Clay Freeman

ME 493: Intermediate Dynamics

Dr. Fields - Spring 2019

Exam II

9 May, 2019

Problem 1a: Horizontal Disk With Bar and Spring - Derive Kane's Equations

```
   (1) % ME 493: Intermediate Dynamics
   (2) % Final Exam Problem 1
   (3) NewtonianFrame N
   (4) RigidBody D, A
   (5) Constant L, rD, g, k, Id, Jx, Jy
   (6) Point R(A), P(A), Q()
   (7) Variable Q1', Q2', U1', U2'
   (8) SetGeneralizedSpeed(U1, U2)
   (9) D.SetMass(Md)
   (10) D.SetInertia(Dcm, Id*Nx>*Nx> + 0*Ny>*Ny> + 0*Nz>*Nz>)
   (11) % D.SetInertia(Dcm, Md*rD^2/4*(Dx>*Dx> + Dy>*Dy> + 2*Dz>*Dz>))
   (12) A.SetMass(Ma)
   (13) A.SetInertia(Acm, (Jx*Ax>*Ax> + Jy*Ay>*Ay>+ Jy*Az>*Az>))
   (14) rD = 0.7*L
-> (15) rD = 0.7*L

   (16) g = 9.81
-> (17) g = 9.81

   (18) %%% Kinematical Equations %%%
   (19) Q1' = U1
-> (20) Q1' = U1

   (21) Q2' = U2
-> (22) Q2' = U2

   (23) %%% Rotations %%%
   (24) D.RotateX(N, -Q1)
-> (25) D_N = [1, 0, 0;  0, cos(Q1), -sin(Q1);  0, sin(Q1), cos(Q1)]
-> (26) w_D_N> = -U1*Nx>
-> (27) alf_D_N> = -U1'*Nx>

   (28) A.RotateY(D, Q2)
-> (29) A_D = [cos(Q2), 0, -sin(Q2);  0, 1, 0;  sin(Q2), 0, cos(Q2)]
-> (30) w_A_D> = U2*Ay>
-> (31) w_A_N> = -cos(Q2)*U1*Ax> + U2*Ay> - sin(Q2)*U1*Az>
-> (32) alf_A_D> = U2'*Ay>
-> (33) alf_A_N> = (sin(Q2)*U1*U2-cos(Q2)*U1')*Ax> + U2'*Ay> + (-cos(Q2)*U1*U2-
        sin(Q2)*U1')*Az>

   (34) %%% Translations %%%
   (35) % R.SetPosition(Dcm, rD*Dy>)
   (36) Ao.SetPosition(Dcm, -rD*Dy>)
```

```
-> (37) p_Dcm_Ao> = -rD*Dy>

   (38) Acm.SetPosition(Ao, 0.5*L*Ax>)
-> (39) p_Ao_Acm> = 0.5*L*Ax>

   (40) P.SetPosition(Ao, L*Ax>)
-> (41) p_Ao_P> = L*Ax>

   (42) Q.SetPosition(Dcm, 3*L*Dx>)
-> (43) p_Dcm_Q> = 3*L*Dx>

   (44) %%% Velocities %%%
   (45) Acm.SetVelocityAcceleration(N, 0.7*U1*Ax> + U1*Ay> + U2*Az>)
-> (46) v_Acm_N> = 0.7*U1*Ax> + U1*Ay> + U2*Az>
-> (47) a_Acm_N> = (U2^2+sin(Q2)*U1^2+0.7*U1')*Ax> + (cos(Q2)*U1*U2+U1'-0.7*sin
        (Q2)*U1^2)*Ay> + (U2'-0.7*U1*U2-cos(Q2)*U1^2)*Az>

   (48) Dcm.SetVelocityAcceleration(N, 0>)
-> (49) v_Dcm_N> = 0>
-> (50) a_Dcm_N> = 0>

   (51) Q.SetVelocityAcceleration(N, 0>)
-> (52) v_Q_N> = 0>
-> (53) a_Q_N> = 0>

   (54) %%% Partial Velocities %%%
   (55) VD_Partials =  Dcm.GetPartialVelocity(N)
-> (56) VD_Partials = [0>;  0>]

   (57) VDcm_Partial1> = VD_Partials[1]
-> (58) VDcm_Partial1> = 0>

   (59) VDcm_Partial2> = VD_Partials[2]
-> (60) VDcm_Partial2> = 0>

   (61) VA_Partials = Acm.GetPartialVelocity(N)
-> (62) VA_Partials = [0.7*Ax> + Ay>;  Az>]

   (63) VAcm_Partial1> = VA_Partials[1]
-> (64) VAcm_Partial1> = 0.7*Ax> + Ay>

   (65) VAcm_Partial2> = VA_Partials[2]
-> (66) VAcm_Partial2> = Az>

   (67) WD_Partial1> = D.GetPartialAngularVelocity(N, U1)
```

```
-> (68) WD_Partial1> = -Nx>

   (69) WD_Partial2> = D.GetPartialAngularVelocity(N, U2)
-> (70) WD_Partial2> = 0>

   (71) WA_Partial1> = A.GetPartialAngularVelocity(N, U1)
-> (72) WA_Partial1> = -cos(Q2)*Ax> - sin(Q2)*Az>

   (73) WA_Partial2> = A.GetPartialAngularVelocity(N, U2)
-> (74) WA_Partial2> = Ay>

   (75) %%% Forces %%%
   (76) System.AddForceGravity( g*Nx> )
-> (77) Force_Acm> = Ma*g*Nx>
-> (78) Force_Dcm> = Md*g*Nx>

   (79) P.AddForceSpring(Q, k, L)
-> (80) Force_P_Q> = -k*L*(1-L/sqrt(rD^2+10*L^2-6*L^2*cos(Q2)))*Ax> + 3*k*L*(1-
