Problem 4, hw 5, matlab solution

Nathan Barton

September 30, 1999

First some general comments. There is a main driver file for the problem and the output from that driver file is given, as are the figures that are generated. This matlab solution is more involved than the solutions that we would require of students in the course, but there are some advantages to the style of the programming. Whenever possible, matlab is left to do manipulation of vectors, reducing the amount of hand calculation and thus the chances for error. The code is somewhat more general than the problem at hand, but that can make it easier to read, debug, and use for other problems!

1 Driver file

prob4_script.m

```
%
disp(sprintf('matlab solution for problem 4 on hw 5'));
disp(sprintf('by Nathan Barton\n\n'));
%
disp('note: when looking at the bike _i_ direction is downward,')
disp('      _j_ is in direction of forward motion of the bike, and')
disp('      angles of the components are ccw defined relative to _i_')
%
global w4 a4 r r2;
global rg2a rg3c rg4d rg3b rg4c;
global m2 m3 m4 i2 i3 i4 g_by_gc gc;
global ifa1 ifa2 ifb1 ifb2 ifc1 ifc2 ifd1 ifd2 imd;
%
% figure counter
ifig = 0;
%
% define indices into force vector, used in solving dynamics problem;
% this is useful so that you do not have to remember the numbers
ifa1=1; ifa2=2; ifb1=3; ifb2=4; ifc1=5; ifc2=6; ifd1=7; ifd2=8; imd=9;
%
disp('check that get results for problem to which answer is known')
disp('  .. run functions on geometry for problem 4 on hw4')
%
r  = [30 18 20 8]';
r2 = r*r';
%
th4_init = pi/2;
thzero   = bike_angles(r,r2,th4_init);
w4 = -2*pi; a4 = 0.0e0;
[thdot, thddot] = bike_solve(0, thzero)
disp('..yes, get answer from last problem on hw4 (given sign conventions)');
```

```
% set values that are known; units are inches, lbf, lbm, radians, and seconds
%
% lengths, useful to have products of lengths handy, so compute r2
r  = [29 18 20 8]'; r2 = r*r';
%
% initial angle of crank
th4_init = pi/2;
%
% angular velocity and acceleration of crank
w4 = -2.0*pi; a4 = 0.0e0;
%
% gravitation
g       = [32.2 0.0]'; % direction 1 is down
gc      = norm(g);
if (gc == 0.0) % just in case g is set to [0.0 0.0]
gc = 1.0;
end;
g_by_gc = g/gc;
%
% specify mass in lbm
m2 = 30.0; m3 = 15.0; m4 = 0.0;
%
% lengths to centers of mass, moments of inertia
rg2b = r(2)*0.5; i2 = m2*r2(2,2)/12.0; rg2a = r(2)-rg2b;
rg3c = r(3)*0.5; i3 = m3*r2(3,3)/12.0; rg3b = r(3)-rg3c;
rg4d = r(4)*0.5; i4 = m4*r2(4,4)/12.0; rg4c = r(4)-rg4d;
%
% get initial angles
%
thzero = bike_angles(r,r2,th4_init)
%
disp('integrate to get angles for a full cycle (1 second)')
