APPENDIX G:
HMLT MODAL FREQUENCY ALGORITHM

MULTIPLE DEGREE-OF-FREEDOM EIGENVALUE SOLUTION

Natural frequencies and mode shapes for HMLTs would be necessary to solve any advanced methods for wind-induced loads and displacements that account for dynamic effects. Rather than creating a finite element model, a simpler multiple-degree-of-freedom model can be solved using matrix methods. For this study, a MATLAB program was written to aid in the analysis, and provided a reasonably close approximation to values obtained experimentally. This method was chosen for three reasons:

1. The relative ease with which an eigenvalue problem could be used to solve a multiple degree-of-freedom system,
2. The time needed to create a finite element model for each of the HMLTs studied,
3. The ability to distribute such a program for other's use on future HMLT design is easily done as the MATLAB script can be exported as a stand-alone executable file (for example, it could be provided to AASHTOWare).

The program begins by prompting the user for geometrical input for the HMLT of interest: shape of pole, top and bottom pole diameters, number of tubular sections, thickness of each section, length of section overlap, and weight of luminary. Next, the user is prompted for the number of elements to discretize the model. Then, stiffness and mass quantities are calculated for each segment and the stiffness matrix k and mass matrix m are assembled. The eigenvalue problem:

$$[k - \omega_n^2 m]\phi_n = 0$$

is then solved for natural frequencies ω_n, and mode shape vectors ϕ_n. Natural frequencies for the first four modes of vibration, as calculated by the program, are shown in Table G-1 along with the experimental values for comparison. Each of the HMLTs in the table was modeled using discrete elements approximately ten feet in length. The average error for the values in the table is 4.5%, indicating good correlation. Mode shapes for the first four modes of vibration of the South Dakota site are shown in Figure G-1 as an example.

Mode	CA		IA		KS		ND		SD	
	EXP	EVS	EXP	EVS	EXP	EVS	EXP	EVS	EXP	EVS
1	0.37	0.37	0.32	0.34	0.36	0.33	0.31	0.31	0.29	0.30
2	1.73	1.79	1.33	1.35	1.66	1.62	1.28	1.23	1.17	1.17
3	4.72	4.97	3.41	3.39	4.45	4.49	3.15	3.04	2.98	2.94
4	9.45	9.94	6.53	6.56	9.21	9.00	5.94	5.81	5.81	5.72
Mode	**OK-NE**		**OK-SW**		**PA**		**WY-CJE**		**WY-CJW**	
	EXP	EVS	EXP	EVS	EXP	EVS	EXP	EVS	EXP	EVS
1	0.26	0.29	0.26	0.29	0.39	0.40	0.35	0.39	0.35	0.39
2	1.03	1.12	1.20	1.27	1.68	1.67	1.50	1.53	1.53	1.53
3	2.83	3.02	3.25	3.46	4.60	4.34	3.85	3.85	3.89	3.85
4	5.15	5.74	6.91	6.61	9.37	8.52	7.54	7.92	7.56	7.92

Table G-1: Comparison of experimental modal frequencies (EXP) with eigenvalue problem solution (EVS)