        L/sqrt(rD^2+10*L^2-6*L^2*cos(Q2)))*Dx> + k*rD*(1-L/sqrt(rD^2+10*L^2-6*L^2
        *cos(Q2)))*Dy>

   (81) %%% Effective Moments %%%
   (82) NMD_Dcm> = dot(I_D_Dcm>>, alf_D_N>) + cross(w_D_N>, dot(I_D_Dcm>>,
w_D_N>))
-> (83) NMD_Dcm> = -Id*U1'*Nx>

   (84) NMA_Acm> = dot(I_A_Acm>>, alf_A_N>) + cross(w_A_N>, dot(I_A_Acm>>,
w_A_N>))
-> (85) NMA_Acm> = Jx*(sin(Q2)*U1*U2-cos(Q2)*U1')*Ax> +
(Jx*sin(Q2)*cos(Q2)*U1^2
        +Jy*U2'-Jy*sin(Q2)*cos(Q2)*U1^2)*Ay> + (Jx*cos(Q2)*U1*U2-2*Jy*cos(Q2)*
        U1*U2-Jy*sin(Q2)*U1')*Az>

   (86) %%% Kane's Equations %%%
   (87) F1 = dot(A.GetResultantForce(), VAcm_Partial1>) + dot(D.GetResultantForce(),
VDcm_Partial1>) + dot(A.GetResultantMoment(Acm), WA_Partial1>) +
dot(D.GetResultantMoment(Dcm), WD_Partial1>)
-> (88) F1 = k*rD*(1-L/sqrt(rD^2+10*L^2-6*L^2*cos(Q2))) + 0.7*cos(Q2)*(Ma*g+3*k
        *L*(1-L/sqrt(rD^2+10*L^2-6*L^2*cos(Q2)))) - 0.7*k*L*(1-L/sqrt(rD^2+10*L^2
        -6*L^2*cos(Q2))) - 0.5*k*L*rD*sin(Q2)*(1-L/sqrt(rD^2+10*L^2-6*L^2*cos(
        Q2)))

   (89) F1_N = dot(Ma*a_Acm_N>, VAcm_Partial1>) + dot(Md*a_Dcm_N>,
VDcm_Partial1>) + dot(NMA_Acm>, WA_Partial1>) + dot(NMD_Dcm>, WD_Partial1>)
-> (90) F1_N = Id*U1' + 0.7*Ma*(U2^2+1.428571*cos(Q2)*U1*U2+2.128571*U1')
```

```
      + Jy*sin(Q2)*(2*cos(Q2)*U1*U2+sin(Q2)*U1') - Jx*cos(Q2)*(2*sin(Q2)*U1*
      U2-cos(Q2)*U1')

  (91) F2 = dot(A.GetResultantForce(), VAcm_Partial2>) + dot(D.GetResultantForce(),
VDcm_Partial2>) + dot(A.GetResultantMoment(Acm), WA_Partial2>) +
dot(D.GetResultantMoment(Dcm), WD_Partial2>)
-> (92) F2 = sin(Q2)*(Ma*g+3*k*L*(1-L/sqrt(rD^2+10*L^2-6*L^2*cos(Q2)))-1.5*k*L^2
      *(1-L/sqrt(rD^2+10*L^2-6*L^2*cos(Q2))))

  (93) F2_N = dot(Ma*a_Acm_N>, VAcm_Partial2>) + dot(Md*a_Dcm_N>,
VDcm_Partial2>) + dot(NMA_Acm>, WA_Partial2>) + dot(NMD_Dcm>, WD_Partial2>)
-> (94) F2_N = Jx*sin(Q2)*cos(Q2)*U1^2 + Jy*U2' - Jy*sin(Q2)*cos(Q2)*U1^2
      - 0.7*Ma*(U1*U2+1.428571*cos(Q2)*U1^2-1.428571*U2')

  (95) zero[1] = explicit(F1 - F1_N)
-> (96) zero[1] = 0.1428869*k*L^2*sin(Q2)*(-2.44949+L/sqrt(-L^2*(-1.748333+cos(
      Q2)))) + 0.8573214*cos(Q2)*(8.009831*Ma-k*L*(-2.44949+L/sqrt(-L^2*(-1.748333
      +cos(Q2))))) + Jx*cos(Q2)*(2*sin(Q2)*U1*U2-cos(Q2)*U1') - Id*U1' - 0.7*
      Ma*(U2^2+1.428571*cos(Q2)*U1*U2+2.128571*U1') - Jy*sin(Q2)*(2*cos(Q2)*
      U1*U2+sin(Q2)*U1')

  (97) zero[2] = explicit(F2 - F2_N)
-> (98) zero[2] = Jy*sin(Q2)*cos(Q2)*U1^2 + 0.7*Ma*(U1*U2+1.428571*cos(Q2)*U1^2
      -1.428571*U2') - 0.6123724*sin(Q2)*(2*k*L*(-2.44949+L/sqrt(-L^2*(-1.748333
      +cos(Q2))))-16.01966*Ma-k*L^2*(-2.44949+L/sqrt(-L^2*(-1.748333+cos(Q2)))))
      - Jx*sin(Q2)*cos(Q2)*U1^2 - Jy*U2'

  (99) zero_auto = System.GetDynamicsKane()
-> (100) zero_auto[1] = (1.49*Ma+Id+Jx*cos(Q2)^2+Jy*sin(Q2)^2)*U1' -
0.7*Ma*g*cos(Q2)
       - 0.5*k*(4.2*L*cos(Q2)-1.4*L-rD*(-2+L*sin(Q2)))*(1-L/sqrt(rD^2+10*L^2-
       6*L^2*cos(Q2))) - 0.7*U2*(2.857143*Jx*sin(Q2)*cos(Q2)*U1-2.857143*Jy*
       sin(Q2)*cos(Q2)*U1-Ma*(U2+1.428571*cos(Q2)*U1))

-> (101) zero_auto[2] = (Ma+Jy)*U2' - sin(Q2)*(Ma*g-1.5*k*L*(-2+L)*(1-L/sqrt(rD^2
       +10*L^2-6*L^2*cos(Q2)))) - 0.7*U1*(Ma*(U2+1.428571*cos(Q2)*U1)-1.428571
       *(Jx-Jy)*sin(Q2)*cos(Q2)*U1)

  (102) check = explicit(zero + zero_auto)
-> (103) check = [0;  0]

  (104) % ODE(zero, U1', U2') Exam1_matlab.m
```