%
tstart = 0.0; tstop = 2.0*pi/abs(w4); nt = 1001;
tspan = linspace(tstart, tstop, nt);
dtime = (tstop - tstart)/(nt-1);
%
options = odeset('AbsTol',1e-10);
[t, th] = ode45('bike_solve',tspan,thzero,options);
%
% check that got back to starting point
%
check = abs((th(end,:)'-thzero(:)));
disp(sprintf('check that returned to original position:\n %e %e %e', ...
check(1),check(2),check(3)-2*pi))
%
% check point part way through cycle
%
% note: I was careful to code bike_angles so that it works anywhere in the
```

```
% of time ... students need not go to such lengths
%
frac=0.85; imid=floor(nt*frac);
thmid = bike_angles(r,r2,th(imid,3));
disp(sprintf('check of angles at %5.2f of the cycle: \n',frac))
check = th(imid,:)-thmid;
disp(sprintf('%e %e %e\n',check(1),check(2),check(3)));
%
% make plot of positions in time;
% rotate axes to appear as would expect looking at a bicycle
%
p = zeros(nt,2,4);
for it = 1:nt;
p(it,:,:) = [ 0 1 ; -1 0 ] * bike_pos(th(it,:));
end;
ifig = ifig+1; figure(ifig);
plot(...
p(:,1,1),p(:,2,1),'*', ...
p(:,1,2),p(:,2,2), ...
p(:,1,3),p(:,2,3), ...
p(:,1,4),p(:,2,4),'*')
axis('equal');
xlabel('horizontal position (in)'); ylabel('vertical position (in)');
title('hip, knee, pedal, and crankshaft positions for a full cycle');
%
% loop to compute forces and moments;
%
md = zeros(nt,1); fa=zeros(nt,2);
for it = 1:nt;
[thdot, thddot, ag, f] = bike_solve(t(it), th(it,:));
fa(it,:) = f([ifa1,ifa2])';
md(it)  = f(9);
end;
%
% plot forces and moments as a function of time
%
% moment at the crank does work
ifig = ifig+1; figure(ifig);
plot(t,md);
xlabel('time (s)'); ylabel('crank moment (in * lbf)');
title('moment at crankshaft for a full cycle');
%
% forces at hip do no work, but are fun to plot anyway
ifig = ifig+1; figure(ifig);
plot(t,fa(:,1),t,fa(:,2));
xlabel('time (s)'); ylabel('force (lbf)');
title('forces at hip for a full cycle (these forces do no work)');
%
% approximate integral of md, noting that time steps are evenly spaced
```

```
integ = sum(md(1:(end-1)))*dtime*w4;
integ_scale = sum(abs(md(1:(end-1))))*dtime*w4;
disp('integral in time of crank power for one cycle as a fraction of the');
disp('   integral of the absolute value of the same (to give');
disp('   fractional error in the conservation of energy): ');
disp(sprintf('%e',integ/integ_scale));
disp('   ... should be quite close to zero');
%
% notes:
% if set m2 = m3 = m4 = 0 (and thus i2 = i3 = i4 = 0), get md = 0 at
%  all times
% if set g = [0.0 0.0]' then get nonzero moment at crank, but it still
% integrates to zero over a cycle
%
```

