HW 10, matlab solution

Nathan Barton

November 12, 1999

1 Problem 3

The matlab files for the solution are given below. First is the output from the driver file, followed by the driver file itself and then the m-file fan_solve.m. The m-files are also available for download from the web page.

Given that I had already programmed the force calculation to work over the full cycle, the m-files are not much different this week. There is the added global variable l_statics which I use to turn on and off the forces from dynamics. It is interesting to note, that for this problem, a statics analysis tells you that the motor is strong enough to drive the mechanism, but when you include the dynamical forces the motor is no longer adequate. You were not required to do the statics version yourselves, I just did it for fun.

1.1 output from driver file

```
>> fan_driver
check of energy conservation -2.031e-08
maximum moment exerted by motor 6.828
maximum moment exerted on motor 10.546
the motor is not capable of driving the mechanism over a cycle
for the case of statics ...
the motor is capable of driving the mechanism over a cycle
```

1.2 Driver file

fan_driver.m

```
% some of the values that are set in this file are not used until later
% parts of the problem

global r r2 rg lthg;
global iwk iwu1 iwu2 ium;
global w_data;
global gc g_by_gc;
global m ai;
global l_statics;

% rated torque of motor
max_motor_torque_output = 6.0;

% do as dynamics problem
```

```
l_statics = 0;

% figure counter
ifig = 0;

% some of geometry of linkage
%
% lengths of bars in linkage (meters)
r = 0.01*[10.0 40.0 30.0 40.0]';
r2 = r*r';
% angle (relative to local coordinates on link) to the center of mass;
% see m-file linkage_ag.m
lthg = [0.0 0.0 pi/6]';
% distance to centers of mass from starting point of link
rg = [r(1)*0.5 r(2)*0.5 r(3)/cos(lthg(3))]';

% masses, moments of inertia
m = [0.0 0.0 10.0]'; % in kg
% moment of inertia for uniform distribution of mass over links
ai = zeros(3,1);
ai(1) = m(1)*r2(1,1)/12.0;
ai(2) = m(2)*r2(2,2)/12.0;
diam_fan = 1.0;
ai(3) = m(3)*diam_fan*diam_fan/16.0;

% gravitation
g       = [0.0 -9.8]'; % direction 2 is down, in m/s^2
gc      = 1.0; % metric, trust mass to be in kg
g_by_gc = g/gc;

th1_init = atan2(3,4);
th_init = fan_angles(r,r2,th1_init);

% data for initial angular velocities
w_data = [2.0*pi 0.0 0.0]';
% indices for known (iwk) and unknown (iwu*) angular velocities
% and index for unknown moment (ium)
iwk = 1; iwu1 = 2; iwu2 = 3; ium = 1;

tstart = 0.0; tstop = 2.0*pi/abs(w_data(iwk)); nt = 101;
tspan = linspace(tstart, tstop, nt);
dtime = (tstop - tstart)/(nt-1);


options = odeset('AbsTol',1e-10);
[t, th_num] = ode45('fan_solve',tspan,th_init,options);
```

```
% obtain accelerations over a full cycle

thdot_all = zeros(nt,3);
ag_fan = zeros(nt,2);   mag_ag_fan = zeros(nt,1);
thddot_fan = zeros(nt); mag_thddot_fan = zeros(nt,1);
for it = 1:nt;
   [thdot,thddot,ag] = fan_solve(t(it), th_num(it,:));
   % store results at time it
   ag_fan(it,:) = ag(:,3)';
   mag_ag_fan(it) = norm(ag(:,3));
   thddot_fan(it) = thddot(3);
   mag_thddot_fan(it) = abs(thddot(3));
   thdot_all(it,:) = thdot';
end;

% plot all of the angular velocities of the moving links
ifig = ifig+1; figure(ifig);
plot(...
   t, thdot_all(:,1), ':', ...
   t, thdot_all(:,2), '-', ...
   t, thdot_all(:,3), '--')
xlabel('time (s)'); ylabel('angular velocity (radians/s)');
title('angular velocities of links, from numerical integration');
legend('thdot1','thdot2','thdot3');

% plot angular acceleration of link 3

ifig = ifig+1; figure(ifig);
plot(t,thddot_fan);
xlabel('time (s)'); ylabel('angular acceleration (rad/s^2)');
title('angular acceleration of fan for a full cycle');

