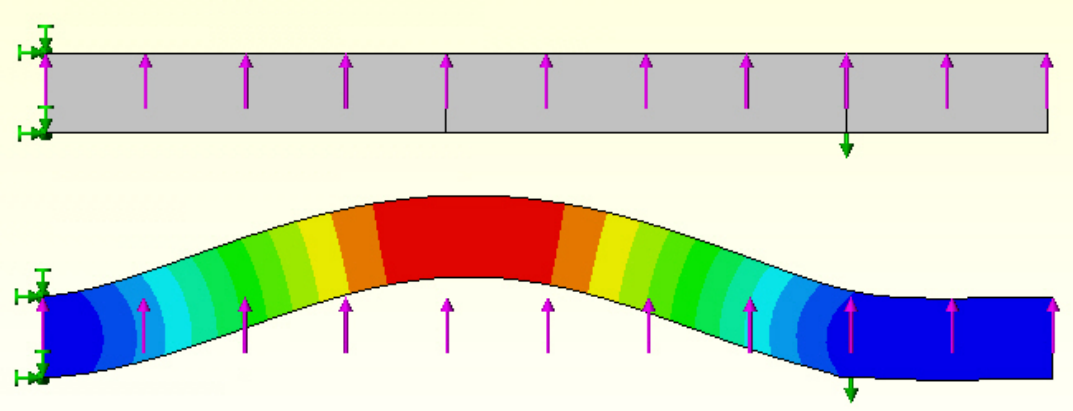
Matlab Classic Beam FEA Solution

(Draft 2, 2/14/08)

Introduction

Here the goal is to solve a staticly indeterminant propped cantilever beab subjected to a unifor, load. The distance between supports is L while that of the overhang is L/4. The bending stiffness has been taken as EI = 1.



The above figure, from CosmosWorks, shows the beam fixxed on the left end and a vertical roller support at L. The expected deflected shape is also shown. This problem will be solved with the modular script listed at the end. (The functions are listed alphabetically and include those heat transfer, plane stress, etc.

Hermite cubic beam element:

The element here is a cubic line element with nodal values of the transverse deflection and its slope (an L2_C1 element), thus it has the C1 continuity required by the 4-th order beam differential equation. The matrices are well known and written in closed form. (A numerically intergrated version will also be posted.)

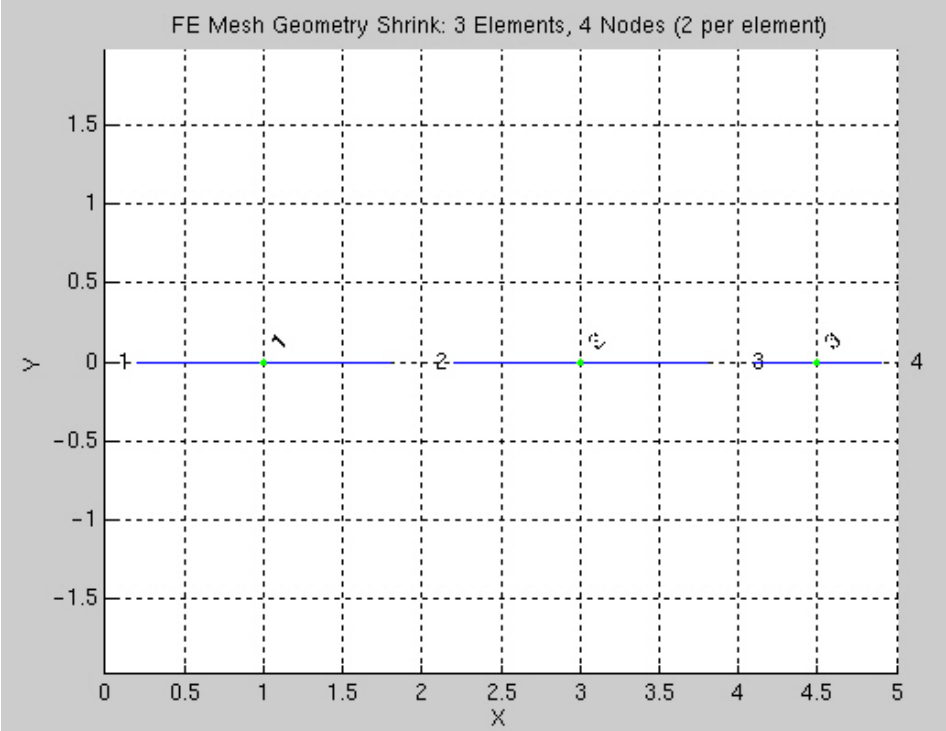
Starting in Unix, Matlab is invoked:

```
>> Modular_Beam_X
Read 4 nodes with bc_flag & 1 coordinates.
[echo of file msh_bc_xyz.tmp]
    11     0
     0     2
    10     4
     0     5
```