2 output from driver file

```
>> prob4_script
prob4_script
matlab solution for problem 4 on hw 5
by Nathan Barton


note: when looking at the bike _i_ direction is downward,
      _j_ is in direction of forward motion of the bike, and
      angles of the components are ccw defined relative to _i_
check that get results for problem to which answer is known
  .. run functions on geometry for problem 4 on hw4

thdot =

   -2.8137
    1.6292
   -6.2832


thddot =

  -11.7402
   -4.7614
         0

..yes, get answer from last problem on hw4 (given sign conventions)

thzero =

    0.9682   -0.3485    1.5708
```

```
integrate to get angles for a full cycle (1 second)
check that returned to original position:
 7.098105e-05 4.413656e-05 8.881784e-16
check of angles at  0.85 of the cycle:

-7.856223e-05 1.681088e-05 0.000000e+00

integral in time of crank power for one cycle as a fraction of the
   integral of the absolute value of the same (to give
   fractional error in the conservation of energy):
-8.037813e-05
   ... should be quite close to zero
>>
```

3 m files

```
function th=bike_angels(r,r2,th4)
%
% calculate angles of components in leg/crank 4 bar linkage on a bike
% r  :  lengths of components
% r2 : r'*r, products of lengths
%
% assume know crank angle th(3)=th4, and assume th1 definted to be zero and
% is not part of vector th
%
% use dirty (and expensive) trick to get th4 in the range 0 to pi
gam5 = pi - acos(cos(th4));
%
[rac, rac2] = law_cos_len(r2(1,1), r2(4,4), r2(1,4), gam5);
%
gam1 = law_cos_ang(r2(1,1), r2(4,4), rac2, r(4)*rac);
gam2 = pi - gam1 - gam5;
%
% now fix signs if th4 is between -pi and zero
if sin(th4) < 0
gam1 = -gam1; gam2 = -gam2;
end;
%
% expect 0 <= gamb,gam3,gam4 <= pi
gamb = law_cos_ang(rac2, r2(2,2), r2(3,3), r2(2,3));
gam3 = law_cos_ang(r2(3,3), rac2, r2(2,2), rac*r(2));
gam4 = pi - gam3 - gamb;
th2 = gam2 + gam3;
%
th3 = th2 - pi + gamb; % not a triangle based equation
%
% set values
%
```

```
th(2) = th3;
th(3) = th4;
```

```
function [nab, tab, nbc, tbc, ncd, tcd] = bike_ntvec(th)
%
% get useful unit vectors
% nab is the unit vector from b pointing toward a
% tab is the result of k ^ nab (where ^ means cross-product)
%
% extract values from th
th2 = th(1); th3 = th(2); th4 = th(3);
%
tab = [ sin(th2) -cos(th2)]'; nab = [-cos(th2) -sin(th2)]';
tbc = [ sin(th3) -cos(th3)]'; nbc = [-cos(th3) -sin(th3)]';
tcd = [-sin(th4)  cos(th4)]'; ncd = [ cos(th4)  sin(th4)]';
```

```
function p = bike_pos(th)
%
% find positions of points A, B, C, and D; start from A fixed
%
global r;
%
p = zeros(2,4);
pa = [0.0 0.0]';
%
% calculate unit vectors
[nab, tab, nbc, tbc, ncd, tcd] = bike_ntvec(th);
%
pb = (-nab)*r(2);
pc = pb + (-nbc)*r(3);
pd = pc + (-ncd)*r(4);
%
p = [pa pb pc pd];
```

```
function [thdot,thddot,ag,f] = bike_solve(t, th)
%
% given angles, find angular velocity etc
% th     : current angles
% thdot  : angular velocity
% thddot : angular acceleration
% ag     : center of mass linear acceleration
```

```
%
% only compute values if they are actually asked for when the function is
% called -- use matlab's nargout to determine the number of output arguments
%
global w4 a4 r r2;
global rg4d rg3c rg2a rg3b rg4c;
global m2 m3 m4 i2 i3 i4 g_by_gc gc;
global ifa1 ifa2 ifb1 ifb2 ifc1 ifc2 ifd1 ifd2 imd;
%
% calculate unit vectors
[nab, tab, nbc, tbc, ncd, tcd] = bike_ntvec(th);
nba = -nab; ncb = -nbc; ndc = -ncd;
%
% extract values from th
%
th2 = th(1); th3 = th(2); th4 = th(3);
%
% calculate angular velocities
%
w2 = -(w4*r(4)*tcd'*nbc) / (r(2)*tab'*nbc);
w3 = -(w4*r(4)*tcd'*nab) / (r(3)*tbc'*nab);
%
% finally, set values in thdot