% plot componenets of linear acceleration of fan's center of mass over cycle

ifig = ifig+1; figure(ifig);
plot(...
   t,ag_fan(:,1), '-', ...
   t,ag_fan(:,2), '--');
xlabel('time (s)'); ylabel('linear acceleration (m/s^2)');
title('components of acceleration of fan for a full cycle');
legend('x component','y component');



% loop over cycle to get moment at motor
```

```
ma = zeros(nt,1);
for it = 1:nt;
[thdot,thddot,ag,f] = fan_solve(t(it), th_num(it,:));
ma(it) = f(9);
end;

% check of conservation of energy over a cycle, scale the integral
% by the integral of the absolute value (to get the error as a
% relative magnitude)
%
integ = sum(ma(1:(nt-1)))*dtime*w_data(iwk);
integ_scale = sum(abs(ma(1:(nt-1))))*w_data(iwk);
fprintf('check of energy conservation %.3e\n', ...
integ/integ_scale);

if (w_data(iwk) > 0.0);
  % moment is in same direction as rotation so work flows into
  % the system when the moment is positive
  sign_wk = sign(w_data(iwk));
else
  % moment is in opposite direction as rotation so work flows into
  % the system when the moment is negative
end

% moment at motor
ifig = ifig+1; figure(ifig);
plot(t,ma*sign_wk);
xlabel('time (s)'); ylabel('moment (N/m)');
title('moment for a full cycle; when positive, work flows into system');

max_moment_by = max( ma*sign_wk);
max_moment_on = max(-ma*sign_wk);
%
fprintf('maximum moment exerted by motor %.3f\n',max_moment_by);
fprintf('maximum moment exerted on motor %.3f\n',max_moment_on);

if (max_moment_by > max_motor_torque_output);
  fprintf('the motor is not capable of driving the mechanism over a cycle\n');
else
  fprintf('the motor is capable of driving the mechanism over a cycle\n');
end

% now check and see if the motor could have driven the fan if we had just
% done statics (not required, but fun to check)

l_statics = 1;
%
```

```
% see fan_solve function for the solution this week; all of the kinematics
% are the same, just have to rerun the force and moment calculation
ma_statics = zeros(nt,1);
for it = 1:nt;
[thdot,thddot,ag,f] = fan_solve(t(it), th_num(it,:));
ma_statics(it) = f(9);
end;

% moment at motor, statics
ifig = ifig+1; figure(ifig);
plot(t,ma_statics*sign_wk);
xlabel('time (s)'); ylabel('moment (N/m)');
title('moment for a full cycle; statics; when positive, work flows into system');

max_moment_by_statics = max( ma_statics*sign_wk);

fprintf('for the case of statics ...\n');
if (max_moment_by_statics > max_motor_torque_output);
  fprintf('the motor is not capable of driving the mechanism over a cycle\n');
else
  fprintf('the motor is capable of driving the mechanism over a cycle\n');
end
```

1.3 big chunky matlab file to calculate lots of things

fan_solve.m

```
function [thdot,thddot,ag,f] = fan_solve(t, th)
%
% NOTE ON USE OF NARGOUT:
% If this function is not called with all of the output arguments, then
% the extra output arguments are not computed -- no sense in finding the
% values if the caller of the function is not going to use them. For
% example, if this function is passed to ode45 with something like
%        [t, th_num] = ode45('fan_solve',tspan,th_init,options);
% than ode45 will call fan_solve with only one output argument, the
% equivalent of you calling fan_solve like
%        [thdot] = fan_solve(t_current, th_current);
% In that case, only one output argument has been requested, and matlab
% sets the automatic variable nargout to be equal to 1
%
% INPUT arguments:
% t  --         scalar, input, current time; never used for anything
% th --         vector of length nlink (number of moving links); angle
%               of a link measured counterclockwise from the i-direction
% OUTPUT arguments:
% given angles of components, find:
```