Problem 1b: Would it have been easier to use generalized speeds defined by ${}^{N}\overline{V}^{Acm} = U_1\hat{a}_x + U_2\hat{a}_y + U_3\hat{a}_z$?

No. All of the degrees of freedom are accounted for using Q_1 and Q_2. Adding a third variable and associated generalized speed would add a constraint between variables. This is supported by the geometry governing where $\hat{a}_z$ is able to move since it is attached to the disk at point R.

Problem 1c: Does your answer for the velocity of ${}^{N}\overline{V}^{Acm}$ make sense?

Yes, the position of the center of mass for bar A in the x and y direction is heavily dependent on the rotation angle of disk D due to the attachment point at R. If we think generally about how this object would behave in the real world, we could spin the disk and the spring would apply significant velocity on the bar in the z-direction but the remaining velocities would be set by the disk above.

Problem 2a: Double Arm Welded to Disk on Slope - Derive Kane's Equations

```
   (1) % ME 493: Intermediate Dynamics
   (2) % Final Exam Problem 2
   (3) NewtonianFrame N
   (4) RigidBody A, B, C
   (5) RigidFrame D
   (6) Point P(), Q()
   (7) Constant L, rB, g, tB
   (8) Variable Q{1:3}', U{1:3}'
   (9) A.SetMass(Ma)
   (10) A.SetInertia(Acm, 0, 0, IA)
   (11) B.SetMass(Mb)
   (12) B.SetInertia(Bcm, 0, 0, jB)
   (13) % B.SetInertia(Bcm, (0*Bx>*Bx> + 0*By>*By> + jB*Bz>*Bz>))
   (14) % B.SetInertia(Bcm, (0*Nx>*Nx> + 0*Ny>*Ny> + jB*Nz>*Nz>))
   (15) % B.SetInertia(Bcm, Mb*rB^2/4*(Bx>*Bx> + By>*By> + 2*Bz>*Bz>))
   (16) C.SetMass(Mc)
   (17) C.SetInertia(Ccm, 0, IC, 0)
   (18) SetGeneralizedSpeed(U1, U2, U3)
   (19) %%% Kinematical Equations %%%
   (20) Q1' = U1
-> (21) Q1' = U1

   (22) Q2' = U2
-> (23) Q2' = U2

   (24) Q3' = U3
-> (25) Q3' = U3

   (26) %%% Rotations %%%
   (27) A.RotateZ(N, tB+Q3)
-> (28) A_N = [cos(tB+Q3), sin(tB+Q3), 0;  -sin(tB+Q3), cos(tB+Q3), 0;  0, 0, 1]
-> (29) w_A_N> = U3*Az>
-> (30) alf_A_N> = U3'*Az>

   (31) B.RotateZ(N, tB+Q3)
-> (32) B_N = [cos(tB+Q3), sin(tB+Q3), 0;  -sin(tB+Q3), cos(tB+Q3), 0;  0, 0, 1]
-> (33) w_B_N> = U3*Bz>
-> (34) alf_B_N> = U3'*Bz>

   (35) C.Rotate(A, BodyY, -Q1)
-> (36) C_A = [cos(Q1), 0, sin(Q1);  0, 1, 0;  -sin(Q1), 0, cos(Q1)]
-> (37) w_C_A> = -U1*Cy>
-> (38) w_C_N> = sin(Q1)*U3*Cx> - U1*Cy> + cos(Q1)*U3*Cz>
```

```
-> (39) alf_C_A> = -U1'*Cy>
-> (40) alf_C_N> = (cos(Q1)*U1*U3+sin(Q1)*U3')*Cx> - U1'*Cy> + (cos(Q1)*U3'-sin
       (Q1)*U1*U3)*Cz>

   (41) D.RotateZ(N, tB)
-> (42) D_N = [cos(tB), sin(tB), 0;  -sin(tB), cos(tB), 0;  0, 0, 1]
-> (43) w_D_N> = 0>
-> (44) alf_D_N> = 0>

   (45) %%% Translations %%%
   (46) Ao.Translate(No, Q2*By>)
-> (47) p_No_Ao> = Q2*By>
-> (48) v_Ao_N> = -Q2*U3*Bx> + U2*By>
-> (49) a_Ao_N> = (-2*U2*U3-Q2*U3')*Bx> + (U2'-Q2*U3^2)*By>

   (50) Do.Translate(Ao, 0>)
