/g
puts $masses "Grid Nodal Mass is $gridMass"

# Build GRID ----------------------------------------------
# exterior nodes already defined, need to define center grid nodes
puts $geometry " "
puts $geometry "Roof Grid Nodes"
puts $masses " "
puts $masses "Roof Grid Masses"

set NodeN 11001;
set Z 0.0;
set Y $ShellHeight;

for {set k 0} {$k <= $NBayZ} {incr k 1} {
set X 0.0
      if {$k == 1
            || $k == [expr $NBayZ - 1]
      } then {
            set NodeN [expr $NodeN + $NBayX*10000 + 10000];
            set Z [expr $Z - $griddist];
      } else {
      for {set i 0} {$i <= $NBayX} {incr i 1} {
            if {$i==1
                  || $i==[expr $NBayX - 1]
            } then {
                  set NodeN [expr $NodeN + 10000];
            } else {
                  node $NodeN $X $Y $Z
                  puts $geometry "$NodeN $X $Y $Z"
                  mass $NodeN $gridMass 0. $gridMass 0. 0. 0.
                  puts $masses "$NodeN $gridMass 0. $gridMass"
                  set X [expr $X + $griddist];
                  set NodeN [expr $NodeN + 10000];
            }
      }
      }
set Z [expr $Z + $griddist];
set NodeN [expr $NodeN + 1 - $NBayX*10000 - 10000];
}
```

B.6b Building elements   (RoofGrid.ElementBuild)

```
# Define Elements and Sections for Roof Grid -----------------------------
-------------------

source {C:\Users\Joy\Desktop\MEng\model\BuildRCrectSection.tcl}
# Ubig 1.e10

# Material Parameters ---------------------------------------------
# from ElementBuildTest, get fc [psf], Ec [psf], nu, Gc, confined and
unconfined concrete parameters
# IDconcCore, IDconcCover
# steel: Fy, Es, Bs, IDSteel

#Define Fiber Sections ---------------------------------------------------
--------
set SectionType FiberSection ;

# Section tags
set RoofGridSecTag [expr $GirdSecTagFiber + 1];
set RoofGridSecTagFiber [expr $RoofGridSecTag + 1];
# secTagTorsion 70

#Grid geometry, from before get Hgrid, Bgrid
set cover [expr 2.5/12.];    # rectangular-RC-Column cover, convert from
in to ft
set numBarsTopRoof $numBarsTopCol;      # number of long-reinf bars on top
layer
set numBarsBotRoof $numBarsBotCol;      # number of long-reinf bars on
bottom layer
set numBarsIntRoof $numBarsIntCol;      # TOTAL number of reinforcing bars
on the intermediate layers
set barAreaTopRoof [expr 1./144.];# long-reinf bar area, ft^2
set barAreaBotRoof [expr 1./144.];# long-reinf bar area, ft^2
set barAreaIntRoof [expr 1./144.];# long-reinf bar area, ft^2
```

```
BuildRCrectSection $RoofGridSecTagFiber $Hgrid $Bgrid $cover $cover
$IDconcCore  $IDconcCover $IDSteel $numBarsTopRoof $barAreaTopRoof
$numBarsBotRoof $barAreaBotRoof $numBarsIntRoof $barAreaIntRoof  $nfCoreY
$nfCoreZ $nfCoverY $nfCoverZ
section Aggregator $RoofGridSecTag $SecTagTorsion T -section
$RoofGridSecTagFiber

set IDRoofGridTransfX [expr $IDGirdTransf+1]; # beams parallel to X axis
set IDRoofGridTransfZ [expr $IDRoofGridTransfX + 1]; # beams parallel to Z
axis

geomTransf Corotational $IDRoofGridTransfX 0 0 1
geomTransf Corotational $IDRoofGridTransfZ -1 0 0

puts $elements "Roof Grid Elements"

