ECE-597: Optimal Control
Homework #4

Due October 4, 2007

1. In this problem, we will first derive a somewhat complicated analytical solution, and then simulate the system.

Consider the problem

$$\begin{aligned}\text{minimize } J &= \frac{1}{2}(x(N)-d)^2 + \frac{1}{2}\sum_{i=0}^{N-1} wu(i)^2 \\ \text{subject to } &= x(i+1) = ax(i) + bu(i)\end{aligned}$$

where $x(0) = x_0$, N, and the desired final point, d are given.

a) Show that the discrete Euler Lagrange equations are

$$\begin{aligned}\lambda(i) &= a\lambda(i+1) \\ \lambda(N) &= x(N) - d \\ H_{u(i)} &= wu(i) + \lambda(i+1)b = 0\end{aligned}$$

b) Show that we can use these equations to write

$$x(i+1) = ax(i) - \gamma\lambda(i+1)$$

where $\gamma = b^2/w$.

c) Assuming the form $\lambda(i) = cr^i$, show that we get the equation

$$\lambda(i) = \lambda(N)a^{N-i}$$

and that, combining with the answer to (b) we get

$$x(i+1) = ax(i) - \gamma a^{N-i-1}\lambda(N)$$

By taking z transforms, the above equation becomes

$$X(z) = \frac{z}{z-a}x_0 - \gamma a^{N-1}\lambda(N)\left[\frac{z}{(z-a)(z-\frac{1}{a})}\right]$$

Taking the inverse z-transforms, we get

$$x(i) = a^i x_0 - \gamma\lambda(N)a^{N-i}\frac{(1-a^{2i})}{1-a^2}$$

d) At the final time N, we have

$$x(N) \ = \ a^N x_0 - \lambda(N)\Lambda$$

where

$$\Lambda \ = \ \gamma \frac{1-a^{2N}}{1-a^2}$$

Using the boundary condition for λ, show that we get

$$\lambda(N) \ = \ \frac{1}{1+\Lambda}(a^N x_0 - d)$$

and hence

$$\lambda(i) \ = \ \frac{1}{1+\Lambda}(a^N x_0 - d)a^{N-i}$$

e) Finally, show that we get the optimal input

$$u(i) \ = \ \frac{b}{w(1+\Lambda)}(d - a^N x_0)a^{N-i-1}$$

f) We will examine the specific problem with $a = -0.8$ and $b = 0.2$ in all that follows. Write a Matlab script file (or two files) to use the **dop0.m** routine to find the optimal u's. Your subroutine 'name' that is passed to **dop0** will not use the variables dt or t. Run the simulation using tol = 5e-8, mxit = 5000, k = 0.1, w = 0.5, d = 0.3, $x_0 = 1$ for one step. Show that the routine agrees with the analytical value for the optimal u. (I get u = 0.4074)

g) Now run the **dop0** routine and estimate the first 8 values of u, and compare to your analytical values.

h) Now run **dop0** for $N = 25$ (twenty five control signals) with $w = 0.05, 0.5$ and 5. Plot the values of the states versus time, as well as the control signal. Turn in your plots. Be sure to turn in your code and the Matlab output.

2. Consider the problem

$$\begin{aligned} \text{minimize } J \ &= \ \frac{1}{2}(x(N) - d)^2 + \frac{1}{2}\sum_{k=0}^{k=N-1} rx(k)^2 + wu(k)^2 \\ \text{subject to} \quad & x(i+1) = ax(i) + bu(i) \end{aligned}$$

where we will use the same values of a, b, and d as we used above, with the same initial conditions.

a) Write the code you need to use the **dop0** routine to find the optimal values of u.

b) Plot the control signal and the state for $N = 25$ for $w = 0.001, r = 0.001$, $w = 1, r = 10$, and $w = 0.1, r = 0.001$ Turn in your code and your plots.