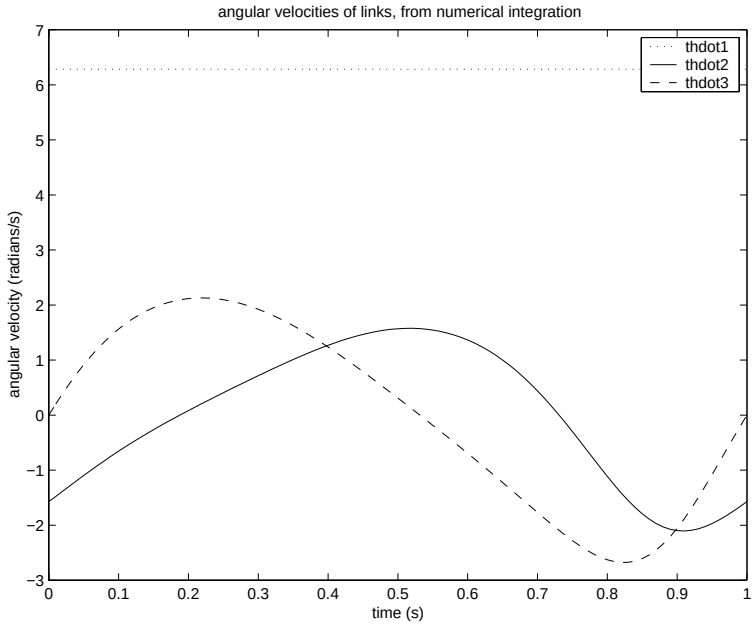
```
% thdot --      vector of length nlink; angular velocities
% thddot --     vector (nlink); angular accelerations
% ag     --     matrix (2,nlink); 2 because two-dimensional vector for
%               the accelerations; linear acceleration of mass centers
% f      --     vector of length 9; 8 forces at pins and 1 unknown moment
%
% GLOBALS that have been assigned in the driver script
global r r2 rg lthg;
%    r --    vector(nlink); length of the link
%    r2 --   matrix(nlink,nlink); "squares" of links, r2(i,j)=r(i)*r(j)
%    rg --   vector(nlink); length from beginning of link to center of mass
%    lthg -- vector(nlink); angle from line of link to line from the
%            beginning of the link to the center of mass (counterclockwise)
global iwk iwu1 iwu2 ium;
%    iwk --  scalar; index for link for which angular velocity is known
%    iwu1 -- scalar; index for first link for which angular velocity is unknown
%    iwu2 -- scalar; index for second link --||--
%    ium  -- scalar; index for link to which unknown moment is applied
global w_data;
%    w_data -- vector(nlink); data giving known angular velocity; entry
%              iwk is the only number used in this vector
global gc g_by_gc;
%    gc      -- scaling used when using English units
%    g_by_gc -- vector(2) giving the direction in which gravity acts
global m ai;
%    m  -- vector(nlink); masses of links
%    ai -- vector(nlink); angular inertia of links
global l_statics;
%    l_statics -- treat as statics problem if true
%
nlink = 3; % number of moving links
if (length(th) ~= nlink);
      disp('error, th of wrong length');
      return;
end;
%
% GET UNIT VECTORS:
% vectors nvec are along the links and vectors tvec are perpendicular to
% the links and in the direction of motion of the end of the link when
% the angular velocity is positive;
%
% nvec -- matrix(2,nlink)
% tvec -- matrix(2,nlink)
%
[nvec, tvec] = linkage_vec(th);
%
% solve for ANGULAR VELOCITIES;
```