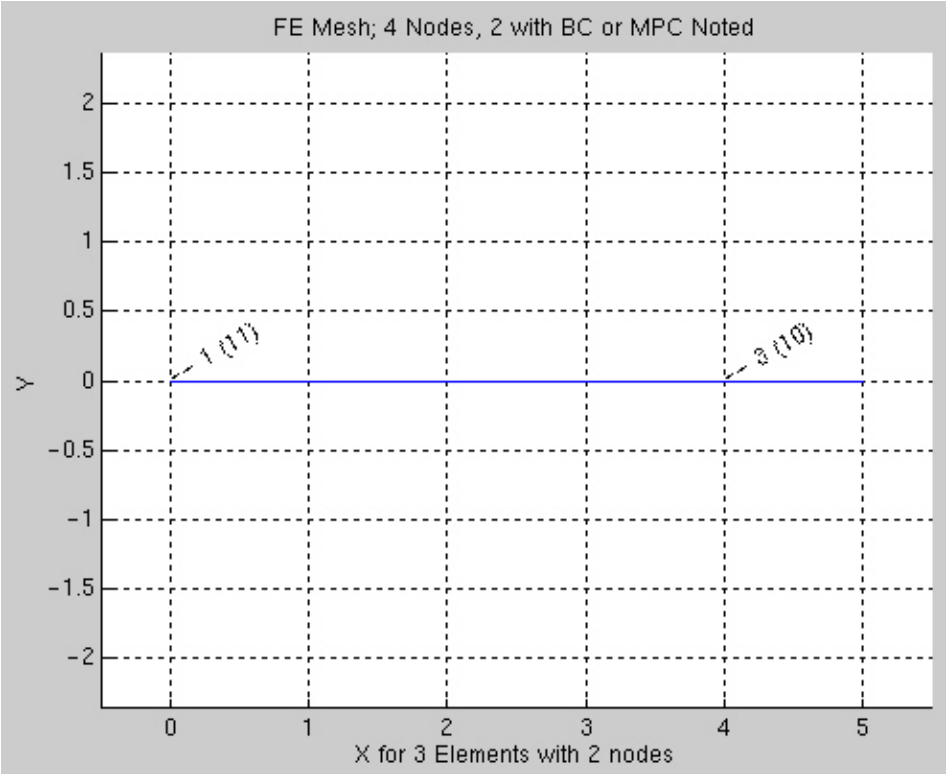
```
Read 3 elements with type number & 2 nodes each.
Maximum number of element types = 1.
[echo of file msh_typ_nodes.tmp]
     1     1     2
     1     2     3
     1     3     4

Note: expecting 3 EBC values.
Read 3 EBC data with Node, DOF, Value.
[echo of file msh_ebc.tmp]
     1     1     0
     1     2     0
     3     1     0

Application properties are:
Elastity modulus = 1
Moment of inertia = 1
Line Load = [ 1 1 ]

Applied Loads Summary:
Node, DOF, Resultant Force (1) or Moment (2)
1 1 1
1 2 0.333333
2 1 2
3 1 1.5
3 2 -0.25
4 1 0.5
4 2 -0.0833333
Totals =     5.0000    0.0000

Computed Solution:
Node  Y_displacement  Z_rotation at 4 nodes
1     0               0
2     1.08333         0.208333
3     0              -0.833333
4    -0.708333       -0.666667

Computed Reactions:Node, DOF, Value for 3 Reactions
1 1 -2.3125
1 2 -1.75
3 1 -2.6875
Totals =    -5.0000   -1.7500

>> % use plotting scripts
>> addpath /net/course-a/mech517/public_html/Matlab_Plots
```

>> mesh_shrink_plot

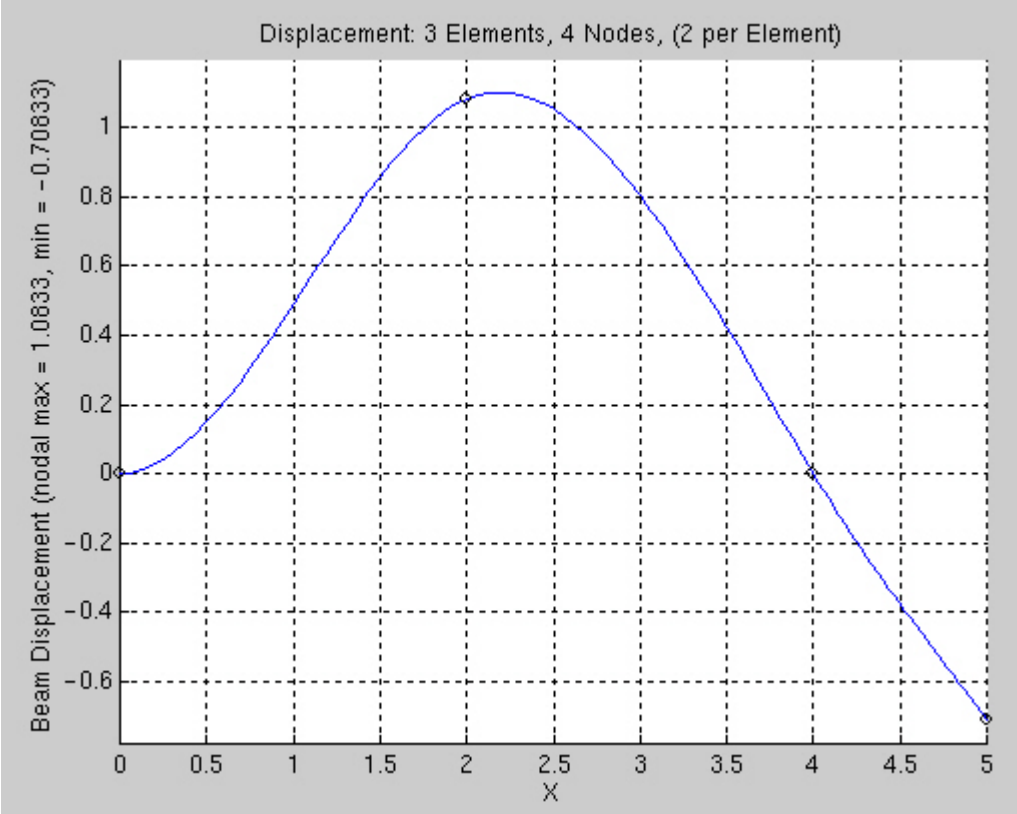
Read 4 mesh coordinate pairs, 3 elements with 2 nodes each