-> (51) p_Ao_Do> = 0>
-> (52) v_Do_N> = -Q2*U3*Bx> + U2*By>
-> (53) a_Do_N> = (-2*U2*U3-Q2*U3')*Bx> + (U2'-Q2*U3^2)*By>

   (54) Bcm.Translate(Ao, 0>)
-> (55) p_Ao_Bcm> = 0>
-> (56) v_Bcm_N> = -Q2*U3*Bx> + U2*By>
-> (57) a_Bcm_N> = (-2*U2*U3-Q2*U3')*Bx> + (U2'-Q2*U3^2)*By>

   (58) Acm.Translate(Ao, L*Ax>)
-> (59) p_Ao_Acm> = L*Ax>
-> (60) v_Acm_N> = L*U3*Ay> - Q2*U3*Bx> + U2*By>
-> (61) a_Acm_N> = -L*U3^2*Ax> + L*U3'*Ay> + (-2*U2*U3-Q2*U3')*Bx> +
(U2'-Q2*U3^2)*By>

   (62) Q.Translate(Ao, 2*L*Ax>)
-> (63) p_Ao_Q> = 2*L*Ax>
-> (64) v_Q_N> = 2*L*U3*Ay> - Q2*U3*Bx> + U2*By>
-> (65) a_Q_N> = -2*L*U3^2*Ax> + 2*L*U3'*Ay> + (-2*U2*U3-Q2*U3')*Bx> +
(U2'-Q2*U3^2)*By>

   (66) Co.Translate(Q, -1.5*L*Cz>)
-> (67) p_Q_Co> = -1.5*L*Cz>
-> (68) v_Co_N> = 1.5*L*(1.333333+sin(Q1))*U3*Ay> - Q2*U3*Bx> + U2*By> +
1.5*L*U1*Cx>
-> (69) a_Co_N> = -2*L*U3^2*Ax> + 1.5*L*(2*cos(Q1)*U1*U3+1.333333*U3'+sin(Q1)*
       U3')*Ay> + (-2*U2*U3-Q2*U3')*Bx> + (U2'-Q2*U3^2)*By> - 1.5*L*(sin(Q1)*
       cos(Q1)*U3^2-U1')*Cx> + 1.5*L*(U1^2+sin(Q1)^2*U3^2)*Cz>
```

```
   (70) Ccm.Translate(Q, -0.75*L*Cz>)
-> (71) p_Q_Ccm> = -0.75*L*Cz>
-> (72) v_Ccm_N> = 0.75*L*(2.666667+sin(Q1))*U3*Ay> - Q2*U3*Bx> + U2*By> +
0.75*L*U1*Cx>
-> (73) a_Ccm_N> = -2*L*U3^2*Ax> +
0.75*L*(2*cos(Q1)*U1*U3+2.666667*U3'+sin(Q1)
      *U3')*Ay> + (-2*U2*U3-Q2*U3')*Bx> + (U2'-Q2*U3^2)*By> - 0.75*L*(sin(Q1)
      *cos(Q1)*U3^2-U1')*Cx> + 0.75*L*(U1^2+sin(Q1)^2*U3^2)*Cz>

   (74) P.Translate(Do, -rB*Bx>)
-> (75) p_Do_P> = -rB*Bx>
-> (76) v_P_N> = -Q2*U3*Bx> + (U2-rB*U3)*By>
-> (77) a_P_N> = (rB*U3^2-2*U2*U3-Q2*U3')*Bx> + (U2'-Q2*U3^2-rB*U3')*By>

   (78) %%% Constraints %%%
   (79) Dependent[1] = dot(V_P_N>, By>)
-> (80) Dependent[1] = U2 - rB*U3

   (81) Constrain(Dependent[U2])
-> (82) U2 = rB*U3
-> (83) U2' = rB*U3'

   (84) %%% Forces %%%
   (85) System.AddForceGravity(g*Nx>)
-> (86) Force_Acm> = Ma*g*Nx>
-> (87) Force_Bcm> = Mb*g*Nx>
-> (88) Force_Ccm> = Mc*g*Nx>

   (89) zero = System.GetDynamicsKane()
-> (90) zero[1] = 0.5625*(1.777778*IC+Mc*L^2)*U1' - 0.5625*Mc*L*cos(Q1)*(1.333333
      *g*cos(tB+Q3)+U3*(2.666667*U2+2.666667*L*U3+L*sin(Q1)*U3)) - 0.75*Mc*L*
      Q2*cos(Q1)*U3'

-> (91) zero[2] = 0.75*g*(1.333333*Mb*(rB*sin(tB+Q3)+Q2*cos(tB+Q3))+1.333333*
      Ma*(L*sin(tB+Q3)+rB*sin(tB+Q3)+Q2*cos(tB+Q3))+Mc*(1.333333*rB*sin(tB+
      Q3)+1.333333*Q2*cos(tB+Q3)+L*sin(tB+Q3)*(2.666667+sin(Q1)))) + 2*Ma*Q2*
      U2*U3 + 2*Mb*Q2*U2*U3 +
0.75*Mc*(2.666667*Q2*U2*U3+L*Q2*sin(Q1)*U1^2+1.5
      *L^2*cos(Q1)*(2.666667+sin(Q1))*U1*U3) + 0.5625*(1.777778*IA+1.777778*

jB+1.777778*Mb*Q2^2+1.777778*Ma*(L^2+2*L*rB+Q2^2)+Mc*(1.777778*Q2^2+L^2
      *(2.666667+sin(Q1))^2)+1.777778*rB*(Ma*rB+Mb*rB+Mc*(rB+1.5*L*(2.666667+
      sin(Q1)))))*U3' - rB*U3*(Ma*Q2*U3+Mb*Q2*U3+Mc*(Q2*U3-1.5*L*cos(Q1)*U1))
      - 0.75*Mc*L*Q2*cos(Q1)*U1'
```