#Along X Direction
puts $elements "Roof Grid Beams along X Direction"
set Nodei 11001;

for {set k 0} {$k<=$NBayZ} {incr k 1} {
      if {$k==1
            || $k==[expr $NBayZ - 1]
      } then {
      } else {
            for {set i 1} {$i<=$NBayX} {incr i 1} {
                  set Nodej [expr $Nodei + 10000];
                  element nonlinearBeamColumn $ElemNum $Nodei $Nodej
$numIntgrPts $RoofGridSecTag $IDRoofGridTransfX;
                  puts $elements "$ElemNum $Nodei $Nodej"
                  set ElemNum [expr $ElemNum + 1];
                  set Nodei $Nodej;
            }
      }
if {$k==1} {
      set Nodei 11003;
} elseif {$k==[expr $NBayZ-1]} {
      set Nodei [expr $Nodei + 1];
} else {
      set Nodei [expr $Nodei -$NBayX*10000 + 1];
}
}

#Along Z Direction
puts $elements "Roof Grid Beams along Z Direction"
set Nodei 11001;

for {set i 0} {$i<=$NBayX} {incr i 1} {
      if {$i==1
            || $i==[expr $NBayX - 1]
      } then {
      } else {
            for {set k 1} {$k<=$NBayZ} {incr k 1} {
                  set Nodej [expr $Nodei + 1];
                  element nonlinearBeamColumn $ElemNum $Nodei $Nodej
$numIntgrPts $RoofGridSecTag $IDRoofGridTransfZ;
                  puts $elements "$ElemNum $Nodei $Nodej"
                  set ElemNum [expr $ElemNum + 1];
                  set Nodei $Nodej;
            }
      }
if {$i==1} {
      set Nodei 31001;
} elseif {$i==[expr $NBayX-1]} {
      set Nodei [expr $Nodei + 10000];
} else {
      set Nodei [expr $Nodei - $NBayZ + 10000];
}
}

set RoofGridTot [expr $ElemNum - 1 - $NBeamsTot - $NColTot - $NGirdTot];
puts $elements "Total number of roof grid beams is $RoofGridTot"
```

B.6c Applying gravity loads (RoofGrid.Gravity)

```
#Assign Gravity loads for roof grid
set QRoof [expr $totalgridWeight/$griddist];
set RoofGridWeight [expr $totalgridWeight*$RoofGridTot];

set TotalWeight [expr $TotalWeight + $RoofGridWeight];
set TotalMass [expr $TotalWeight/$g];

set RoofGridNumber $GirdNumber;
for {set i 1} {$i <= $RoofGridTot} {incr i 1} {
      eleLoad -ele $RoofGridNumber -type -beamUniform -$QRoof 0.;
      puts $BeamLoads "$RoofGridNumber $QRoof lb/ft"
      set RoofGridNumber [expr $RoofGridNumber + 1];
}
```

B.7 Other scripts

B.7a Eigenvalue analysis (AnalysisOptn_eigen)

```
# AnalysisOptn_5.tcl written by Jeffrey Hunt

# AnalysisOptn "eigen": Type: Eigen
# -------------------------------
# Constraint Handler
constraints  Transformation
# Convergence Test
test  EnergyIncr  +1.000000E-006    25     0     2
# Integrator
integrator  Newmark  +5.000000E-001  +2.500000E-001  +0.000000E+000
+0.000000E+000  +0.000000E+000  +0.000000E+000
# Solution Algorithm
algorithm  Newton
# DOF Numberer
numberer  RCM
# System of Equations
system  SparseSPD
# Analysis Type
analysis  Transient
```

B.7b Building RC section (BuildRCrectSection)

```
proc BuildRCrectSection {id HSec BSec coverH coverB coreID coverID steelID
numBarsTop barAreaTop numBarsBot barAreaBot numBarsIntTot barAreaInt
nfCoreY nfCoreZ nfCoverY nfCoverZ} {

# Build fiber rectangular RC section, 1 steel layer top, 1 bot, 1 skin,
confined core
# Define a procedure which generates a rectangular reinforced concrete
section
# with one layer of steel at the top & bottom, skin reinforcement and a
# confined core.