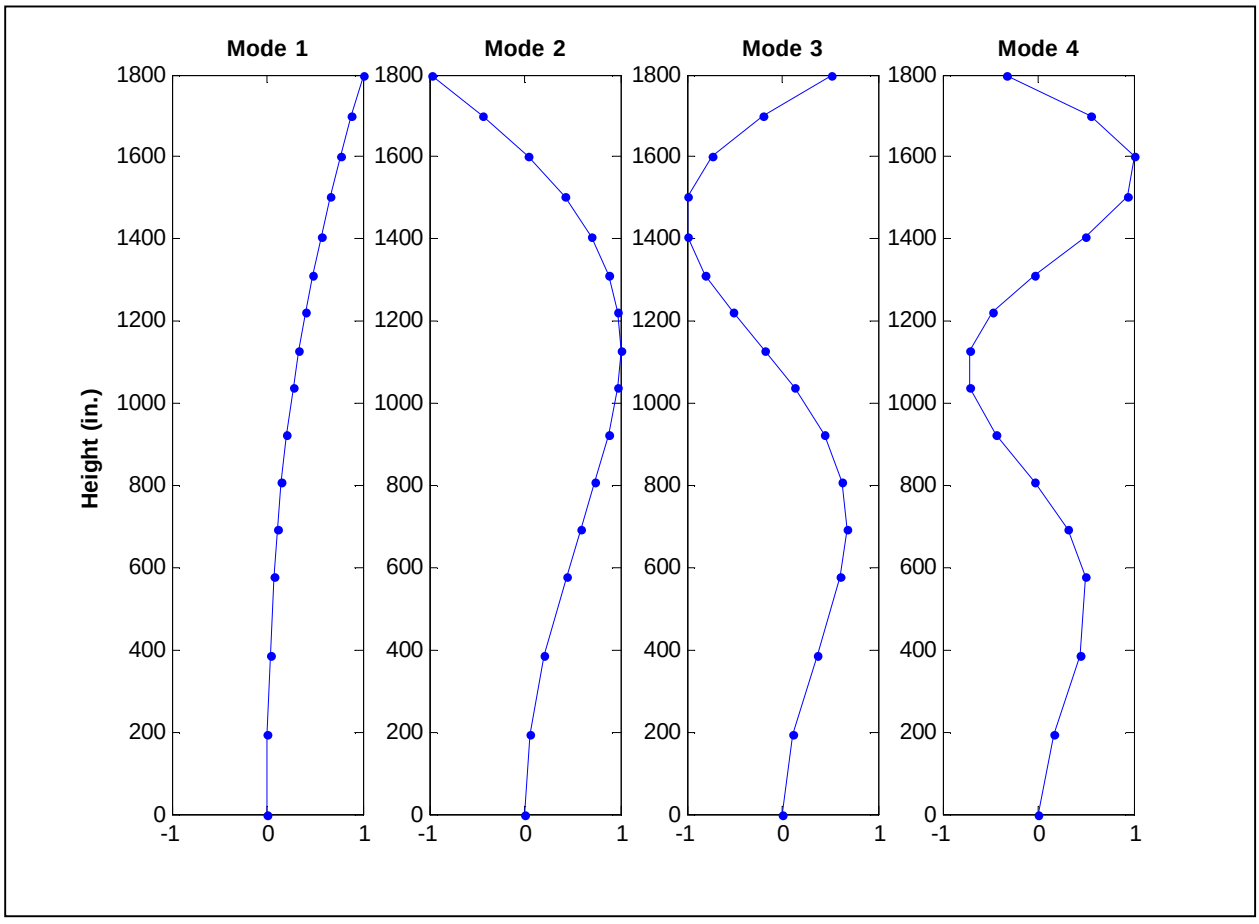

Figure G-1: First four mode shapes of SD HMLT calculated by eigenvalue problem solution (EVS)