3. Consider the problem

$$\begin{aligned} \text{minimize } J &= \frac{1}{2}(x(t_f)^2 + y(t_f)^2) \\ \text{subject to} \quad & \dot{x}(t) = t + u(t) \\ & \dot{y}(t) = x(t) \end{aligned}$$

where $x(0) = 1$, $y(0) = 1$, $t_f = 0.5$. We want to transform this system into a discrete time system and use $N = 50$ time steps.

a) Show that an equivalent discrete-time system is

$$\begin{aligned} \text{minimize } J &= \frac{1}{2}(x(N)^2 + y(N)^2) \\ \text{subject to} \quad & x(k+1) = x(k) + \frac{1}{2}\Delta T^2 + \Delta T u(k) \\ & y(k+1) = y(k) + \Delta T x(k) + \frac{1}{6}\Delta T^3 + \frac{1}{2}\Delta T^2 u(k) \end{aligned}$$

b) Simulate the system using **dop0**. You may need to make the input variable k equal to about 50 or so to get convergence. Using the optimal control values, determine the values of x and y from 0 to 0.5, and plot the values of the states on one graph. (It should be clear that the final values should be very near to zero.) Also plot the value of the control signal on another graph. (Use the subplot command to get them both on one page.) Turn in your code and your graphs.

4. In this problem we will determine the equations necessary to solve the following discrete-time tracking problem:

Determine the control sequence $u(k)$ to minimize

$$J = \frac{1}{2}[y(N) - r(N)]^T Q_f [y(N) - r(N)] + \sum_{k=0}^{N-1} \left\{ \frac{1}{2}[y(k) - r(k)]^T Q [y(k) - r(k)] + \frac{1}{2} u(k)^T R u(k) \right\}$$

subject to

$$\begin{aligned} x(k+1) &= Gx(k) + Hu(k) \\ y(k) &= Cx(k) \\ x(0) &= x_0 \end{aligned}$$

where $R = R^T$, $Q = Q^T$, $Q_f = Q_f^T$. $r(k)$ is the signal we want our system to track. We assume R^{-1} exists in what follows.

a) Form the Hamiltonian $H(k)$ and the terminal condition $\phi(k)$. Show that the Euler-Lagrange equations lead to the following equations:

$$\begin{aligned}
\lambda(k) &= C^TQCx(k) - C^TQr(k) + G^T\lambda(k+1) \\
u(k) &= -R^{-1}H^T\lambda(k+1) \\
\lambda(N) &= C^TQ_fCx(N) - C^TQ_fr(N)
\end{aligned}$$

b) Starting from the state equation $x(k+1) = Gx(k) + Hu(k)$, insert your expression for $u(k)$ derived in part **a**, and then assume $\lambda(k) = S(k)x(k) - v(k)$. Show that we can write

$$\begin{aligned}
x(k+1) &= \Delta(k)Gx(k) + \Delta(k)HR^{-1}H^Tv(k+1) \\
\Delta(k) &= \left[I + HR^{-1}H^TS(k+1)\right]^{-1}
\end{aligned}$$

c) We also have

$$\lambda(k) = S(k)x(k) - v(k) \quad = \quad C^TQCx(k) - C^TQr(k) + G^T\lambda(k+1)$$

Use your assumed form for $\lambda(k)$ from part **b** and then your expression for $x(k+1)$ from part **b** to eliminate all $x(k+1)$ and all $\lambda(k+1)$ terms.

d) Show that you can rearrange the expression from part **c** into the form

$$\begin{aligned}
&\left\{v(k) + G^TS(k+1)\Delta(k)HR^{-1}H^Tv(k+1) - G^Tv(k+1) - C^TQr(k)\right\} \\
&\qquad + \left\{-S(k) + G^TS(k+1)\Delta(k)G + C^TQC\right\}x(k) = 0
\end{aligned}$$