#           by: Silvia Mazzoni, 2006
#                 adapted from Michael H. Scott, 2003

      set coverY [expr $HSec/2.0];
      set coverZ [expr $BSec/2.0];
      set coreY [expr $coverY-$coverH];
      set coreZ [expr $coverZ-$coverB];
      set numBarsInt [expr $numBarsIntTot/2];

# Define the fiber section
section fiberSec $id {
      # Define the core patch
      patch quadr $coreID $nfCoreZ $nfCoreY -$coreY $coreZ -$coreY -$coreZ
$coreY -$coreZ $coreY $coreZ

      # Define the four cover patches
```

```
	patch quadr $coverID 2 $nfCoverY -$coverY $coverZ -$coreY $coreZ 
$coreY $coreZ $coverY $coverZ
	patch quadr $coverID 2 $nfCoverY -$coreY -$coreZ -$coverY -$coverZ 
$coverY -$coverZ $coreY -$coreZ
	patch quadr $coverID $nfCoverZ 2 -$coverY $coverZ -$coverY -$coverZ 
-$coreY -$coreZ -$coreY $coreZ
	patch quadr $coverID $nfCoverZ 2 $coreY $coreZ $coreY -$coreZ 
$coverY -$coverZ $coverY $coverZ

	# define reinforcing layers
	layer straight $steelID $numBarsInt $barAreaInt  -$coreY $coreZ 
$coreY $coreZ;   # intermediate skin reinf. +z
	layer straight $steelID $numBarsInt $barAreaInt  -$coreY -$coreZ 
$coreY -$coreZ;  # intermediate skin reinf. -z
	layer straight $steelID $numBarsTop $barAreaTop $coreY $coreZ $coreY 
-$coreZ;    # top layer reinfocement
	layer straight $steelID $numBarsBot $barAreaBot  -$coreY $coreZ  -
$coreY -$coreZ;  # bottom layer reinforcement

	};    # end of fibersection definition
};          # end of procedure
```

B.7c Display scripts

DisplayModel3D

```
proc DisplayModel3D { {ShapeType nill} {dAmp 5}  {xLoc 0} {yLoc 0} 
{xPixels 0} {yPixels 0} {nEigen 1} } {

	## display Node Numbers, Deformed or Mode Shape in all 3 planes
	##               Silvia Mazzoni & Frank McKenna, 2006

	global TunitTXT ;                        # load global unit variable
	global ScreenResolutionX ScreenResolutionY;   # read global values 
for screen resolution

if {  [info exists TunitTXT] != 1} {set TunitTXT ""};
if {  [info exists ScreenResolutionX] != 1} {set ScreenResolutionX 1024};
if {  [info exists ScreenResolutionY] != 1} {set ScreenResolutionY 768};

	if {$xPixels == 0} {
		set xPixels [expr int($ScreenResolutionX/2)];
		set yPixels [expr int($ScreenResolutionY/2)]
		set xLoc 10
		set yLoc 10
	}
	if {$ShapeType == "nill"} {
		puts ""; puts ""; puts "------------------"
		puts "View the Model? (N)odes, (D)eformedShape, 
anyMode(1),(2),(#). Press enter for NO."
		gets stdin answer
		if {[llength $answer]>0 } {
```

```
                  if {$answer != "N" & $answer != "n"} {
                        puts "Modify View Scaling Factor=$dAmp? Type
factor, or press enter for NO."
                        gets stdin answerdAmp
                        if {[llength $answerdAmp]>0 } {
                              set dAmp $answerdAmp
                        }
                  }
                  if {[string index $answer 0] == "N" || [string index
$answer 0] == "n"} {
                        set ShapeType NodeNumbers
                  } elseif {[string index $answer 0] == "D" ||[string index
$answer 0] == "d" } {
                        set ShapeType DeformedShape
                  } else {
                        set ShapeType ModeShape
                        set nEigen $answer
                  }
            } else {
                  return
            }
      }