MATALAB CODE

```
% EV_SOLN - Solution of natural frequencies and modeshapes of a dicretized
% system using the eigenvalue problem.

clear all; close all;

go = struct('a',[],'b',[]);
while 1
    go.a = double(input...
        ('Load e(x)isting HMLT data or enter (n)ew? x/n <x>:  ','s'));
    if isempty(go.a), go.a = 120; end
    if go.a == 110 || go.a == 120, break; end
end

switch go.a
    case 110
        hmlt = HMLT_data(struct([]));
        go.b = 1;
        fname_r = [];
    case 120
        [fname_r path_r] = uigetfile('*.mat','HMLT data to load...');
        load([path_r fname_r]);
        go.b = 0;
        while 1
            go.a = double(input...
                ('Edit HMLT data? y/n <n>:  ','s'));
            if isempty(go.a), go.a = 110; end
            if go.a == 110 || go.a == 121, break; end
        end
        switch go.a
            case 121
                hmlt = HMLT_data(hmlt);
                go.b = 1;
        end
end

E = 29000000; %psi
g = 386.4; %in/s^2
rho = 490/1728/g; %lbf*s^2/in^4
mL = hmlt.WL/g;

while 1
    N = input('\nNumber of elements in HMLT:  ');
    if N == floor(N) && N >= hmlt.NS,
        break,
    else
        fprintf('\nPlease enter integer value >= # of sections!');
    end
end

%Position vector
%Lm = mass location;    Lk = stiffness location
Nper = floor(N/hmlt.NS);
R = N - hmlt.NS*Nper;
```

```
Nper = Nper*ones(hmlt.NS,1);
for ii = 1:R
    Nper(ii) = Nper(ii) + 1;
end
Lsec = zeros(hmlt.NS+1,1);
for ii = 1:hmlt.NS
    Lsec(ii+1) = Lsec(ii) +...
        hmlt.SL(ii,1) - hmlt.SL(ii,2)/2 - hmlt.SL(ii+1,2)/2;
end
Lm = zeros(N+1,1);
Lk = zeros(N,1);
ii = 2;
for jj = 1:hmlt.NS
    kk = 1;
    while kk <= Nper(jj)
        Lm(ii) = Lm(ii-1) + (Lsec(jj+1) - Lsec(jj))/Nper(jj);
        ii = ii + 1;
        kk = kk + 1;
    end
end
Lm = 12*Lm; %in.
L = Lm(ii-1);
for ii = 1:N
    Lk(ii) = (Lm(ii) + Lm(ii+1))/2;
end

%Diameter values for each element
%Dm = mass diameter;    Dk = stiffness diameter
Dm = zeros(N,1);
Dm(1) = hmlt.Dt;
Dk = zeros(N,1);
Dk(1) = hmlt.Dt + (hmlt.Db - hmlt.Dt)*Lk(1)/L;
for ii = 2:N
    Dm(ii) = Dm(ii-1) + (hmlt.Db - hmlt.Dt)*(Lm(ii) - Lm(ii-1))/L;
    Dk(ii) = Dk(ii-1) + (hmlt.Db - hmlt.Dt)*(Lk(ii) - Lk(ii-1))/L;
end
Dm(1) = Dm(1) + 0.25*(Dm(2) - Dm(1));

%Thickness values for each element
%Tm = mass thickness;   Tk = stiffness thickness
S = size(Lm,1);
Tm = zeros(N,1);
Tk = zeros(N,1);
for ii = 1:hmlt.NS
    Tm((12*Lsec(ii) <= Lm(1:S-1)) & (Lm(1:S-1) < 12*Lsec(ii+1))) =...
        hmlt.Th(ii);
    Tk((12*Lsec(ii) <= Lk) & (Lk < 12*Lsec(ii+1))) = hmlt.Th(ii);
end

%Diameter measured to mid-thickness of wall
Dm = Dm - Tm;
Dk = Dk - Tk;

%Element section properties
switch hmlt.Sh
    case 1  %Round
```

```
        Area = pi*Dm.*Tm;
        momI = (pi/8)*Dk.^3.*Tk;
    case 2  %12 Sides
        Area = (6.43/2)*Dm.*Tm;
        momI = (3.29/8)*Dk.^3.*Tk;
    case 3  %16 Sides
        Area = (6.37/2)*Dm.*Tm;
        momI = (3.22/8)*Dk.^3.*Tk;
end

%Mass Matrix
M = zeros(N);
M(1,1) = 0.5*Area(1)*(Lm(2)-Lm(1))*rho + mL;
jj = 2;
for ii = 2:N
    if Lm(ii) == 12*Lsec(jj)
        M(ii,ii) =...
            (Area(ii)*(Lk(ii)-Lk(ii-1)) + Area(ii)*12*hmlt.SL(jj,2))*rho;
        %Not exact, but OK
        jj = jj + 1;
    else
        M(ii,ii) = Area(ii)*(Lk(ii)-Lk(ii-1))*rho;
    end
end

%Stiffness Matrix
if isempty(hmlt.Kb)
    K = K_stiff(N,E,momI,Lm);
else
    K = K_stiff_spr(N,E,momI,Lm,hmlt.Kb);
end

%Problem Solution
[V,D] = eig(K,M);
D = D*ones(N,1);
[~,I] = sort(D);
fn = sort(D.^0.5/2/pi);
phi = zeros(N);
for ii = 1:N
    phi(:,ii) = V(:,I(ii));
end

%Print first 4 modal frequencies
fprintf('\n Mode\t   fn');
fprintf('\n-----\t-----');
for ii = 1:4