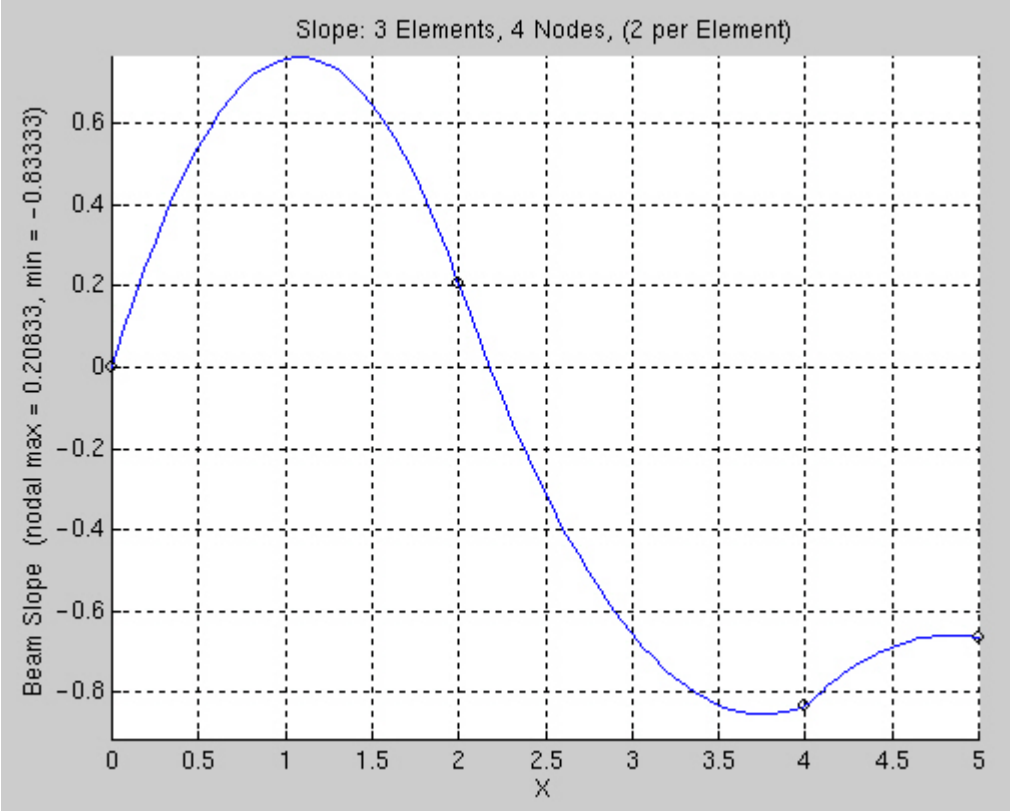
>> bc_flags_plot



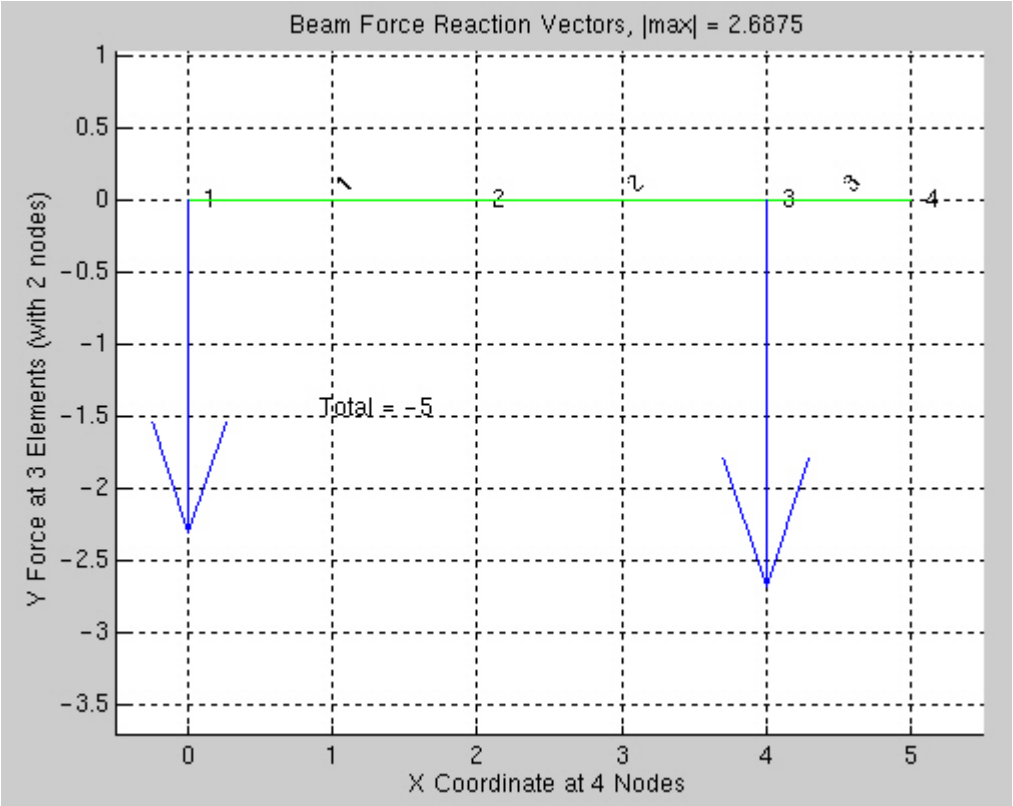
```
>> beam_C1_L2_defl  % displacement
Read 8 nodal dof, with 2 per node
```