      if {$ShapeType ==  "ModeShape" } {
            set lambdaN [eigen $nEigen];
            set lambda [lindex $lambdaN [expr $nEigen-1]];
            set omega [expr pow($lambda,0.5)]
            set PI      [expr 2*asin(1.0)];          # define constant
            set Tperiod [expr 2*$PI/$omega];         # period
            set fmt1 "Mode Shape, Mode=%.1i Period=%.3f %s  "
            set windowTitle [format $fmt1 $nEigen $Tperiod $TunitTXT ]
      } elseif  {$ShapeType ==  "NodeNumbers" } {
            set windowTitle "Node Numbers"
      } elseif  {$ShapeType ==  "DeformedShape" } {
            set windowTitle0 "Deformed Shape "
      }

      if {$ShapeType ==  "DeformedShape" } {
            set xPixels [expr int($xPixels/2)]
            set yPixels [expr int($yPixels/2)]
            set xLoc1 [expr $xLoc+$xPixels]
            set yLoc1 [expr $yLoc+$yPixels]
            set planeTXT "-Plane"

            set viewPlane XY
            set windowTitle $windowTitle0$viewPlane$planeTXT
      recorder display $windowTitle $xLoc1 $yLoc $xPixels $yPixels  -wipe
; # display recorder
            DisplayPlane $ShapeType $dAmp $viewPlane
            set viewPlane ZY
            set windowTitle $windowTitle0$viewPlane$planeTXT
            recorder display $windowTitle $xLoc $yLoc $xPixels $yPixels  -
wipe ; # display recorder
```

```
           DisplayPlane $ShapeType $dAmp $viewPlane
           set viewPlane ZX
           set windowTitle $windowTitle0$viewPlane$planeTXT
           recorder display $windowTitle $xLoc $yLoc1 $xPixels $yPixels
-wipe ; # display recorder
           DisplayPlane $ShapeType $dAmp $viewPlane
           set viewPlane 3D
           set windowTitle $windowTitle0$viewPlane
           recorder display $windowTitle $xLoc1 $yLoc1 $xPixels $yPixels
-wipe ; # display recorder
           DisplayPlane $ShapeType $dAmp $viewPlane
      } else {
           recorder display $windowTitle $xLoc $yLoc $xPixels $yPixels -
nowipe; # display recorder
           set viewPlane XY
           DisplayPlane $ShapeType $dAmp $viewPlane $nEigen 1
           set viewPlane ZY
           DisplayPlane $ShapeType $dAmp $viewPlane $nEigen 2
           set viewPlane ZX
           DisplayPlane $ShapeType $dAmp $viewPlane $nEigen 3
           set viewPlane 3D
           DisplayPlane $ShapeType $dAmp $viewPlane $nEigen 4
      }
}
```

DisplayPlane

```
proc DisplayPlane {ShapeType dAmp viewPlane {nEigen 0}  {quadrant 0}} {
      ## setup display parameters for specified viewPlane and display
      ##                 Silvia Mazzoni & Frank McKenna, 2006

      set Xmin [lindex [nodeBounds] 0];
      set Ymin [lindex [nodeBounds] 1];
      set Zmin [lindex [nodeBounds] 2];
      set Xmax [lindex [nodeBounds] 3];
      set Ymax [lindex [nodeBounds] 4];
      set Zmax [lindex [nodeBounds] 5];

      set Xo 0;
      set Yo 0;
      set Zo 0;

      set uLocal [string index $viewPlane 0];
      set vLocal [string index $viewPlane 1];

      if  {$viewPlane =="3D" } {
           set uMin $Zmin+$Xmin
           set uMax $Zmax+$Xmax
           set vMin $Ymin
           set vMax $Ymax
           set wMin -10000
           set wMax 10000
           vup 0 1 0; # dirn defining up direction of view plane
```