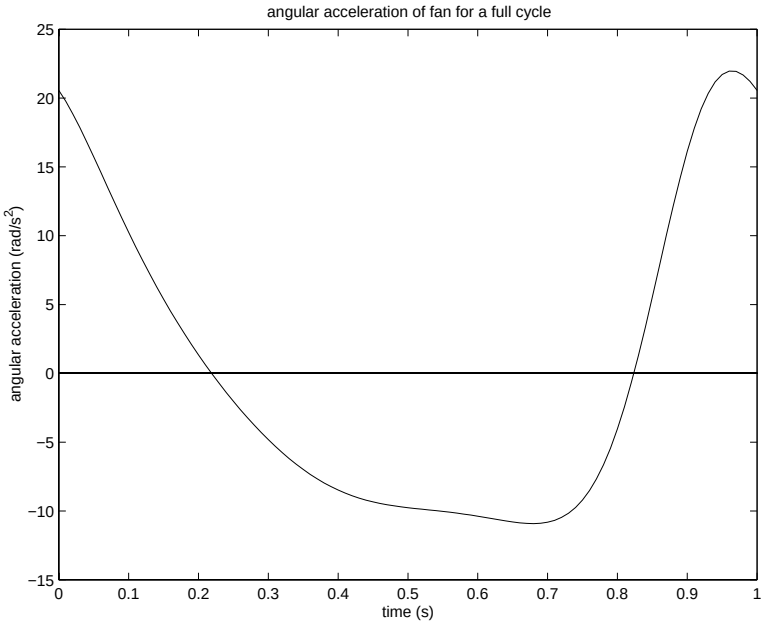
```
% note that tvec(:,i) and nvec(:,i) are column vectors so that
% A becomes a (2,2) matrix and y becomes a 2-column-vector, or (2,1) matrix
%
A = [r(iwu1)*tvec(:,iwu1) r(iwu2)*tvec(:,iwu2)];
y = [w_data(iwk)*r(iwk)*tvec(:,iwk)];
x = -A\y;
%
% set values in thdot
%
thdot = zeros(3,1);
thdot(iwk)  = w_data(iwk);
thdot(iwu1) = x(1);
thdot(iwu2) = x(2);
%
% end of calculation of angular velocities
%
if (nargout > 1);
%
% find ANGULAR ACCELERATIONS
%
% get squares of angular velocities
thdot2 = thdot.*thdot;
% thdot2 -- vector(nlink)
%
A = [r(iwu1)*tvec(:,iwu1) r(iwu2)*tvec(:,iwu2)];
y = [-thdot2(iwk)* r(iwk)* nvec(:,iwk)  + ...
     -thdot2(iwu1)*r(iwu1)*nvec(:,iwu1) + ...
     -thdot2(iwu2)*r(iwu2)*nvec(:,iwu2)];
x = -A\y;
%
% set values in thddot
%
thddot = zeros(3,1);
thddot(iwk)  = 0.0;
thddot(iwu1) = x(1);
thddot(iwu2) = x(2);
%
end; % end of calculation of angular accelerations
%
if (nargout > 2);
%
% find LINEAR ACCELERATION of mass centers
%
[ag,ng] = linkage_ag(thdot2,thddot,r,rg,lthg,nvec,tvec);
%
end; % end calculation of linear acceleration of mass centers
%
```