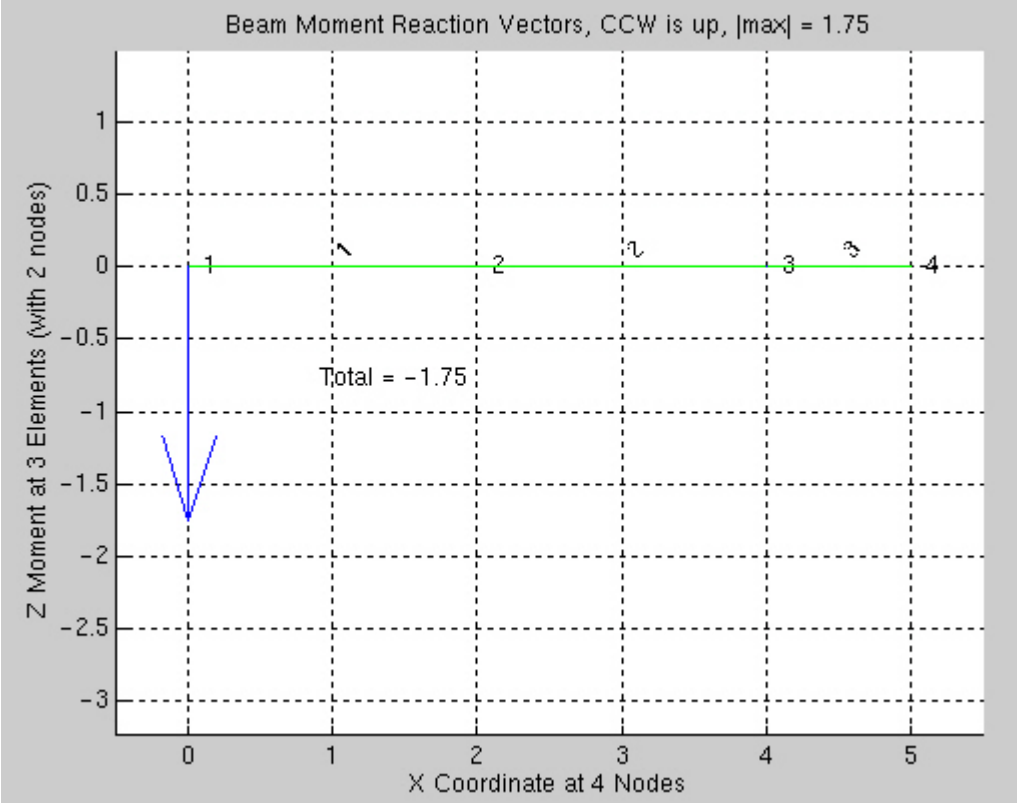

```
>> beam_C1_L2_defl(2) % slope
```



>> quiver_reaction_vec_mesh(1) % forces



>> quiver_reaction_vec_mesh(2) % moments



>> quit

Source Listing

```
function Modular_Beam_X (load_pt, pre_p, pre_e)
%.....................................................
%     Classic cubic beam for point loads & couples, line load
%             X COORDINATES CLOSED FORM INTEGRALS
%.....................................................
% pre_e = # of dummy items before el_type & connectivity
% pre_p = # of dummy items before BC_flag % coordinates
if ( nargin == 2 )                ;  % check for optional data
  pre_e = 0                       ;  % no el # in msh_typ_nodes
elseif ( nargin == 1 )            ;  % check for optional data
  pre_p = 0 ; pre_e = 0           ;  % no pt # in msh_bc_xyz
elseif ( nargin == 0 )            ;  % check for optional data
  load_pt = 0 ; pre_p = 0 ; pre_e = 0 ; % no point source data
end % if from argument count

%         Application and element dependent controls
n_g = 2     ; % number of DOF per node (deflection, slope)
n_q = 0     ; % number of quadrature points required
n_r = 1     ; % number of rows in B_e matrix

%              Read mesh input data files
[n_m, n_s, P, x, y, z]          = get_mesh_nodes    (pre_p) ;
[n_e, n_n, n_t, el_type, nodes] = get_mesh_elements (pre_e) ;
n_d = n_g*n_m                ; % system degrees of freedom (DOF)
n_i = n_g*n_n                ; % number of DOF per element
S = zeros (n_d, n_d) ; C = zeros (n_d, 1)      ; % initalize sums

%      Extract EBC flags from packed integer flag P
[EBC_flag] = get_ebc_flags (n_g, n_m, P)      ; % unpack flags
EBC_count  = sum( sum ( EBC_flag > 0 ) )      ; % # of EBC
fprintf ('Note: expecting %g EBC values. \n', EBC_count)

%                  Read EBC values, if any
if ( EBC_count > 0 )                              ; % need EBC data
  [EBC_value] = get_ebc_values (n_g, n_m, EBC_flag) ; % read data
end % if any EBC data expected

%    Read Neumann point loads data, if any, and insert in C
if ( load_pt > 0 )                        ; % need point loads data
  [C] = get_and_add_point_sources (n_g, n_m, C); % add point loads
end % if any point source expected

%  ============== ASSUMING HOMOGENOUS PROPERTIES =================

%    GENERATE ELEMENT MATRICES AND ASSYMBLE INTO SYSTEM
%  Assemble n_d by n_d square matrix terms from n_e by n_e

for j = 1:n_e                          ; % loop over elements ====>>
  S_e = zeros (n_i, n_i)                           ; % clear array
  M_e = zeros (n_i, n_i)                           ; % clear array
  C_p = zeros (n_i, 1) ; C_e = zeros (n_i, 1)     ; % clear arrays
  e_nodes = nodes (j, 1:n_n)                       ; % connectivity
```

```
%               SET ELEMENT PROPERTIES & GEOMETRY
  [I, E, Rho, Line_e] = set_constant_beam_prop (n_n); % properties
  L_e = x(e_nodes(2)) - x(e_nodes(1)) ; % beam length

%      ELEMENT CONDUCTION AND INTERNAL SOURCE MATRICES
  % for q = 1:n_q        ; % Loop over element quadrature points ---->

%         Constant cubic element square matrices
S_e = (E*I/L_e^3)*[ 12,    6*L_e,   -12,    6*L_e     ;
                    6*L_e, 4*L_e^2, -6*L_e, 2*L_e^2   ;
                   -12,   -6*L_e,    12,   -6*L_e     ;
                    6*L_e, 2*L_e^2, -6*L_e, 4*L_e^2 ] ; % stiffness

M_e = (Rho*L_e)*[ 156,     22*L_e,   54,     -13*L_e  ;
                  22*L_e,  4*L_e^2,  13*L_e, -3*L_e^2 ;
                  54,      13*L_e,   156,    -22*L_e  ;
                 -13*L_e, -3*L_e^2, -22*L_e,  4*L_e^2 ]/420; % mass

%  Map line load to node forces & moments; C_p = p_2_F * Line_e
    if ( any (Line_e) )            ; % then form forcing vector
      p_2_F = L_e * [ 21,    9     ;
                      3*L_e, 2*L_e ;
                      9,     21    ;
                     -2*L_e -3*L_e ] / 60 ; % cubic H, linear Line
      % 4 x 2 * 2 x 1 = 4 x 1 result
      C_p = p_2_F (1:n_i, 1:n_n) * Line_e ; % force moment @ nodes
      C_e = C_e + C_p                     ; % scatter