```
} else {
      set keyAxisMin "X $Xmin Y $Ymin Z $Zmin"
      set keyAxisMax "X $Xmax Y $Ymax Z $Zmax"
      set axisU [string index $viewPlane 0];
      set axisV [string index $viewPlane 1];
      set uMin [string map $keyAxisMin $axisU]
      set uMax [string map $keyAxisMax $axisU]
      set vMin [string map $keyAxisMin $axisV]
      set vMax [string map $keyAxisMax $axisV]
      if {$viewPlane =="YZ" || $viewPlane =="ZY" } {
            set wMin $Xmin
            set wMax $Xmax
      } elseif  {$viewPlane =="XY" || $viewPlane =="YX" } {
            set wMin $Zmin
            set wMax $Zmax
      } elseif  {$viewPlane =="XZ" || $viewPlane =="ZX" } {
            set wMin $Ymin
            set wMax $Ymax
      } else {
      return -1
      }
}

set epsilon 1e-3;

set uWide [expr $uMax - $uMin+$epsilon];
set vWide [expr $vMax - $vMin+$epsilon];
set uSide [expr 0.25*$uWide];
set vSide [expr 0.25*$vWide];
set uMin [expr $uMin - $uSide];
set uMax [expr $uMax + $uSide];
set vMin [expr $vMin - $vSide];
set vMax [expr $vMax + 2*$vSide];
set uWide [expr $uMax - $uMin+$epsilon];
set vWide [expr $vMax - $vMin+$epsilon];
set uMid [expr ($uMin+$uMax)/2];
set vMid [expr ($vMin+$vMax)/2];

vrp $Xo $Yo $Zo;
if {$vLocal == "X"} {
      vup 1 0 0; # dirn defining up direction of view plane
} elseif {$vLocal == "Y"} {
      vup 0 1 0; # dirn defining up direction of view plane
} elseif {$vLocal == "Z"} {
      vup 0 0 1; # dirn defining up direction of view plane
}
if {$viewPlane =="YZ" } {
      vpn 1 0 0; # direction of outward normal to view plane
      prp 10000. $uMid $vMid ;
      plane 10000 -10000;
} elseif  {$viewPlane =="ZY" } {
      vpn -1 0 0; # direction of outward normal to view plane
      prp -10000. $vMid $uMid ;
```

```
                plane 10000 -10000;
        } elseif  {$viewPlane =="XY"  } {
                vpn 0 0 1; # direction of outward normal to view plane
                prp $uMid $vMid 10000;
                plane 10000 -10000;
        } elseif  {$viewPlane =="YX" } {
                vpn 0 0 -1; # direction of outward normal to view plane
                prp $uMid $vMid -10000;
                plane 10000 -10000;
        } elseif  {$viewPlane =="XZ" } {
                vpn 0 -1 0; # direction of outward normal to view plane
                prp $uMid -10000 $vMid ;
                plane 10000 -10000;
        } elseif  {$viewPlane =="ZX" } {
                vpn 0 1 0; # direction of outward normal to view plane
                prp $uMid 10000 $vMid ;
                plane 10000 -10000;
        } elseif  {$viewPlane =="3D" } {
                vpn 1 0.25 1.25; # direction of outward normal to view plane
                prp -100 $vMid 10000;
                plane 10000 -10000;
        }  else {
                return -1
        }
        if  {$viewPlane =="3D" } {
                viewWindow [expr $uMin-$uWide/4] [expr $uMax/2] [expr $vMin-
0.25*$vWide] [expr $vMax]
        } else {
                viewWindow $uMin $uMax $vMin $vMax
        }
        projection 1;    # projection mode, 0:prespective, 1: parallel
        fill 1;          # fill mode; needed only for solid elements

        if {$quadrant == 0} {
                port -1 1 -1 1
        } elseif {$quadrant == 1} {
                port 0 1 0 1
        } elseif {$quadrant == 2} {
                port -1 0 0 1
        } elseif {$quadrant == 3} {
                port -1 0 -1 0
        } elseif {$quadrant == 4} {
                port 0 1 -1 0
        }