This expression must be true for all $x(k)$, so we have

$$\begin{aligned}
S(k) &= G^TS(k+1)\Delta(k)G + C^TQC \\
v(k) &= \left[-G^TS(k+1)\Delta(k)HR^{-1}H^T + G^T\right]v(k+1) + C^TQr(k) \\
\Delta(k) &= \left[I + HR^{-1}H^TS(k+1)\right]^{-1}
\end{aligned}$$

e) Show that $S(N) = C^TQ_fC$ and $v(N) = C^TQ_fr(N)$

f) Finally, we need a control law. Starting with the expression $Ru(k) = -H^T\lambda(k+1)$, show that we have

$$\begin{aligned}
u(k) &= -K(k)x(k) + u_f(k) \\
K(k) &= \left[R + H^TS(k+1)H\right]^{-1}H^TS(k+1)G \\
u_f(k) &= \left[R + H^TS(k+1)H\right]^{-1}H^Tv(k+1)
\end{aligned}$$

where $K(k)$ is a time-varying feedback gain, and $u_f(k)$ is a feedforward control signal.

In the following parts of the problem we are going to simulate both a one and two degree of freedom torsional ECP system from the controls lab using Matlab and Simulink. The continuous time model has been sampled for you, don't worry about how this is done, or why there seems to be an extra state. At this point you mostly have to run the routines. You will need the routines **DT_sv1_optimal_tracker.mdl**, **tracker_driver_sv1.m**, **optimal_tracker.m**, **bobs_1dof_model205.mat**, **DT_sv2_optimal_tracker.mdl**, **tracker_driver_sv2.** and **bobs_2dof_model205.mat**.

f) For the one degree of freedom system, if you run the system with $Q = 0.01$, $R = 0.1$, and $Q_f = 0.05$ for a 10 degree step input, you should get plots like those shown in Figure 1. You will also get plots of the time-varying gains K.

g) Modify Q, R, and Q_f so $|r(N) - y(N)| < 0.5$ and the control effort does not saturate. Turn in your plot with the values of the weights you used.

h) Modify Q,R, and Q_f so $|r(N) - y(N)| < 0.5$ and $|r(k) - y(k)| < 0.4$ over 75% of the time and the control effort does not saturate. Turn in your plot with the values of the weights you used.

i) Modify the programs to use constant feedback gains (uncomment lines 88 and 89) and repeat parts **g** and **h**. How do the results compare? Turn in your plots.

j) The LQR algorithm in Matlab is used to determine the continuous time Linear Quadratic Regulator gains in continuous time. In discrete-time we use the dlqr command. Using your weights from parts **g** and **h**, type into Matlab

```
dlqr(G,H,diag([Q 0 0]),R)
```

and compare the gains returned with the gains from our simulation in steady state. They should be very similar!

k) Change the signal we want to track to $r(k) = 15^o \sin(10\pi k \Delta T)$ and modify Q, R, and Q_f so the system tracks the reference signal $r(k)$ reasonably well with constant feedback gains and the control effort does not saturate. You should not that the control signal *leads* the signal we want to track. Turn in your plots and the values of the weights you used.

l) For the two degree of freedom system, controlling the position of the **second disk**, if you run the system with $Q = 0.01$, $R = 0.1$, and $Q_f = 0.05$ for a 10 degree step input, you should get plots like those shown in Figure 2. You will also get plots of the time-varying gains K.

m) Modify Q, R, and Q_f so $|r(N) - y(N)| < 0.5$ and the control effort does not saturate. Turn in your plot with the values of the weights you used.

n) Modify Q,R, and Q_f so $|r(N) - y(N)| < 0.5$ and $|r(k) - y(k)| < 0.4$ over 75% of the time and the control effort does not saturate. Turn in your plot with the values of the weights you used.

o) Modify the programs to use constant feedback gains (uncomment lines 91 and 92) and repeat parts **m** and **n**. How do the results compare? Turn in your plots.

p) Change the signal we want to track to $r(k) = 15^o \sin(10\pi k \Delta T)$ and modify Q, R, and Q_f so the system tracks the reference signal $r(k)$ reasonably well with constant feedback gains and the control effort does not saturate. You should not that the control signal *leads* the signal we want to track. Turn in your plots and the values of the weights you used.

