Computational Dynamics Homework 4

Jackson Wills

February 20, 2020

This is my first go at using LaTeX, so I would appreciate any and all advice and criticisms. I decided to just give you my code verbatim, and then answer your questions formatted well. I hope that is okay.

```
using SymPy
using Plots

@vars t f m1 m2 k1 k2 c1 c2 z1 z2 z3 z4 u
x1 = SymFunction("x1")(t)
x2 = SymFunction("x2")(t)
x1 = SymFunction("x1")(t)
x1 = SymFunction("x1")(t)
x2 = SymFunction("x2")(t)
x2 = SymFunction("x2")(t)

diff(x1,t)
diff(x1,t,t)
diff(diff(x1,t),t)

#defining system
q = [x1;x2]
Q = [0;f] #right hand side of equation

# Determine kinetic energy, potential energy, and dissipation function
T = 1//2*m1*(diff(x1,t))^2 + 1//2*m2*(diff(x2,t))^2 |> subs(diff(x1,t),x1) |> subs(diff(x2,t)

V = 1//2*k1*x1^2 + 1//2*k2*(x2-x1)^2
```

```
D = 1//2*c1*(diff(x1,t))^2 + 1//2*c2*(diff(x2,t)-diff(x1,t))^2 |> subs(diff(x1,t);x1) |>
subs(diff(x2,t);x2)

L = T-V

#Left Hand Side
Q1 = diff(diff(L;x1),t) - diff(L,x1) + diff(D;x1)|> subs(diff(x1,t);x1) |> subs(diff(x2,t);x2
Q2 = diff(diff(L;x2),t) - diff(L,x2) + diff(D;x2) |> subs(diff(x1,t);x1)|> subs(diff(x2,t);x2

# state the Lagrangian equations of motion
eqn1 = Q[1] - Q1 #this equals zero
eqn2 = Q[2] - Q2 #this equals zero




## NOW DOING HAMILTONIAN
P1 = diff(L;x1)
P2 = diff(L;x2)
@vars p1 p2
zero1 = P1 - p1
zero2 = P2 - p2
sol = solve( [zero1,zero2] , [x1;x2])

 = T + V |> subs(sol)

Qdamping1 = -diff(D;x1) |> subs(sol)
Qdamping2 = -diff(D;x2) |> subs(sol)


reversesol = solve( [zero1,zero2] , [p1,p2])

p1 = -diff( ,x1)+Qdamping1 + Q[1] |> subs(reversesol)
p2 = -diff( ,x2)+Qdamping2 + Q[2] |> subs(reversesol)
#˙p1 = m1*x1 and˙p2 = m2*x2

# z = x - x0
# u = f - f0
# z = [x1; x2;˙x1;˙x2]
# u = [0;f]
rule = Dict(x1=>z1,x2=>z2;x1=>z3;x2=>z4,f=>u)
```

```
z = [x1;x2;p1/m1;p2/m2]
z = z.subs(reversesol)
z = z.subs(rule)

#plug in values
values = Dict(m1=>1,m2=>1,k1=>20,k2=>10,c1=>.4,c2=>.2)

#˙z = A*z + B*u
# y = C*z + D*u
A = fill(exp(x2^x1),(4,4))
let A = A
    for i = 1:4
        A[i,1] = diff(z[i],z1)
        A[i,2] = diff(z[i],z2)
        A[i,3] = diff(z[i],z3)
        A[i,4] = diff(z[i],z4)
    end
    return A
end
A = A.subs(values)
afloat = fill(NaN,(size(A,1),size(A,2)))
A = oftype(afloat,A)


B = fill(exp(x2^x1),(4,1))
let B = B
    for i = 1:4
        #B[i,1] = 0
        B[i,1] = diff(z[i],u)
    end
    return B
end
B = B.subs(values)
bfloat = fill(NaN,(size(B,1),size(B,2)))
B = oftype(bfloat,B)
C = [0 1 0 0.]
D = [0]

bfloat = fill(NaN,(size(B,1),size(B,2)))
B = oftype(bfloat,B)
afloat = fill(NaN,(size(A,1),size(A,2)))
```

```
A = oftype(afloat,A)

Cm = hcat(B,A*B,A^2*B,A^3*B)
Om = vcat(C,C*A,C*A^2,C*A^3)

import LinearAlgebra
check1 = LinearAlgebra.det(Cm) #not zero so full rank
(blah1,check2,blah2) = LinearAlgebra.svd(Cm)
    check2 # 4 non zero singular values, so rank=4