    end % if or set up resultant node loads for line load
% end % for loop over n_q element quadrature points          <----

%         SCATTER TO (ASSEMBLE INTO) SYSTEM ARRAYS
% Insert completed element matrices into system matrices
  [rows] = get_element_index (n_g, n_n, e_nodes); % eq numbers
  S (rows, rows) = S (rows, rows) + S_e  ; % add to system sq
  C (rows) = C (rows) + C_e              ; % add to sys column
end % for each j element in mesh                              <<====

%    ALLOCATE STORAGE FOR OPTIONAL REACTION RECOVERY
if ( EBC_count > 0 )                        ; % reactions occur
  [EBC_row, EBC_col] = save_reaction_matrices (EBC_flag, S, C);
end % if essential BC exist (almost always true)

%                   ECHO PROPERTIES
fprintf ('Application properties are: \n')
fprintf ('Elastity modulus = %g \n', E)
fprintf ('Moment of inertia = %g \n', I)
fprintf ('Line Load = [ %g %g ] \n', Line_e(1), Line_e(2))

%         ENFORCE ESSENTIAL BOUNDARY CONDITIONS
save_resultant_load_vectors (n_g, C)
[S, C] = enforce_essential_BC (EBC_flag, EBC_value, S, C);

%             COMPUTE SOLUTION & SAVE
T = S \ C                 ; % Compute displacements & rotations
list_save_beam_displacements (n_g, n_m, T)   ; % save and print
```

```
%          OPTIONAL REACTION RECOVERY & SAVE
if ( EBC_count > 0 )                    ; % reactions exist ?
  [EBC_react] = recover_reactions_print_save (n_g, n_d, ...
                EBC_flag, EBC_row, EBC_col, T); % reaction to EBC
end % if EBC exist

%       POST-PROCESS ELEMENT REACTIONS (MEMBER FORCES)
% output_beam_element_reactions (n_e, n_g, n_n, n_q, nodes, x, y, T)

% End finite element calculations.
% See /home/mech517/public_html/Matlab_Plots for graphic options
% http://www.owlnet.rice.edu/~mech517/help_plot.html for help
% end of Modular_Beam_X

%  ++++++++++++ functions in alphabetical order +++++++++++++++++

function [S, C] = enforce_essential_BC (EBC_flag, EBC_value, S, C)
% modify system linear eqs for essential boundary conditions
% (by trick to avoid matrix partitions, loses reaction data)
 n_d = size (C, 1)                        ; % number of DOF eqs
if ( size (EBC_flag, 2) > 1 ) ; % change to vector copy
  flag_EBC  = reshape ( EBC_flag',  1, n_d) ;
  value_EBC = reshape ( EBC_value', 1, n_d) ;
else
  flag_EBC  = EBC_flag ;
  value_EBC = EBC_value ;
end % if
 for j = 1:n_d                % check all DOF for essential BC
   if ( flag_EBC (j) )                          % then EBC here
%    Carry known columns*EBC to RHS. Zero that column and row.
%    Insert EBC identity, 1*EBC_dof = EBC_value.
     EBC = value_EBC (j)              ; % recover EBC value
     C (:) = C (:) - EBC * S (:, j) ; % carry known column to RHS
     S (:, j) = 0 ; S (j, :) = 0    ; % clear, restore symmetry
     S (j, j) = 1 ; C (j) = EBC     ; % insert identity into row
   end % if EBC for this DOF
 end % for over all j-th DOF
% end enforce_essential_BC (EBC_flag, EBC_value, S, C)

function [C] = get_and_add_point_sources (n_g, n_m, C)
  load msh_load_pt.tmp            ; % node, DOF, value (eq. number)
  n_u = size(msh_load_pt, 1)      ; % number of point sources
  if ( n_u < 1 )                  ; % missing data
    error ('No load_pt data in msh_load_pt.tmp')
  end % if user error
  fprintf ('Read %g point sources. \n', n_u)
  fprintf ('Node DOF Source_value \n')
  for j = 1:n_u                    ; % non-zero Neumann pts
    node  = msh_load_pt (j, 1)         ; % global node number
    DOF   = msh_load_pt (j, 2)         ; % local DOF number
    value = msh_load_pt (j, 3)         ; % point source value
    fprintf ('%g  %g  %g \n', node, DOF, value)
    Eq    = n_g * (node - 1) + DOF     ; % row in system matrix
    C (Eq) = C (Eq) + value     ; % add to system column matrix
```

```
  end % for each EBC
  fprintf ('\n')
% end get_and_add_point_sources (n_g, n_m, C)

function [EBC_flag] = get_ebc_flags (n_g, n_m, P)
 EBC_flag = zeros(n_m, n_g)   ; % initialize
 for k = 1:n_m                          ; % loop over all nodes
   if ( P(k) > 0 )                      ; % at least one EBC here