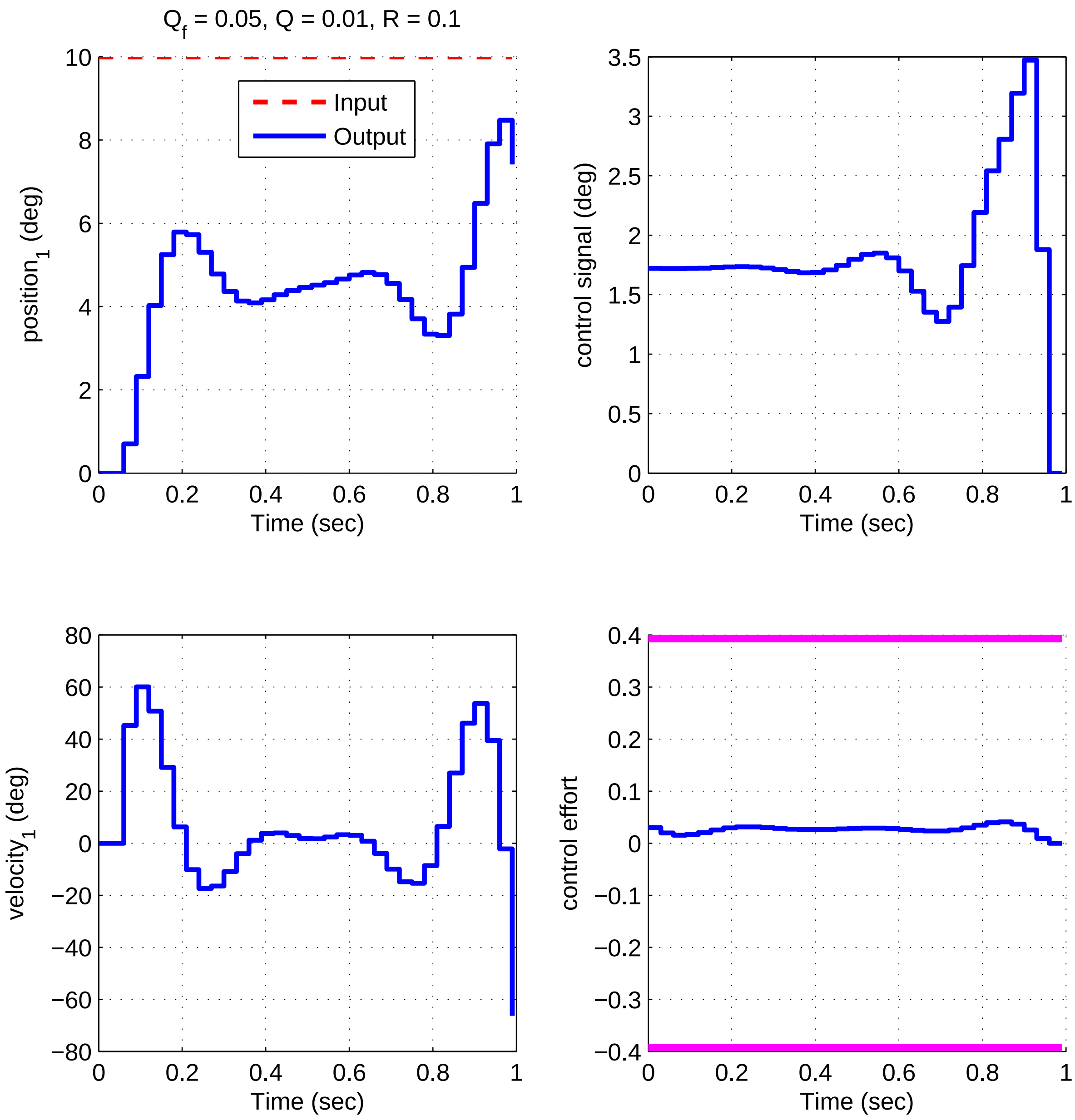

Figure 1: Initial results from the one degree of freedom torsional system using $Q = 0.01$, $R = 0.1$ and $Q_f = 0.05$