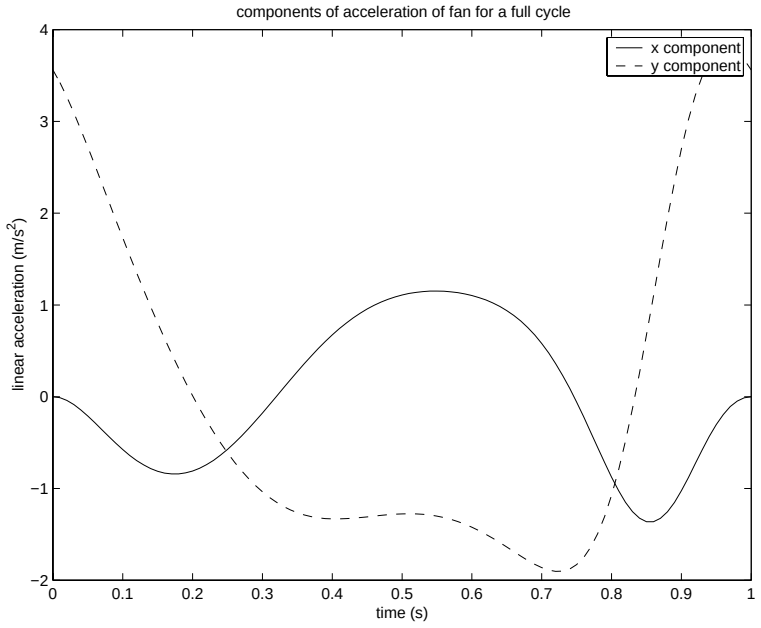
```
%
if (nargout > 3);
%
% find FORCES AND MOMENT
%
% The code below works for a sequence of links with forces defined in the
% negative sense at the beginning of a link (connection to previous link)
% and in the position sense at the end of a link (connection to next link)
%
% We have two equations from force balance and one equation from
% moment balance for each link
%
neq_per_link = 2+1; % number of equations per link
neq = nlink*neq_per_link; % number of total equations
%
% allow only one unknown moment; its index is imd
imd = (nlink+1)*2+1;
%
% zero necessary arrays
A = zeros(neq,neq);
b = zeros(neq,1);
f = zeros(neq,1);
%
% initialize equation counter; as entries are add to the system matrix A,
% this index will keep track of the equation number (row) to which numbers
% are being added;
eq_num = 0;
%
for i_link = 1:nlink; % loop over links
 %
 % equation numbers;
 % ieq -- force equation in the i-direction for the current link
 % jeq -- force equation in the j-direction for the current link
 % meq -- moment equation (in the k-direction) for the current link
 %
 ieq = eq_num+1; jeq = eq_num+2; meq = eq_num+3;
 %
 % increment the equation counter
 eq_num = eq_num + neq_per_link;
 %
 % force indices for ends of links
 if_base = (i_link-1)*2;
 % forces at start of link in i and j directions
 ifsi = 1 + if_base; ifsj = 2 + if_base;
 if_base = if_base + 2;
 % forces at end of link in i and j directions
 ifei = 1 + if_base; ifej = 2 + if_base;
```

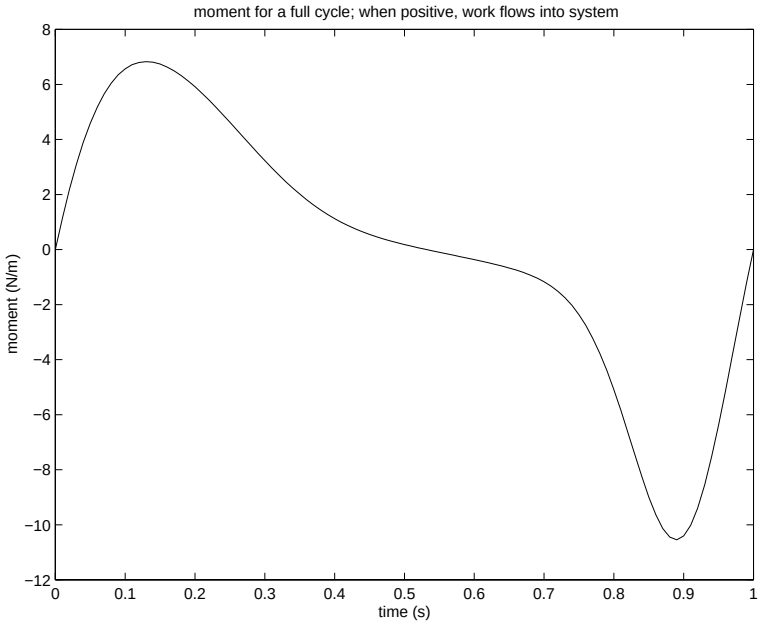
```
 %
 % force equations;
 %   forces at S act in negative direction and forces at E act in
 %   positive direction
 A(ieq,[ifsi, ifei]) = [-1.0 1.0];
 A(jeq,[ifsj, ifej]) = [-1.0 1.0];
 if (l_statics);
   b([ieq,jeq]) = -m(i_link)*g_by_gc(:);
 else
   b([ieq,jeq]) =  m(i_link)*(ag(:,i_link)/gc - g_by_gc(:));
 end
 %
 % moment equation, about beginning of link, point S
 %
 ntemp = cross_2d_ij(nvec(:,i_link));
 % ntemp is a 2-vector and you needn't worry about trying to give it a
 % physical meaning, it is simply the vector that when dotted with the
 % force at E (and multiplied by the length of the link) gives the moment
 % about S
 %
 A(meq,[ifei,ifej]) = r(i_link)*ntemp';
 if (ium == i_link);
   %this is the link on which unknown moment is applied
   A(meq, imd) = 1.0;
 end;
 if (l_statics);
   b(meq) = ...
        -m(i_link)*rg(i_link)*cross_2d(ng(:,i_link),g_by_gc);
 else
   b(meq) = ai(i_link)*thddot(i_link)/gc + ...
        +m(i_link)*rg(i_link)*cross_2d(ng(:,i_link),ag(:,i_link)) + ...
        -m(i_link)*rg(i_link)*cross_2d(ng(:,i_link),g_by_gc);
 end;
 %
end; % end of loop over links
%
% finally, ready to solve
%
f = A\b;
%
end; % end of force and moment calculations
```