check3 = LinearAlgebra.eigvals(Cm)
check4 = LinearAlgebra.rank(Cm)

check5 = LinearAlgebra.det(Om) #not zero so full rank
(blah3,check6,blah4) = LinearAlgebra.svd(Om)
    check2 # 4 non zero singular values, so rank=4
check7 = LinearAlgebra.eigvals(Om)
check8 = LinearAlgebra.rank(Om)



#Make A Controller
OS = 5/100
Ts = .25
zeta = sqrt(log(OS)^2/(pi^2+log(OS)^2))
omega = 4/Ts/zeta
Ts2 = 5*Ts
omega2 = 4/Ts2/zeta

# p1 = Polynomials.poly([-1,-2,-3,-4])
s1 = -zeta*omega+1im*omega*sqrt(1 - zeta^2)
s2 = -zeta*omega2+1im*omega2*sqrt(1 - zeta^2)

import Polynomials
p1 = Polynomials.poly([s1, conj(s1), s2, conj(s2)  ])Λ
 = zeros(4,4)
for i = 0:4
        global Λ += p1.a[i+1]*A^i
endΛ
 = real(Λ)

e_c = hcat(B, A*B, A^2*B, A^3*B)
```

```
K = (e_c\Λ)[end,:]
LinearAlgebra.eigvals(A - B*K') # just to check that they went where we wanted them


function ode2MassSprings(dz,z,p,t)
    #p = [M1,M2,K1,K2,C1,C2]

    x0 = [0 0 0 0.]
    r(t) = 0
    x = z - x0
    U = - LinearAlgebra.dot(K,x)

    dz[1] = z[3]
    dz[2] = z[4]
    dz[3] = (-p[5]*z[3] - p[6]*(2*z[3] - 2*z[4])/2 -p[3]*z[1] - p[4]*(2*z[1] - 2*z[2])/2)/p[1]
    dz[4] = (-p[6]*(-2*z[3] + 2*z[4])/2 - p[4]*(-2*z[1] + 2*z[2])/2 + U)/p[2]
end

import DifferentialEquations
tspan = (0.0,10)
z0 = [2 1 0 0.]
p = [1,1,20,10,.4,.2]
prob = DifferentialEquations.ODEProblem(ode2MassSprings,z0,tspan,p)
sol = DifferentialEquations.solve(prob)
plot(sol, vars = (0,1))
plot(sol, vars = (0,2))
```

```
using SymPy
using Plots

@vars t f m1 m2 k1 k2 c1 c2 z1 z2 z3 z4 u
x1 = SymFunction("x1")(t)
x2 = SymFunction("x2")(t)
ẋ1 = SymFunction("ẋ1")(t)
ẍ1 = SymFunction("ẍ1")(t)
ẋ2 = SymFunction("ẋ2")(t)
ẍ2 = SymFunction("ẍ2")(t)
* Dr. Fitzgerald, The ẋ are x dots and x double dots. I considered changing all of
them for this doc, but I thought that it may introduce errors.
```

```
diff(x1,t)
diff(x1,t,t)
diff(diff(x1,t),t)

#defining system
q = [x1;x2]
Q = [0;f] #right hand side of equation

# Determine kinetic energy, potential energy, and dissipation function
T = 1//2*m1*(diff(x1,t))^2 + 1//2*m2*(diff(x2,t))^2 |> subs(diff(x1,t),ẋ1) |>
subs(diff(x2,t),ẋ2)

V = 1//2*k1*x1^2 + 1//2*k2*(x2-x1)^2

D = 1//2*c1*(diff(x1,t))^2 + 1//2*c2*(diff(x2,t)-diff(x1,t))^2 |> subs(diff(x1,t),ẋ1)
|> subs(diff(x2,t),ẋ2)

L = T-V

#Left Hand Side
Q1 = diff(diff(L,ẋ1),t) - diff(L,x1) + diff(D,ẋ1)|> subs(diff(ẋ1,t),ẍ1) |>
 subs(diff(ẋ2,t),ẍ2)
Q2 = diff(diff(L,ẋ2),t) - diff(L,x2) + diff(D,ẋ2) |> subs(diff(ẋ1,t),ẍ1)|>
subs(diff(ẋ2,t),ẍ2)

# state the Lagrangian equations of motion
eqn1 = Q[1] - Q1 #this equals zero
eqn2 = Q[2] - Q2 #this equals zero




## NOW DOING HAMILTONIAN
P1 = diff(L,ẋ1)
P2 = diff(L,ẋ2)
@vars p1 p2
zero1 = P1 - p1
```

```
zero2 = P2 - p2
sol = solve( [zero1,zero2] , [ẋ1,ẋ2])

  = T + V |> subs(sol)

Qdamping1 = -diff(D,ẋ1) |> subs(sol)
Qdamping2 = -diff(D,ẋ2) |> subs(sol)


reversesol = solve( [zero1,zero2] , [p1,p2])