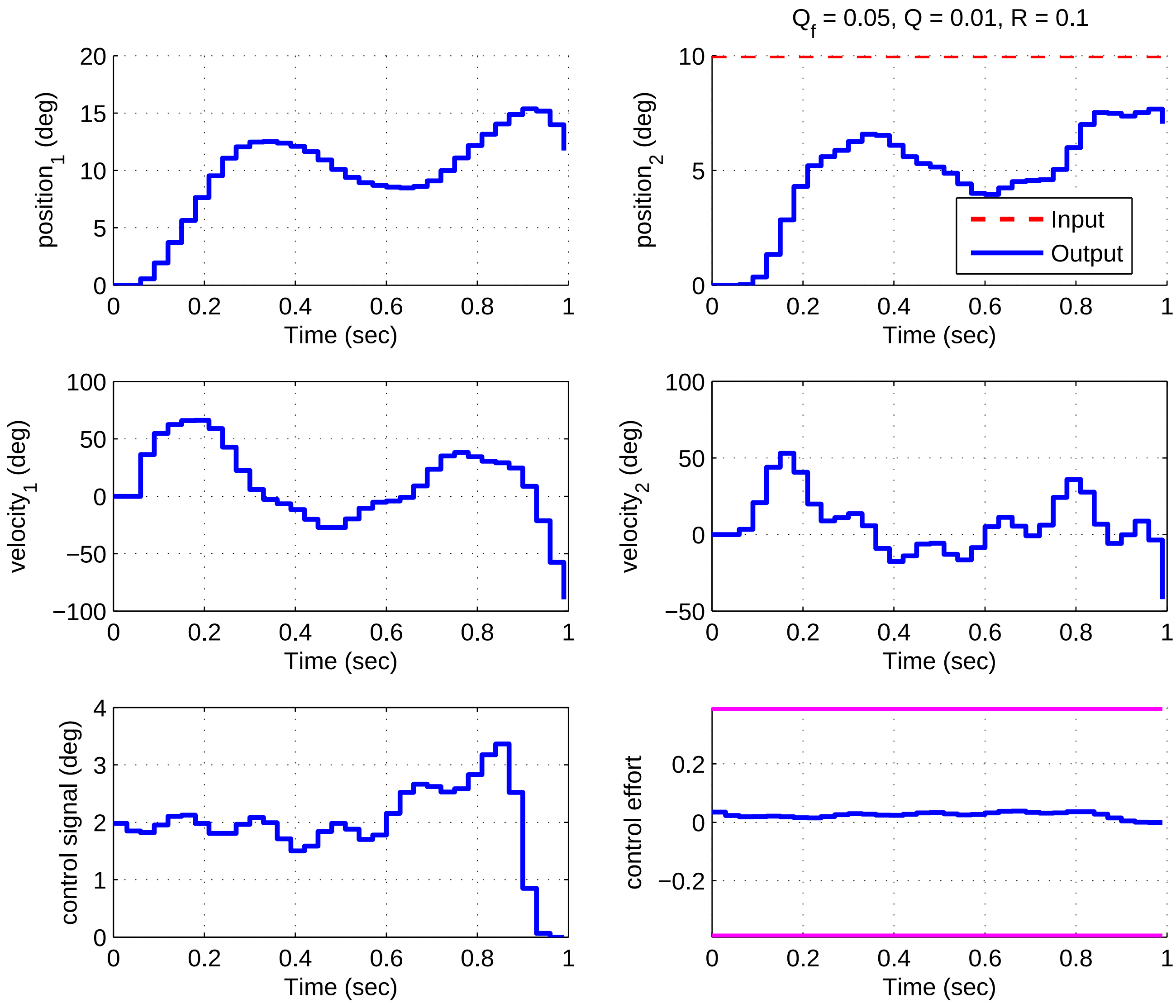

Figure 2: Initial results from the two degree of freedom torsional system controlling the position of the second disk using $Q = 0.01$, $R = 0.1$ and $Q_f = 0.05$

Appendix

dop0.m routine

Only very simple problems can be solved analytically. Usually we need to use a numerical procedure. We will do this using the *Mayer formulation* of the problem. We have been using the *Bolza formulation.* Both of these methods are *equivalent*, and are just different methods of looking at the same thing.