    fprintf('\n   %2d\t%5.2f',[ii fn(ii)]);
end
fprintf('\n');

%Plot first 4 mode shapes
for ii = 1:4
    subplot(1,4,ii)
    plot([phi(:,ii); 0],abs(Lm - max(Lm)),'-b.')
    title(['Mode ' num2str(ii)]);
end
```

```
%Save HMLT data?
while go.b
    go.a = double(input...
        ('Save HMLT data? y/n <n>:  ','s'));
    if isempty(go.a), go.a = 110; end
    switch go.a
        case 121
            while 1
                if fname_r
                    fname_r = strrep(fname_r,'.mat','');
                    fname = input...
                        (['File name for HMLT data (*.MAT) <',fname_r,'>:
'],'s');
                    if isempty(fname), fname = fname_r;
                    end
                else
                    fname = input...
                        ('File name for HMLT data (*.MAT):  ','s');
                end
                if fname , break; end
            end
            save(fname,'hmlt');
    end
    if go.a == 110 || go.a == 121, break; end
end

%Save dynamic data?
while 1
    go.a = double(input('\nSave dynamic data? y/n <n>:  ','s'));
    if isempty(go.a), go.a = 110; end
    if go.a == 110 || go.a == 121, break; end
end
if go.a == 121,
    DYN_data(   fn,...
                phi,...
                M,...
                Dm,...
                Tm,...
                abs(Lm - max(Lm)),...
                hmlt    );
end
```

```
% HMLT_DATA - Function for entering HMLT data for dynamic analysis
% Subroutine to EV_soln

function [ hmlt ] = HMLT_data( hmlt )

mpt = 0;
if isempty(hmlt)
    hmlt = struct('NS',[],'SL',[],'Sh',[],'Dt',[],...
                  'Db',[],'Th',[],'WL',[],'Kb',[]);
    mpt = 1;
end

fprintf('\n');
fprintf('\t[1]\tRound\n');
fprintf('\t[2]\t12 Sides\n');
fprintf('\t[3]\t16 Sides\n');
while 1
    tmp = input(['\nShape of HMLT <',num2str(hmlt.Sh),'>:  ']);
    if tmp, hmlt.Sh = tmp; end
    if (hmlt.Sh >= 1)&&(hmlt.Sh <= 3), break; end
end

while 1
    tmp = input(['Diameter (across flats) at top (in.) <',...
        num2str(hmlt.Dt),'>:  ']);
    if tmp, hmlt.Dt = tmp; end
    tmp = input(['Diameter (across flats) at bottom (in.) <',...
        num2str(hmlt.Db),'>:  ']);
    if tmp, hmlt.Db = tmp; end
    if hmlt.Dt > 0 && hmlt.Db > 0, break; end
end

fprintf('\nNOTE: enter HMLT section data from the ground up.\n');
while 1
    tmp = input(['Number of HMLT sections? <',num2str(hmlt.NS),'>:  ']);
    if tmp, hmlt.NS = tmp; end
    if hmlt.NS > 0 && hmlt.NS == floor(hmlt.NS), break; end
end

if mpt
    hmlt.SL = zeros(hmlt.NS+1,2);
    hmlt.Th = zeros(hmlt.NS,1);
else
    del = hmlt.NS+1 - size(hmlt.SL,1);
    if del > 0
        hmlt.SL = [hmlt.SL; zeros(del,2)];
        hmlt.Th = [hmlt.Th; zeros(del,1)];
    elseif del < 0
        hmlt.SL(hmlt.NS:size(hmlt.SL,1),:) = [];
        hmlt.Th(hmlt.NS:size(hmlt.Th,1),:) = [];
    end
end

jj = hmlt.NS: -1: 1;
while 1
```

```
    for ii = 1:hmlt.NS
        if mpt
            str = {['Length of section ' num2str(ii) ' (ft):  '];
                   ['Section ' num2str(ii) ' overlap (ft):  '];
                   ['Thickness of section ' num2str(ii) ' (in.):  ']};
        else
            str = {['Length of section ' num2str(ii) ' (ft) <'...
                    num2str(hmlt.SL(jj(ii),1)) '>:  '];
                   ['Section ' num2str(ii) ' overlap (ft) <'...
                    num2str(hmlt.SL(jj(ii)+1,2)) '>:  '];
                   ['Thickness of section ' num2str(ii) ' (in.) <',...
                    num2str(hmlt.Th(jj(ii))),'>:  ']};
        end
        tmp = input(str{1});
        if tmp, hmlt.SL(jj(ii),1) = tmp; end
        if ii > 1,
            tmp = input(str{2});
            if tmp, hmlt.SL(jj(ii)+1,2) = tmp; end