%
thdot = [w2 w3 w4]';
%
if nargout > 1
%
% have been asked for thddot too, angular acceleration; solve system of
% linear equations
%
% find sqaures of angular velocities
w22=w2*w2; w32 = w3*w3; w42 = w4*w4;
%
% zero the necessary arrays
y = zeros(2,1); A = zeros(2,2); x = zeros(2,1);
%
y(1)   = -ncd(1)*w42*r(4) - nbc(1)*w32*r(3) - nab(1)*w22*r(2);
y(2)   = -ncd(2)*w42*r(4) - nbc(2)*w32*r(3) - nab(2)*w22*r(2);
% tcd(1)*a4*r(4) = 0
% tcd(2)*a4*r(4) = 0
A(1,1) = tbc(1)*r(3); A(1,2) = tab(1)*r(2);
A(2,1) = tbc(2)*r(3); A(2,2) = tab(2)*r(2);
x = -A\y; a3 = x(1); a2 = x(2);
%
% set values in thddot
%
thddot = [a2 a3 a4]';
%
```

```
%
if nargout > 2
%
% have asked for linear acceleration of centers of mass
%
ag2 =    nab*w22*rg2a - tab*a2*rg2a;
ag3 =  -ncd*w42*r(4) + tcd*a4*r(4) - nbc*w32*rg3c + tbc*a3*rg3c;
ag4 =  -ncd*w42*rg4d + tcd*a4*rg4d;
%
ag(:,:) = [ag2 ag3 ag4];
%
end; % end of linear acceleration calculation
%
if nargout > 3
%
% have asked for forces
%
% zero necessary arrays
A = zeros(9,9);
b = zeros(9,1);
f = zeros(9,1);
%
% here to global variables ifa1 etc are used to index the force and moment
% degrees of freedom and the equations are entered in a systematic way so
% that debugging is easier; eq_num is a running counter of the equation
% number so that you do not have to worry about keeping track of that;
% ieq, jeq, and meq are indices for the equation numbers for the
% component currently under consideration (ieq for the i-direction, jeq
% for the j-direction and meq for the moment equation); note that
% ifa1 is the index for the force at point a in the i direction,
% ifd2 is the index for the force at d in the j direction, etc.
%
eq_num = 0;
%
% component 2
%
eq_num = eq_num + 1; ieq = eq_num;
eq_num = eq_num + 1; jeq = eq_num;
eq_num = eq_num + 1; meq = eq_num;
%
% force equations
A(ieq,[ifa1,ifb1]) = [1.0 1.0]; b(ieq) = m2*ag2(1)/gc - m2*g_by_gc(1);
A(jeq,[ifa2,ifb2]) = [1.0 1.0]; b(jeq) = m2*ag2(2)/gc - m2*g_by_gc(2);
%
% moment equation
A(meq,ifb1) = r(2)*(-nba(2)); A(meq,ifb2) = r(2)*(nba(1));
b(meq) = i2*a2/gc + m2*rg2a*cross_2d(nba,ag2)/gc - rg2a*m2*cross_2d(nba,g_by_gc);
%
% component 3
```

```
eq_num = eq_num + 1; ieq = eq_num;
eq_num = eq_num + 1; jeq = eq_num;
eq_num = eq_num + 1; meq = eq_num;
%
% force equations
A(ieq,[ifb1,ifc1]) = [-1.0 1.0]; b(ieq) = m3*ag3(1)/gc - m3*g_by_gc(1);
A(jeq,[ifb2,ifc2]) = [-1.0 1.0]; b(jeq) = m3*ag3(2)/gc - m3*g_by_gc(2);
%
% moment equation
A(meq,ifc1) = r(3)*(-ncb(2)); A(meq,ifc2) = r(3)*(ncb(1));
b(meq) = i3*a3/gc + m3*rg3b*cross_2d(ncb,ag3)/gc - rg3b*m3*cross_2d(ncb,g_by_gc);
%
% component 4
%
eq_num = eq_num + 1; ieq = eq_num;
eq_num = eq_num + 1; jeq = eq_num;
eq_num = eq_num + 1; meq = eq_num;
%
% force equations
A(ieq,[ifc1,ifd1]) = [-1.0 1.0]; b(ieq) = m4*ag4(1)/gc - m4*g_by_gc(1);
A(jeq,[ifc2,ifd2]) = [-1.0 1.0]; b(jeq) = m4*ag4(2)/gc - m4*g_by_gc(2);
%
% moment equation
A(meq,ifd1) = r(4)*(-ndc(2)); A(meq,ifd2) = r(4)*(ndc(1));
A(meq, imd) = 1.0;
b(meq) = i4*a4/gc + m4*rg4c*cross_2d(ndc,ag4)/gc - rg4c*m4*cross_2d(ndc,g_by_gc);
%
% finally, ready to solve
%
f = A\b;
%
end; % end of force and moment calculations
```

```
function ang=law_cos_ang(r2c,r2a,r2b,rab)
%
% use law of cosines to compute an angle in a triangle
%
% acos returns in range 0 to pi
%
temp = (r2c - r2a - r2b)/(-2.0*rab);
% check against numerical errors
if (temp <= -1.0)
     ang = pi;
elseif (temp >= 1.0)
     ang = 0;
else
     ang = acos(temp);
```

```
function [len, len2]=law_cos_len(r2a,r2b,rab,th)
%
% use law of cosines to compute an length in a triangle
%
len2 = (r2a + r2b - 2.0*rab*cos(th));
len = sqrt(len2);
```