(92) Input tFinal = 40.0, tStep = 0.1, absError = 1.0E-07
(93) Input tB = 0.1745 rad, g = 9.81 m/s^2, rB = 0.3 m
(94) Input Mb = 1.0 kg, jB = 0.06 kg*m^2, L = 0.8 m
(95) Input Ma = 0.5 kg, IA = 0.12 kg*m^2, Mc = 0.3 kg
(96) Input IC = 0.04 kg*m^2, Q1 = 0.2618 rad, Q2 = 0.02 m
(97) Input Q3 = 0.2618 rad, U1 = 0.0873 rad/s
(98) Input U3 = 0.0 rad/s
(99) OutputPlot t sec, Q1 deg, Q2 m, Q3 deg, U1 rad/s, U2 m/s, U3 rad/s
(100) ODE(zero, U1', U3') Exam2_matlab.m

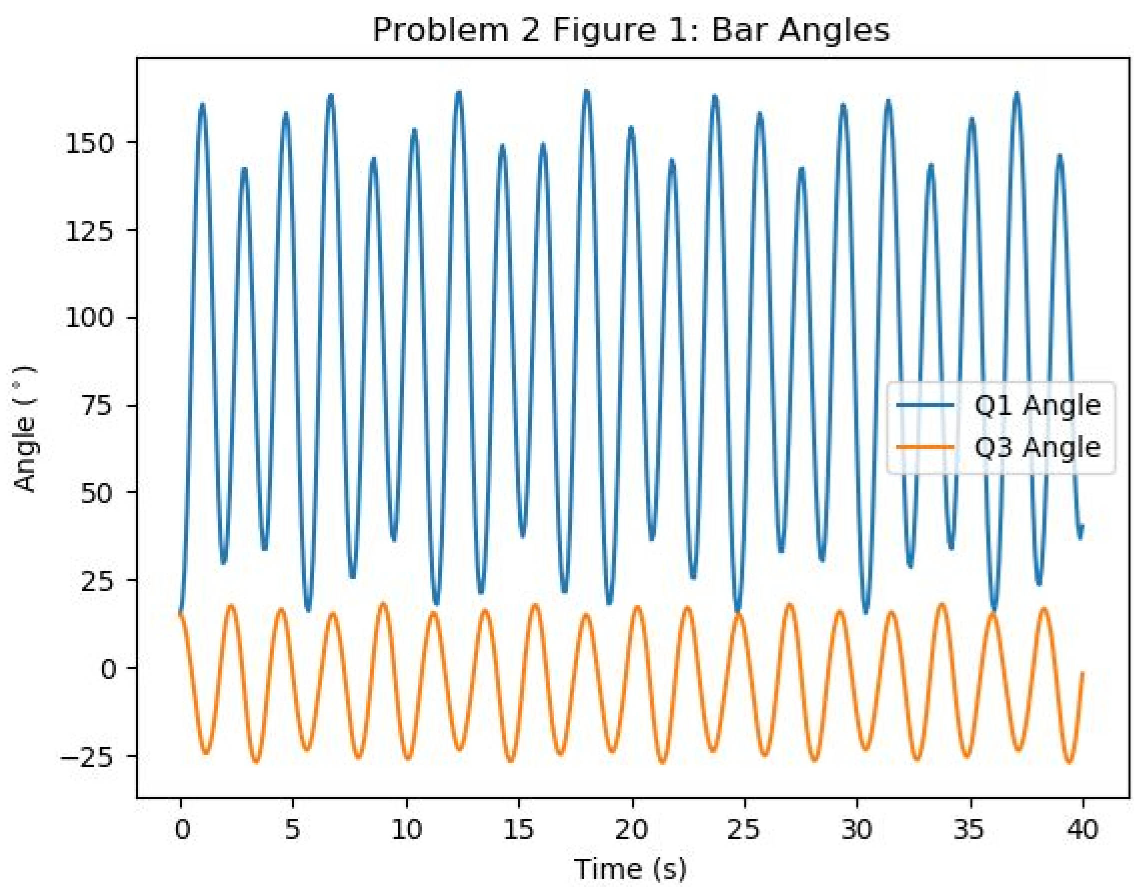
Problem 2b: How many degrees of freedom does the system have?