ṗ1 = -diff( ,x1)+Qdamping1 + Q[1] |> subs(reversesol)
ṗ2 = -diff( ,x2)+Qdamping2 + Q[2] |> subs(reversesol)
# ṗ1 = m1*ẍ1 and ṗ2 = m2*ẍ2

# z = x - x0
# u = f - f0
# z = [x1; x2; ẋ1; ẋ2]
# u = [0;f]
rule = Dict(x1=>z1,x2=>z2,ẋ1=>z3,ẋ2=>z4,f=>u)
ż = [ẋ1;ẋ2;ṗ1/m1;ṗ2/m2]
ż = ż.subs(reversesol)
ż = ż.subs(rule)

#plug in values
values = Dict(m1=>1,m2=>1,k1=>20,k2=>10,c1=>.4,c2=>.2)

# ż = A*z + B*u
# y = C*z + D*u
A = fill(exp(ẍ2^ẍ1),(4,4))
let A = A
    for i = 1:4
        A[i,1] = diff(ż[i],z1)
        A[i,2] = diff(ż[i],z2)
        A[i,3] = diff(ż[i],z3)
        A[i,4] = diff(ż[i],z4)
    end
    return A
```

```
end
A = A.subs(values)
afloat = fill(NaN,(size(A,1),size(A,2)))
A = oftype(afloat,A)


B = fill(exp(ẍ2^ẍ1),(4,1))
let B = B
    for i = 1:4
        #B[i,1] = 0
        B[i,1] = diff(ż[i],u)
    end
    return B
end
B = B.subs(values)
bfloat = fill(NaN,(size(B,1),size(B,2)))
B = oftype(bfloat,B)
C = [0 1 0 0.]
D = [0]

bfloat = fill(NaN,(size(B,1),size(B,2)))
B = oftype(bfloat,B)
afloat = fill(NaN,(size(A,1),size(A,2)))
A = oftype(afloat,A)

Cm = hcat(B,A*B,A^2*B,A^3*B)
Om = vcat(C,C*A,C*A^2,C*A^3)

import LinearAlgebra
check1 = LinearAlgebra.det(Cm) #not zero so full rank
(blah1,check2,blah2) = LinearAlgebra.svd(Cm)
    check2 # 4 non zero singular values, so rank=4
check3 = LinearAlgebra.eigvals(Cm)
check4 = LinearAlgebra.rank(Cm)

check5 = LinearAlgebra.det(Om) #not zero so full rank
(blah3,check6,blah4) = LinearAlgebra.svd(Om)
    check2 # 4 non zero singular values, so rank=4
```

```
check7 = LinearAlgebra.eigvals(Om)
check8 = LinearAlgebra.rank(Om)



#Make A Controller
OS = 5/100
Ts = .25
zeta = sqrt(log(OS)^2/(pi^2+log(OS)^2))
omega = 4/Ts/zeta
Ts2 = 5*Ts
omega2 = 4/Ts2/zeta

# p1 = Polynomials.poly([-1,-2,-3,-4])
s1 = -zeta*omega+1im*omega*sqrt(1 - zeta^2)
s2 = -zeta*omega2+1im*omega2*sqrt(1 - zeta^2)

import Polynomials
p1 = Polynomials.poly([s1, conj(s1), s2, conj(s2)  ])
Λ = zeros(4,4)
for i = 0:4
        global Λ += p1.a[i+1]*A^i
end
Λ = real(Λ)

e_c = hcat(B, A*B, A^2*B, A^3*B)
K = (e_c\Λ)[end,:]
LinearAlgebra.eigvals(A - B*K') # just to check that they went where we wanted them


function ode2MassSprings(dz,z,p,t)
    #p = [M1,M2,K1,K2,C1,C2]

    x0 = [0 0 0 0.]
    r(t) = 0
    x = z - x0
    U = - LinearAlgebra.dot(K,x)
```

```
    dz[1] = z[3]
    dz[2] = z[4]
    dz[3] = (-p[5]*z[3] - p[6]*(2*z[3] - 2*z[4])/2 -p[3]*z[1] - p[4]*(2*z[1] -
    2*z[2])/2)/p[1]
    dz[4] = (-p[6]*(-2*z[3] + 2*z[4])/2 - p[4]*(-2*z[1] + 2*z[2])/2 + U)/p[2]
end

import DifferentialEquations
tspan = (0.0,10)
z0 = [2 1 0 0.]
p = [1,1,20,10,.4,.2]
prob = DifferentialEquations.ODEProblem(ode2MassSprings,z0,tspan,p)
sol = DifferentialEquations.solve(prob)
plot(sol, vars = (0,1))
plot(sol, vars = (0,2))
```

1 Answers

1.1

1) $T = \frac{m1*(\dot{x1})^2}{2} + \frac{m2*(\dot{x2})^2}{2}$

2)

1.2

$\dot{z} =$