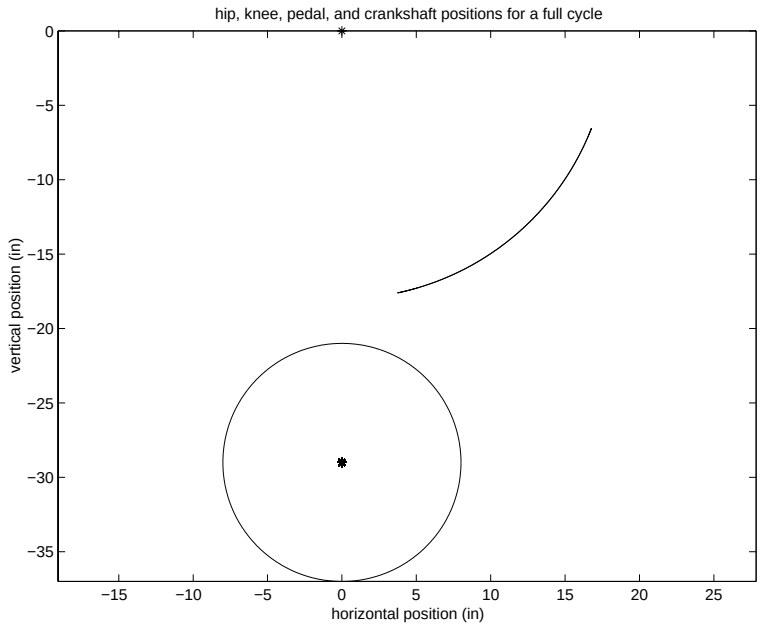

Figure 1: positions

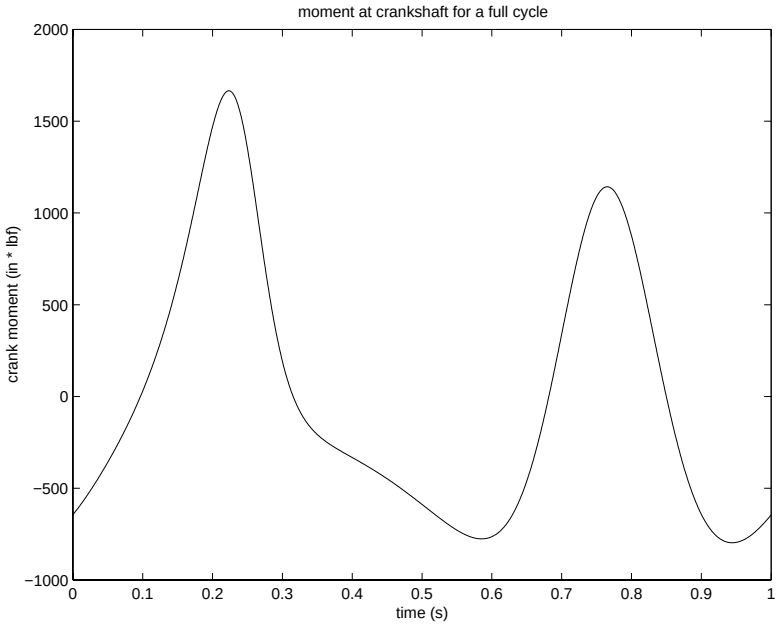

Figure 2: crank moment

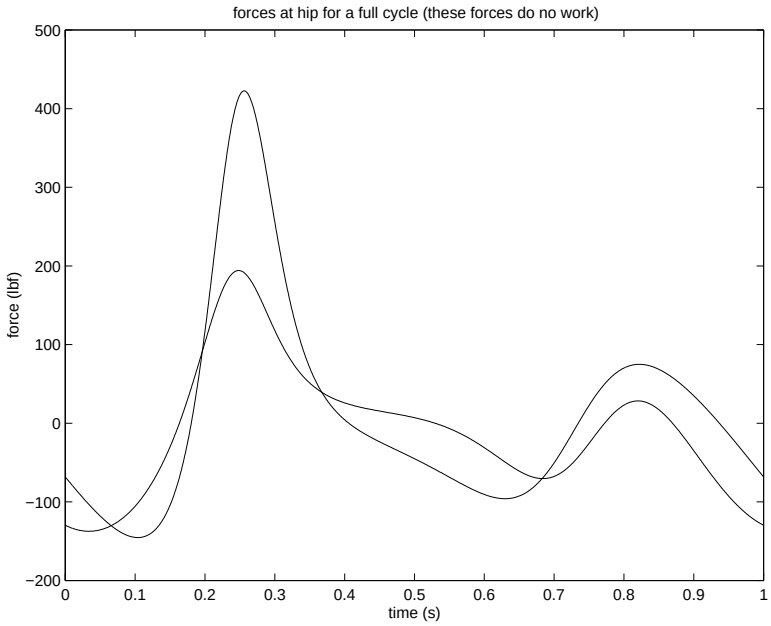

Figure 3: forces at hip