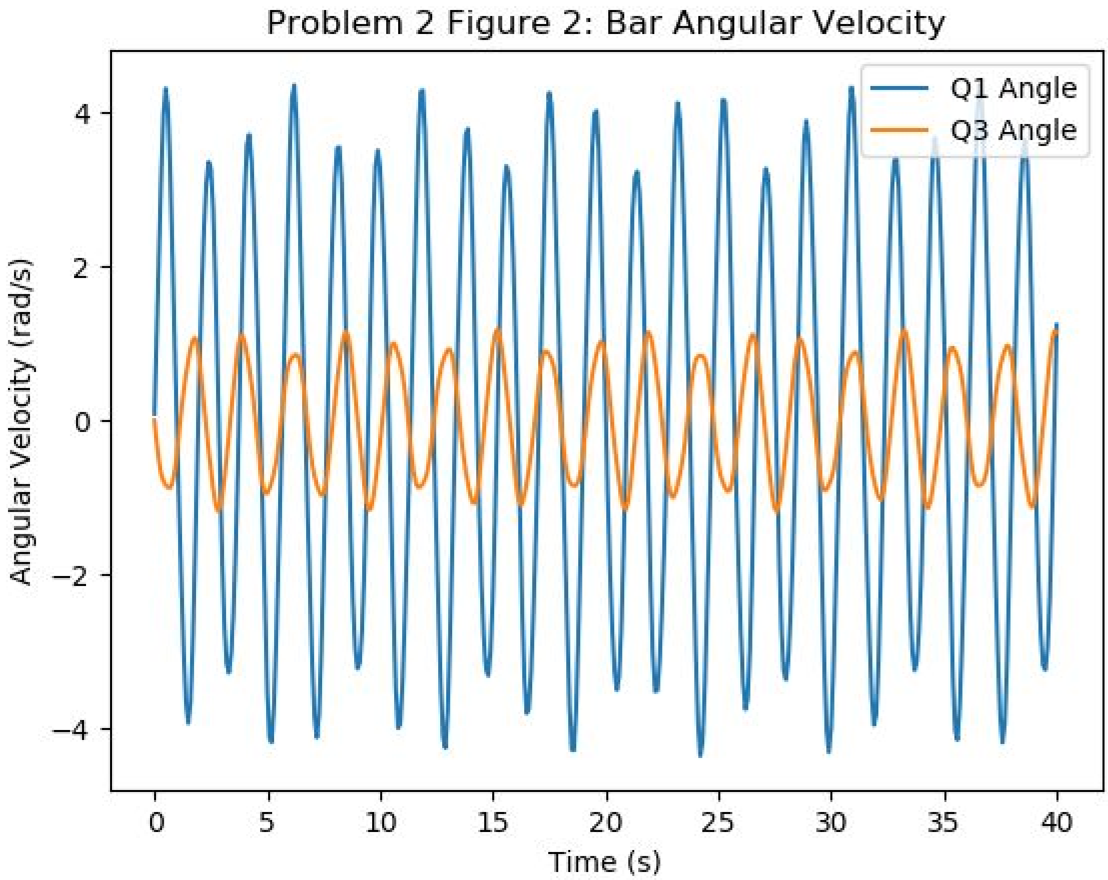
2. Since the bar A is welded to disk B, Q_2 and Q_3 are redundant. It is possible to derive the position or the angle of the top part of the arm using only one of these variables. One of the challenges in this problem was figuring out that I needed to set up an additional RigidFrame, D to describe the constrained point P at the interface between the sloped surface and the disk. The frame needed to follow the slope of the surface without turning along with the rigid bodies.