     [flags] = unpack_pt_flags (n_g, k, P(k)) ; % unpacking
     EBC_flag (k, 1:n_g) = flags (1:n_g)      ; % populate array
   end % if EBC at node k
 end % for loop over all nodes
% end get_ebc_flags

function [EBC_value] = get_ebc_values (n_g, n_m, EBC_flag)
 EBC_value = zeros(n_m, n_g)  ; % initialize to zero
  load msh_ebc.tmp            ; % node, DOF, value (eq. number)
  n_c = size(msh_ebc, 1)      ; % number of constraints
  fprintf ('Read %g EBC data with Node, DOF, Value. \n', n_c)
  disp(msh_ebc)                    ; % echo input
  for j = 1:n_c                    ; % loop over ebc inputs
    node  = round (msh_ebc (j, 1)) ; % node in mesh
    DOF   = round (msh_ebc (j, 2)) ; % DOF # at node
    value = msh_ebc (j, 3)         ; % EBC value
    % Eq  = n_g * (node - 1) + DOF ; % row in system matrix
    EBC_value (node, DOF) = value  ; % insert value in array
    if ( EBC_flag (node, DOF) == 0 ) % check data consistency
      fprintf ('WARNING: EBC but no flag at node %g & DOF %g. \n', ...
                node, DOF)
    end % if common user error
  end % for each EBC
  EBC_count = sum (sum ( EBC_flag > 0 )) ; % check input data
  if ( EBC_count ~= n_c )                ; % probable user error
    fprintf ('WARNING: mismatch in bc_flag count & msh_ebc.tmp')
  end % if user error
% end get_ebc_values

function [rows] = get_element_index (n_g, n_n, e_nodes)
% calculate system DOF numbers of element, for gather, scatter
rows = zeros (1, n_g*n_n)           ; % allow for node = 0
for k = 1:n_n                       ; % loop over element nodes
  global_node = round (e_nodes (k)) ; % corresponding sys node
  for i = 1:n_g                     ; % loop over DOF at node
    eq_global  = i + n_g * (global_node - 1) ; % sys DOF, if any
    eq_element = i + n_g * (k           - 1) ; % el DOF number
    if ( eq_global > 0 )             ; % check node=0 trick
      rows (1, eq_element) = eq_global  ; % valid DOF > 0
    end % if allow for omitted nodes
  end % for DOF i                     % end local DOF loop
end % for each element node           % end local node loop
% end get_element_index

function [n_e, n_n, n_t, el_type, nodes] = get_mesh_elements (pre_e) ;
% MODEL input file controls (for various data generators)
if (nargin == 0)  ; % default to no proceeding items in data
```

```
  pre_e = 0       ; % Dummy items before el_type & connectivity
end % if

load  msh_typ_nodes.tmp        ; % el_type, connectivity list (3)
n_e = size (msh_typ_nodes,1)   ; % number of elements
if ( n_e == 0 )                ; % data file missing
  error ('Error missing file msh_typ_nodes.tmp')
end % if error
n_n = size (msh_typ_nodes,2) - pre_e - 1 ; % nodes per element
fprintf ('Read %g elements with type number & %g nodes each. \n', ...
          n_e, n_n)
el_type = round (msh_typ_nodes(:, pre_e+1)); % el type number >= 1
n_t     = max(el_type)                    ; % number of element types
fprintf ('Maximum number of element types = %g. \n', n_t)
nodes (1:n_e, 1:n_n) = msh_typ_nodes (1:n_e, (pre_e+2:pre_e+1+n_n));
disp(msh_typ_nodes (:, (pre_e+1:pre_e+1+n_n))) % echo data
% end get_mesh_elements

function [n_m, n_s, P, x, y, z] = get_mesh_nodes (pre_p) ;
% MODEL input file controls (for various data generators)
if (nargin == 0) % override default
  pre_p = 0 ; % Dummy items before BC_flag % coordinates
end % if

%       READ MESH AND EBC_FLAG INPUT DATA
% specific problem data from MODEL data files (sequential)
load msh_bc_xyz.tmp       ; % bc_flag, x-, y-, z-coords
n_m = size (msh_bc_xyz,1) ; % number of nodal points in mesh

if ( n_m == 0 )           ; % data missing !
  error ('Error missing file msh_bc_xyz.tmp')
end % if error
n_s = size (msh_bc_xyz,2) - pre_p - 1 ; % number of space dimensions
fprintf ('Read %g nodes with bc_flag & %g coordinates. \n', n_m, n_s)
msh_bc_xyz (:, (pre_p+1))= round (msh_bc_xyz (:, (pre_p+1)));
P = msh_bc_xyz (1:n_m, (pre_p+1)) ; % integer Packed BC flag
x = msh_bc_xyz (1:n_m, (pre_p+2)) ; % extract x column
y (1:n_m, 1) = 0.0 ;  z (1:n_m, 1) = 0.0 ; % default to zero