In the *Mayer* formulation, the state vector is augmented by one state $q(i)$ that is the cumulative sum of L up to step i:

$$q(i+1) = q(i) + L[x(i), u(i), i]$$

The *Bolza* performance index

$$J = \phi[x(N)] + \sum_{i=1}^{N-1} L\left[x(i), u(i), i\right]$$

becomes the *Mayer* performance index

$$J = \phi[x(N)] + q(N) = \bar{\phi}[\bar{x}(N)]$$

where

$$\bar{x} = \begin{bmatrix} x \\ q \end{bmatrix}$$

We usually drop the bar on x and ϕ and the problem is stated as finding a sequence $u(i),\ i = 0, ..., N-1$ to minimize (or maximize)

$$J = \phi[x(N)]$$

subject to

$$x(i+1) = f[x(i), u(i), i]$$

and $x(0)$ and N are specified.

We will be utilizing the routine **dop0.m** in this assignment. This routine solves discrete-time optimization problems of the form: find the input sequence $u(i),\ i = 0, .., N-1$ to minimize

$$J = \phi[x(N)] + \sum_{i=0}^{N-1} L[x(i), u(i), i]$$

subject to the constraints

$$\begin{aligned} x(i+1) &= f[x(i), u(i), i] \\ x(0) &= x_0\ (\textit{known}) \end{aligned}$$

Note that we cannot put any *hard* terminal constraints on this problem. That is, we cannot force $x(N)$ to be anything in particular.

In order to use the routine **dop0.m**, you need to write a routine that returns one of three things depending on the value of the variable **flg**. The general form of your routine will be as follows:

```
function [f1,f2] = bobs_dop0(u,s,dt,t,flg)
```

Here u is the current input, $u(i)$, and s contains the current state (including the augmented state), $s(i)$, so $s(i+1) = f(s(i), u(i), \Delta T)$. dt is the time increment ΔT, and t is the current time. You can compute the current index i using the relationship $i = \frac{t}{\Delta T} + 1$. Your routine should compute the following:

$$\begin{array}{ll} \text{if } \mathbf{flg} = 1 & \text{f1} = s(i+1) = f(s(i), u(i), \Delta T) \\ \text{if } \mathbf{flg} = 2 & \text{f1} = \bar{\phi}[\bar{x}(N)], \text{f2} = \bar{\phi}_{s(N)}[\bar{x}(N)] \\ \text{if } \mathbf{flg} = 3 & \text{f1} = f_s, \text{f2} = f_u \end{array}$$

An example of the usage is:

```
[u,s,la0] = dop0('bobs_dop0',u,s0,tf,k,tol,mxit)
```

The (input) arguments to **dop0.m** are the following:

- the function you just created (in single quotes).
- the initial guess for the value of u that minimizes J. This initial size is used to determine the time step: if there are n components in u, then $\Delta T = \frac{t_f}{n}$
- an initial guess of the states, $s0$. Note that you must include and initial guess for the "cumulative" state q also.
- the final time, tf.
- k, the step size parameter, $k > 0$ to minimize. Often you need to play around with this one.
- *tol*, the tolerance (a stopping parameter), when $|\Delta u| < tol$ between iterations, the programs stops.
- *mxit*, the maximum number of iterations to try.

dop0.m returns the following:

- u the optimal input sequence
- s the corresponding states

- $la0$ the Lagrange multipliers

It is usually best to start with a small number of iterations, like 5, and see what happens as you change k. Start with small values of k and gradually increase them. It can be very difficult to make this program converge, especially if your initial guess is far away from the true solution.

Note!! If you are using the **dop0.m** file, and you use the maximum number of allowed iterations, assume that the solution has NOT converged. You must usually change the value of k and/or increase the number of allowed iterations. Do not set tol to less than about 5e-5. Also try to make k as large as possible and still have convergence.