        end
        tmp = input(str{3});
        if tmp, hmlt.Th(jj(ii)) = tmp; end
    end
    mpt = 0;
    L = sum(hmlt.SL);   L = L(1)-L(2);
    while 1
        go = double(input(['Length of HMLT is ' num2str(L)...
            ' feet.  Is this correct? y/n <n>:  '],'s'));
        if isempty(go), go = 110; end
        if go(1) == 110 || go(1) == 121, break; end
    end
    if go(1) == 121, break, end
end

while 1
    tmp = input(['Weight of luminaire (lbs) <',num2str(hmlt.WL),'>:  ']);
    if tmp >= 0, hmlt.WL = tmp; end
    if hmlt.WL >= 0, break, end
end

fprintf('\n');
fprintf('\t[1]\tRigid\n');
fprintf('\t[2]\tFlexible\n');
while 1
    if isempty(hmlt.Kb), BC = 1; else BC = 2; end
    tmp = input(['\nHMLT base support condition <',num2str(BC),'>:  ']);
    if tmp, BC = tmp; end
    if (BC >= 1)&&(BC <= 2), break; end
end
if BC == 2
    while 1
        tmp = input(['Rotational stiffness of base (k-in) <',...
            num2str(hmlt.Kb),'>:  ']);
        if tmp, hmlt.Kb = tmp; end
        if hmlt.Kb > 0, break; end
    end
else
```

```
    hmlt.Kb = [];
end

end
```

```
% K_STIFF - function for calculating global stiffness matrix for cantilever

function [Kc] = K_stiff(N,E,Iv,Lv)
% N = number of elements
% Iv = vector describing moment of inertia of each node
% Lv = vector describing position of each node

K = zeros(2*N+2);

X = 1:2:2*N-1;      %Displacement dofs
Y = 2:2:2*N;        %Rotation dofs
for ii = 1:N
    %Create element stiffness matrix
    I = Iv(ii);
    L = Lv(ii+1) - Lv(ii);
    k_el = (E*I/L^3)*[ 12    6*L   -12    6*L;
                      6*L  4*L^2  -6*L  2*L^2;
                      -12   -6*L    12   -6*L;
                      6*L  2*L^2  -6*L  4*L^2];
    %Assemble into global stiffness matrix
    pp = 2*ii-1:2*ii+2;
    K(pp,pp) = K(pp,pp) + k_el;
end

%Static condensation
%Schur complement
Kc = K(X,X) - K(X,Y)/K(Y,Y)*K(Y,X);

end
```

```
% K_STIFF_SPR - function for calculating global stiffness matrix for
% cantilever including base stiffness

function [Kc] = K_stiff_spr(N,E,Iv,Lv,Spr)
% N = number of elements
% Iv = vector describing moment of inertia of each node
% Lv = vector describing position of each node
% Spr = base stiffness

%Create base stiffness matrix
k_base = [ Spr -Spr;
          -Spr  Spr];

K = zeros(2*N+3);
K([2*N+2 2*N+3],[2*N+2 2*N+3]) = k_base;

X = 1:2:2*N-1;      %Displacement dofs
Y = 2:2:2*N+2;        %Rotation dofs
for ii = 1:N
    %Create element stiffness matrix
    I = Iv(ii);
    L = Lv(ii+1) - Lv(ii);
    k_el = (E*I/L^3)*[ 12    6*L   -12    6*L;
                       6*L  4*L^2  -6*L  2*L^2;
                       -12   -6*L    12   -6*L;
                       6*L  2*L^2  -6*L  4*L^2];
    %Assemble into global stiffness matrix
    pp = 2*ii-1:2*ii+2;
    K(pp,pp) = K(pp,pp) + k_el;
end

%Static condensation
%Schur complement
Kc = K(X,X) - K(X,Y)/K(Y,Y)*K(Y,X);

end
```

```
%DYN_DATA - save function for output from EV_soln

function [] = DYN_data(fn,phi,Mf,Df,Tf,Lf,hmlt)

fname = input('File name for dynamic data (*.MAT):  ','s');

[R,C] = size(phi);

Df = [(Df + Tf); hmlt.Db];
Tf = [Tf; hmlt.Th(size(hmlt.Th,1))];
Sh = hmlt.Sh;

%Readjust top diameter
Df(1) = hmlt.Dt;

%Add placeholders
phi = [phi; zeros(1,C)];
Mf = Mf*ones(R,1);
Mf = [Mf; 0];

%Change to bottom up convention
rev = R+1:-1:1;
phi = phi(rev,:);
Mf = Mf(rev);
Df = Df(rev);
Lf = Lf(rev);
Tf = Tf(rev);

%Moments of inertia
switch hmlt.Sh
    case 1  %Round
        If = (pi/8) * (Df - Tf).^3.*Tf;
    case 2  %12 Sides
        If = (3.29/8)*(Df - Tf).^3.*Tf;
    case 3  %16 Sides
        If = (3.22/8)*(Df - Tf).^3.*Tf;
end

save(fname,'fn','phi','Mf','Df','Lf','If','Sh');

end
```