if (n_s > 1 )                                ; % check 2D or 3D
  y = msh_bc_xyz (1:n_m, (pre_p+3))          ; % extract y column
end % if 2D or 3D
if ( n_s == 3 )                              ; % check 3D
  z = msh_bc_xyz (1:n_m, (pre_p+4))          ; % extract z column
end % if 3D
disp(msh_bc_xyz (:, (pre_p+1):(pre_p+1+n_s))) ; % echo data
% end get_mesh_nodes

function list_save_beam_displacements (n_g, n_m, T)
 fprintf ('\n') ;
 fprintf('Node  Y_displacement  Z_rotation at %g nodes \n', n_m)
 T_matrix = reshape (T, n_g, n_m)'            ; % pretty shape
 % save results (displacements) to MODEL file: node_results.tmp
 fid = fopen('node_results.tmp', 'w') ; % open for writing
 for j = 1:n_m                        ; % node loop, save displ
```

```
   fprintf (fid, '%g %g \n',  T_matrix (j, 1:n_g)) ; % to file
   fprintf ('%g %g %g \n', j, T_matrix (j, 1:n_g)) ; % to screen
 end % for j DOF
% end list_save_beam_displacements (n_g, n_m, T)

function list_save_displacements_results (n_g, n_m, T)
  fprintf ('\n') ;
  fprintf('X_disp      Y_disp      Z_disp   at %g nodes \n', n_m)
  T_matrix = reshape (T, n_g, n_m)'            ; % pretty shape
  disp (T_matrix)                              ; % print displacements
  % save results (displacements) to MODEL file: node_results.tmp
  fid = fopen('node_results.tmp', 'w')          ; % open for writing
  for j = 1:n_m                                 ; % save displacements
    if ( n_g == 1 )
      fprintf (fid, '%g               \n', T_matrix (j, 1:n_g)) ;
    elseif ( n_g == 2 )
      fprintf (fid, '%g %g            \n', T_matrix (j, 1:n_g)) ;
    elseif ( n_g == 3 )
      fprintf (fid, '%g %g %g         \n', T_matrix (j, 1:n_g)) ;
    elseif ( n_g == 4 )
      fprintf (fid, '%g %g %g %g      \n', T_matrix (j, 1:n_g)) ;
    elseif ( n_g == 5 )
      fprintf (fid, '%g %g %g %g %g   \n', T_matrix (j, 1:n_g)) ;
    elseif ( n_g == 6 )
      fprintf (fid, '%g %g %g %g %g %g \n', T_matrix (j, 1:n_g)) ;
    else
      error ('reformat list_save_displacements_results for n_g > 6.')
    end % if
  end % for j DOF
% end list_save_displacements_results (T)

function [EBC_react] = recover_reactions_print_save (n_g, n_d, ...
                        EBC_flag, EBC_row, EBC_col, T)
% get EBC reaction values by using rows of S & C (before EBC)
  n_d = size (T, 1)                       ; % number of system DOF
% n_c x 1 = n_c x n_d  * n_d x 1 +  n_c x 1
  EBC_react = EBC_row * T - EBC_col       ; % matrix reactions (+-)
  % save reactions (forces) to MODEL file: node_reaction.tmp
  fprintf ('\n')                          ; % Skip a line
  fprintf ('Node, DOF, Value for %g Reactions \n', ...
            sum (sum (EBC_flag > 0)))     ; % header
  fid = fopen('node_reaction.tmp', 'w') ; % open for writing
  if ( size (EBC_flag, 2) > 1 )             ; % change to vector copy
    flag_EBC  = reshape ( EBC_flag',  1, n_d) ; % changed
  else
    flag_EBC  = EBC_flag                      ; % original vector
  end % if
  Totals = zeros (1, n_g)                 ; % zero input totals
  kount = 0                               ; % initialize counter
  for j = 1:n_d                           ; % extract all EBC reactions
    if ( flag_EBC(j) )                        ; % then EBC here
  %   Output node_number, component_number, value, equation_number
      kount = kount + 1                       ; % copy counter
      node  = ceil(j/n_g)                     ; % node at DOF j
      j_g   = j - (node - 1)*n_g              ; % 1 <= j_g <= n_g
```

```
      React = EBC_react (kount, 1)          ; % reaction value
      fprintf ( fid, '%g %g %g \n', node, j_g, React);% save
      fprintf ('%g %g %g \n', node, j_g, React);     % print
      Totals (j_g) = Totals (j_g) + React   ; % sum all components
    end % if EBC for this DOF
  end % for over all j-th DOF
  fprintf ('Totals = ') ; disp(Totals)      ; % echo totals
% end recover_reactions_print_save (EBC_row, EBC_col, T)

function [EBC_row, EBC_col] = save_reaction_matrices (EBC_flag, S, C)