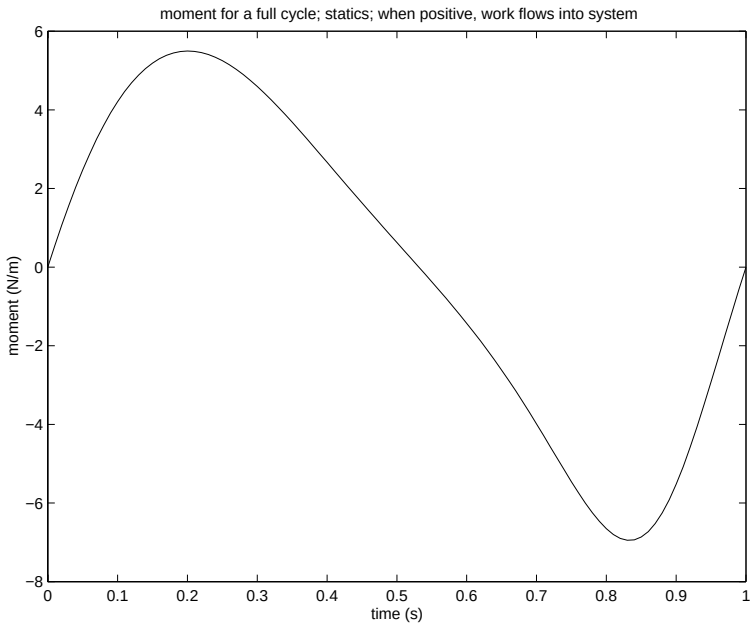
For `cross_2d.m`, `cross_2d_ij.m`, `fan_angles.m`, `linkage_vec.m`, `law_cos_len.m`, `linkage_ag.m` and `law_cos_ang.` see previous solutions or the bottom of the matlab part of the web page.


angular velocities of links, from numerical integration
thdot1
thdot2
thdot3
angular velocity (radians/s)
time (s)


angular acceleration of fan for a full cycle
angular acceleration (rad/s2)
time (s)


components of acceleration of fan for a full cycle
x component
y component
linear acceleration (m/s^2)
time (s)
4
3
2
1
0
−1
−2
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1


moment for a full cycle; when positive, work flows into system
moment (N/m)
time (s)
8
6
4
2
0
−2
−4
−6
−8
−10
−12
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1


moment for a full cycle; statics; when positive, work flows into system
moment (N/m)
time (s)
6
4
2
0
−2
−4
−6
−8
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1