Problem 2c: If the angle θ was not constant, would you still be able to solve the problem using Kane's method?

Yes, the additional variable would change the rotation dynamics of the rigid frame and the overall behavior of the arms, but Kane's method could still handle this problem. We would need to be careful about deciding how to define the rotation of the other rigid bodies A and B.

Problem 2d: Simulation Plot




Problem 2 Figure 2: Bar Angular Velocity
Q1 Angle
Q3 Angle
Angular Velocity (rad/s)
Time (s)
4
2
0
−2
−4
0
5
10
15
20
25
30
35
40

Problem 3: Comeback Can

```
   (1) % ME 493: Intermediate Dynamics
   (2) % Final Exam Problem 3
   (3) NewtonianFrame N
   (4) RigidBody A, B
   (5) Constant rA, rB, g, k
   (6) Point P()
   (7) Variable Q{1:3}', U{1:3}'
   (8) A.SetMass(mA)
   (9) B.SetMass(mB)
   (10) A.SetInertia(Acm, IA, 0, 0)
   (11) B.SetInertia(Bcm, IB, 0, 0)
   (12) % A.SetInertia(Acm, mA*rA^2*(Ax>*Ax>))
   (13) % B.SetInertia(Bcm, mB*rB^2/4*(2*Bx>*Bx> + By>*By> + Bz>*Bz>))
   (14) SetGeneralizedSpeed(U1, U2, U3)
   (15) %%% Kinematical Equations %%%
   (16) Q1' = U1
-> (17) Q1' = U1

   (18) Q2' = U2
-> (19) Q2' = U2

   (20) Q3' = U3
-> (21) Q3' = U3

   (22) %%% Rotations %%%
   (23) A.RotateX(N, Q1)
-> (24) A_N = [1, 0, 0;  0, cos(Q1), sin(Q1);  0, -sin(Q1), cos(Q1)]
-> (25) w_A_N> = U1*Ax>
-> (26) alf_A_N> = U1'*Ax>

   (27) B.RotateX(A, -Q3)
-> (28) B_A = [1, 0, 0;  0, cos(Q3), -sin(Q3);  0, sin(Q3), cos(Q3)]
-> (29) w_B_A> = -U3*Bx>
-> (30) w_B_N> = (U1-U3)*Bx>
-> (31) alf_B_A> = -U3'*Bx>
-> (32) alf_B_N> = (U1'-U3')*Bx>

   (33) %%% Translations %%%
   (34) P.SetPosition(Acm, -rA*Nz>)
-> (35) p_Acm_P> = -rA*Nz>

   (36) Bcm.SetPosition(Acm, -rB*Az>)
-> (37) p_Acm_Bcm> = -rB*Az>
```

```
   (38) %%% Velocities %%%
   (39) P.SetVelocity(N, U2*Ny>)
-> (40) v_P_N> = U2*Ny>

   (41) Acm.SetVelocityAcceleration(N, 0>)
-> (42) v_Acm_N> = 0>
-> (43) a_Acm_N> = 0>

   (44) Bcm.SetVelocityAcceleration(N, 0>)
-> (45) v_Bcm_N> = 0>
-> (46) a_Bcm_N> = 0>

   (47) %%% Constraints %%%
   (48) Dependent[1] = dot(v_P_N>, Ny>)
-> (49) Dependent[1] = U2

   (50) Constrain(Dependent[U2])
-> (51) U2 = 0
-> (52) U2' = 0

   (53) %%% Forces %%%
   (54) System.AddForceGravity( -g*Nz> )
-> (55) Force_Acm> = -mA*g*Nz>
-> (56) Force_Bcm> = -mB*g*Nz>

   (57) B.AddTorque(A, -2*k*Q3*Ax>)
-> (58) Torque_B_A> = -2*k*Q3*Ax>

   (59) zero = System.GetDynamicsKane()
-> (60) zero = [(IA+IB)*U1' - IB*U3';  IB*U3' - 2*k*Q3 - IB*U1']

   (61) % ODE(zero, U1', U3') Exam3_matlab.m
```