  n_d       = size (C, 1)                   ; % number of system DOF
  EBC_count = sum (sum (EBC_flag))          ; % count EBC & reactions
  EBC_row   = zeros(EBC_count, n_d)         ; % reaction data
  EBC_col   = zeros(EBC_count, 1)           ; % reaction data
  if ( size (EBC_flag, 2) > 1 )             ; % change to vector copy
    flag_EBC  = reshape ( EBC_flag',  1, n_d) ; % changed
  else
    flag_EBC  = EBC_flag                      ; % original vector
  end % if
  kount = 0                                 ; % initialize counter
  for j = 1:n_d      % System DOF loop, check for displacement BC
    if ( flag_EBC (j) )                     ; % then EBC here
  %   Save reaction data to be destroyed by EBC solver trick
      kount = kount + 1                     ; % copy counter
      EBC_row(kount, 1:n_d) = S (j, 1:n_d) ; % copy reaction data
      EBC_col(kount, 1)     = C (j)         ; % copy reaction data
    end % if EBC for this DOF
  end % for over all j-th DOF                % end sys DOF loop
% end save_reaction_matrices (S, C, EBC_flag)

function save_resultant_load_vectors (n_g, C)
% save resultant forces to MODEL file: node_resultants.tmp
  n_d = size (C, 1)                       ; % number of system DOF
  fprintf ('\n')                          ; % Skip a line
  % fprintf ('Node, DOF, Resultant Force (1) or Moment (2)  \n')
  fprintf ('Node, DOF, Resultant input sources \n')
  fid = fopen('node_resultant.tmp', 'w'); % open for writing
  Totals = zeros (1, n_g)                 ; % zero input totals
  for j = 1:n_d                           ; % extract all resultants
    if ( C (j) ~= 0. )                        ; % then source here
  %   Output node_number, component_number, value, equation_number
      node  = ceil(j/n_g)                   ; % node at DOF j
      j_g   = j - (node - 1)*n_g            ; % 1 <= j_g <= n_g
      value = C (j)                         ; % resultant value
      fprintf ( fid, '%g %g %g %g \n', node, j_g, value, j);% save
      fprintf ('%g %g %g \n', node, j_g, value);     % print
      Totals (j_g) = Totals (j_g) + value   ; % sum all inputs
    end % if non-zero for this DOF
  end % for over all j-th DOF
  fprintf ('Totals = ') ; disp(Totals)      ; % echo totals
% end save_resultant_load_vectors (n_g, n_m, C)

function [I, E, Rho, Line_e] = set_constant_beam_prop (n_n, Option) ;
  if ( nargin == 1 )
    Option = 1 ;
```

```
  elseif ( nargin == 0 )
    n_n = 2 ; Option = 1 ;
  end % if problem Option number
  Line_e = zeros (n_n, 1)         ; % default line load at nodes
  switch  Option
    case 1 % Propped cantilever with uniform load, L, L/4
           % Wall reactions: V=37*Line*L/64 M=7*Line*L^2/64
           % Roller reaction: R= 43*Line*L/64
           % Total vertical load: 5*Line*L/4
      %       *-----(1)-----*-----(2)-----*--(3)--*     EI
      % Fixed@1     L/2     2      Roller@3  L/4  4
      I = 1.0 ; E = 1.0 ; Rho = 0.0 ; Line_e = [1.0; 1.0] ;
    case 2 % cantilever with uniform load, L
      I = 1.0 ; E = 1.0 ; Rho = 0.0 ; Line_e = [1.0; 1.0] ;
    otherwise
      I = 1.0 ; E = 1.0 ; Rho = 0.0; % default shape & material
  end % switch
% end set_constant_beam_prop;

function [flags] = unpack_pt_flags (n_g, N, flag)
% unpack n_g integer flags from the n_g digit flag at node N
% integer flag contains (left to right) f_1 f_2 ... f_n_g
 full = flag                          ; % copy integer
 check = 0                            ; % validate input
 for Left2Right = 1:n_g               ; % loop over local DOF at k
   Right2Left = n_g + 1 - Left2Right ; % reverse direction
   work = floor (full / 10)           ; % work item
   keep = full - work * 10            ; % work item
   flags (Right2Left) = keep          ; % insert into array
   full = work                        ; % work item
   check = check + keep * 10^(Left2Right - 1) ; % validate
 end % for each local DOF
 if ( flag > check )                  ; % check for likely error
   fprintf ('WARNING: bc flag likely reversed at node %g. \n', N)
 end % if likely user error
% end unpack_pt_flags
```