Example A (*Example 2.1.1 in Bryson.*). Assume we have the discretized equations

$$\begin{aligned} v(i+1) &= v(i) + \Delta T \sin(\gamma(i)) \\ x(i+1) &= x(i) + \Delta l(i) \cos(\gamma(i)) \\ \Delta l(i) &= \Delta T v(i) + \frac{1}{2}\Delta T^2 \sin(\gamma(i)) \end{aligned}$$

We can rewrite the second equation as

$$x(i+1) = x(i) + \Delta T v(i) \cos(\gamma(i)) + \frac{1}{4}\Delta T^2 \sin(2\gamma(i))$$

For this problem we want to **maximize** the final position, so we have

$$\phi[x(N)] = x(N), \quad L = 0$$

Hence $q(i) = 0$ for all i and we do not need to augment the state.

Hence we can write

$$f = \begin{bmatrix} f_1 \\ f_2 \end{bmatrix} = \begin{bmatrix} v(i) + \Delta T \sin(\gamma(i)) \\ x(i) + \Delta T v(i) \cos(\gamma(i)) + \frac{1}{4}\Delta T^2 \sin(2\gamma(i)) \end{bmatrix}$$

Let's define the state variables as

$$s = \begin{bmatrix} v \\ x \end{bmatrix}$$

Then

$$\phi_s = \begin{bmatrix} \phi_v & \phi_x \end{bmatrix} = \begin{bmatrix} 0 & 1 \end{bmatrix}$$

and

$$f_s = \begin{bmatrix} \frac{\partial f}{\partial v(i)} & \frac{\partial f}{\partial x(i)} \end{bmatrix} = \begin{bmatrix} \frac{\partial f_1}{\partial v(i)} & \frac{\partial f_1}{\partial x(i)} \\ \frac{\partial f_2}{\partial v(i)} & \frac{\partial f_2}{\partial x(i)} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ \Delta T \cos(\gamma(i)) & 1 \end{bmatrix}$$

and

$$f_u = \begin{bmatrix} \frac{\partial f_1}{\partial \gamma(i)} \\ \frac{\partial f_2}{\partial \gamma(i)} \end{bmatrix} = \begin{bmatrix} \Delta T \cos(\gamma(i)) \\ -\Delta T v(i) \sin(\gamma(i)) + \frac{1}{2}\Delta T^2 \cos(2\gamma(i)) \end{bmatrix}$$

For this problem we will assume $T_f = 1$. This is implemented in the routine **bobs_dop0_a.m** on the class web site, and it is run using the driver file **dop0_example_a.m**.

Example B We would like to solve the following discrete-time problem:

The Bolza formulation:

We want to find the input sequence $u(0), u(1), ..., u(N-1)$ to minimize

$$J = \phi[x(N)] + \sum_{i=0}^{N-1} L[x(i), u(i), i] = x(N)^T x(N) + \sum_{i=0}^{N-1} u^2(i)$$

subject to the constraint

$$x(i+1) = gx(i) + hu(i) \text{ for } i = 0..N-1$$

We will assume $N = 5$, $T_f = 2$, $x(0) = [1\ 2]^T$, $a = \begin{bmatrix} 1 & 0 \\ 1 & -2 \end{bmatrix}$, $b = \begin{bmatrix} -5 \\ 0 \end{bmatrix}$, and $g = I + a\Delta T$, $h = b\Delta T$.

We now need to convert this to the *Mayer formulation*. We first identify $L(i) = u(i)^2$ and $\phi[x(N)] = x(N)^T x(N)$.