        if {$ShapeType ==  "ModeShape" } {
                display -$nEigen 0  [expr 5.*$dAmp];
        } elseif  {$ShapeType ==  "NodeNumbers" } {
                display 1 -1 0  ;
        } elseif  {$ShapeType ==  "DeformedShape" }  {
                display 1 2 $dAmp;
        }
};
```

Appendix C. Aircraft Crash Impact Loading Determination

This load determination was performed by Eric Keldrauk (2009) and is based off of rescaling data from Riera (1968). The Riera data was determined for a Boeing 707-320 and rescaled for both a Boeing 747-400 and a Boeing 737-900 to take into account differences in mass and velocity. The resultant forces for the cabin and the aircraft wings are adjusted for relative impulse areas. Thus, the load used for the cabin is 84,400 kips while the load for each wing is 32000 kips.

Table C.1 Airplane impact parameters (Keldrauk)

		Riera 1968	GNEP 2009	
Aircraft Type		Boeing 707-320	Boeing 747-400	Boeing 737-900
Aircraft Length (feet)		144.5	231.3	138.2
Maximum Weight (kips)		222	875	188
Impact Velocity (feet/sec)		338	660	590
Time Scaling Factor, X_t		1	1.037	0.691
Force Scaling Factor, X_f		1	7.423	2.134
Force Linearization Parameters based on the Riera Linearization (1968)	t_1 (sec)	0.030	0.031	0.021
	t_2 (sec)	0.165	0.171	0.114
	t_3 (sec)	0.220	0.228	0.152
	t_4 (sec)	0.275	0.285	0.190
	t_5 (sec)	0.280	0.290	0.194
	t_6 (sec)	0.330	0.342	0.228
	F_1 (kips)	4000	29694	8538
	F_2 (kips)	20000	148468	42688

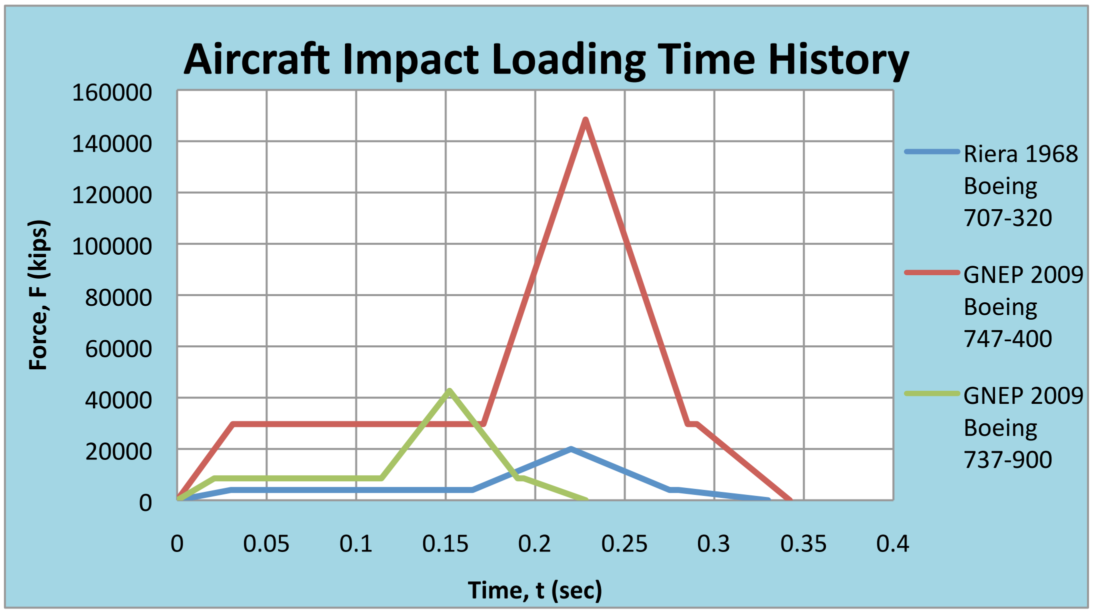

Figure C.1 Aircraft Impact Loading Time History (Keldrauk)