Problem 4a: Restrained Double Pendulum

```
   (1) % ME 493: Intermediate Dynamics
   (2) % Final Exam Problem 4
   (3) NewtonianFrame N
   (4) RigidFrame A, B
   (5) Constant lA, lB, g, k, lko, bA, bB
   (6) Particle P, Q
   (7) Variable Q{1:2}', U{1:2}'
   (8) P.SetMass(mP)
   (9) Q.SetMass(mQ)
   (10) SetGeneralizedSpeed(U1, U2)
   (11) %%% Kinematical Equations %%%
   (12) Q1' = U1
-> (13) Q1' = U1

   (14) Q2' = U2
-> (15) Q2' = U2

   (16) %%% Rotations %%%
   (17) A.RotateZ(N, Q1)
-> (18) A_N = [cos(Q1), sin(Q1), 0;  -sin(Q1), cos(Q1), 0;  0, 0, 1]
-> (19) w_A_N> = U1*Az>
-> (20) alf_A_N> = U1'*Az>

   (21) B.RotateZ(A, Q2)
-> (22) B_A = [cos(Q2), sin(Q2), 0;  -sin(Q2), cos(Q2), 0;  0, 0, 1]
-> (23) w_B_A> = U2*Bz>
-> (24) w_B_N> = (U1+U2)*Az>
-> (25) alf_B_A> = U2'*Bz>
-> (26) alf_B_N> = (U1'+U2')*Az>

   (27) %%% Velocities %%%
   (28) Ao.SetVelocityAcceleration(N, 0>)
-> (29) v_Ao_N> = 0>
-> (30) a_Ao_N> = 0>

   (31) %%% Translations %%%
   (32) P.Translate(Ao, lA*Ax>)
-> (33) p_Ao_P> = lA*Ax>
-> (34) v_P_N> = lA*U1*Ay>
-> (35) a_P_N> = -lA*U1^2*Ax> + lA*U1'*Ay>

   (36) Q.Translate(P, lB*Bx>)
-> (37) p_P_Q> = lB*Bx>
```

```
-> (38) v_Q_N> = lA*U1*Ay> + lB*(U1+U2)*By>
-> (39) a_Q_N> = -lA*U1^2*Ax> + lA*U1'*Ay> - lB*(U1+U2)^2*Bx> + lB*(U1'+U2')*By>

   (40) %%% Forces %%%
   (41) System.AddForceGravity( g*Nx> )
-> (42) Force_P> = mP*g*Nx>
-> (43) Force_Q> = mQ*g*Nx>

   (44) Q.AddForceSpring(Ao, k, lko)
-> (45) Force_Q_Ao> = -k*lA*(1-lko/sqrt(lA^2+lB^2+2*lA*lB*cos(Q2)))*Ax> - k*lB*
        (1-lko/sqrt(lA^2+lB^2+2*lA*lB*cos(Q2)))*Bx>

   (46) A.AddTorque( -bA*U1*Nz> )
-> (47) Torque_A> = -bA*U1*Nz>

   (48) B.AddTorque( -bB*U2*Nz> )
-> (49) Torque_B> = -bB*U2*Nz>

   (50) zero = System.GetDynamicsKane()
-> (51) zero[1] = mP*g*lA*sin(Q1) + mQ*g*(lA*sin(Q1)+lB*sin(Q1+Q2)) + bA*U1
        + bB*U2 + mQ*lA*lB*sin(Q2)*(U1^2-(U1+U2)^2) + mQ*lB*(lB+lA*cos(Q2))*U2'
        + (mP*lA^2+mQ*(lA^2+lB^2+2*lA*lB*cos(Q2)))*U1'

-> (52) zero[2] = mQ*g*lB*sin(Q1+Q2) + bB*U2 + mQ*lA*lB*sin(Q2)*U1^2 +
mQ*lB^2*U2'
        + mQ*lB*(lB+lA*cos(Q2))*U1' - k*lA*lB*sin(Q2)*(1-lko/sqrt(lA^2+lB^2+2*
        lA*lB*cos(Q2)))

   (53) Input tFinal = 30, tStep = 0.1, absError = 1.0E-07
   (54) Input bA = 100 N*s/m, bB = 100 N*s/m, g = 9.81 m/s^2
   (55) Input k = 250 N/m, lA = 1.5 m, lB = 2.0 m
   (56) Input lko = 1.0 m, mP = 10.0 kg, mQ = 15.0 kg
   (57) Input Q1 = 1.5708 rad, Q2 = 1.5708 rad
   (58) Input U1 = 0.0 rad, U2 = 0.0 rad
   (59) OutputPlot t sec, Q1 rad, Q2 rad, U1 rad/s, U2 rad/s
```

Problem 4b: Simulation Plot