Next we define $q(0) = 0$ and $q(i+1) = q(i) + L(i)$. So we have

$$\begin{aligned} q(1) &= L(0) = u(0)^2 \\ q(2) &= q(1) + L(1) = q(1) + u(1)^2 \\ q(3) &= q(2) + L(2) = q(2) + u(2)^2 \\ q(4) &= q(3) + L(3) = q(3) + u(3)^2 \\ q(5) &= q(4) + L(4) = q(4) + u(4)^2 \end{aligned}$$

Then we have

$$J = \phi[x(5)] + q(5) = \bar{\phi}[\bar{x}(5)] = x(5)^T x(5) + q(5)$$

and

$$s = \begin{bmatrix} x \\ q \end{bmatrix}$$

We then need to compute the following:

$$\begin{aligned}
f(s) &= \begin{bmatrix} gx+hu \\ q+u^2 \end{bmatrix} \\
\bar{\phi}[\bar{x}(N)] &= x^T x + q \\
\bar{\phi}_{s(N)}[\bar{x}(N)] &= \begin{bmatrix} 2x^T & 1 \end{bmatrix} \\
f_s &= \begin{bmatrix} g & 0 \\ 0 & 1 \end{bmatrix} \\
f_u &= \begin{bmatrix} h \\ 2u \end{bmatrix}
\end{aligned}$$

This is implemented in the routine **bobs_dop0_b.m** on the class web site, and it is run using the driver file **dop0_example_b.m**. Note that it is very difficult to get **dop0.m** to converge for this case unless you have a good initial guess for the control input!

<u>**Example C**</u> (*Problem 2.2.7 in Bryson.*) In this problem we want to find the input sequence $\theta(i),\ i=0,...,N-1$ to minimize the function

$$\frac{D}{\pi q} = C_p\left(\frac{\pi}{2}\right)\left[r(N)^2\right] + \sum_{i=0}^{N-1} C_p(\theta(i))\left\{r^2(i) - r^2(i+1)\right\}$$

where

$$\begin{aligned}
C_p(\sigma) &= 2\sin^2(\sigma) \\
r(i+1) &= r(i) - \frac{l}{N}\tan(\theta(i)) = r(i) - \Delta T\tan(\theta(i)) \\
\Delta T &= \frac{l}{N} \\
r(0) &= a \text{ (known)}
\end{aligned}$$

We first have to convert this to the Mayer formulation. Let's let

$$s = \begin{bmatrix} r \\ q \end{bmatrix}$$

Then we have

$$\begin{aligned}
q(0) &= 0 \\
q(i+1) &= q(i) + C_p(\theta(i))\left\{r^2(i) - [r(i) - \Delta T\tan(\theta(i))]^2\right\} \\
&= q(i) + 2\sin^2(\theta(i)) + \left\{r^2(i) - \left[r^2(i) - 2\Delta T r(i)\tan(\theta(i)) + \Delta T^2\tan^2(\theta(i))\right]\right\} \\
&= q(i) + 2\sin^2(\theta(i)) + \left\{2\Delta T r(i)\tan(\theta(i)) - \Delta T^2\tan^2(\theta(i))\right\} \\
&= q(i) + 2\sin^2(\theta(i))\Delta T\tan(\theta(i))\left\{2r(i) - \Delta T\tan(\theta(i))\right\}
\end{aligned}$$

Now

$$J = \bar{\phi}[\bar{x}(N)] = 2r^2(N) + q(N)$$

We then need to compute the following:

$$\begin{aligned}
f(s) &= \begin{bmatrix} r - \Delta T \tan(u) \\ q + 2\Delta T \sin^2(u)\tan(u)(2r - \Delta T \tan(u)) \end{bmatrix} \\
\bar{\phi}[\bar{x}(N)] &= 2r^2 + q \\
\bar{\phi}_{s(N)}[\bar{x}(N)] &= \begin{bmatrix} 4r & 1 \end{bmatrix} \\
f_s &= \begin{bmatrix} 1 & 0 \\ 4\Delta T \sin^2(u)\tan(u) & 1 \end{bmatrix} \\
f_u &= \begin{bmatrix} -\Delta T \sec^2(u) \\ \Delta T 4r \left\{ 3\sin(u)^2 + \sin^2(u)\tan^2(u) \right\} - 4\Delta T^2 \left\{ 2\sin^2(u)\tan(u) + \tan^3(u)\sin^2(u) \right\} \end{bmatrix}
\end{aligned}$$

This is implemented in the routine **bobs_dop0_c.m** on the class web site, and it is run using the driver file **dop0_example_c.m**.