ECE-597: Optimal Control Homework #5

Due: Wednesday April 26, 2006

Continuous-Time Problems with Hard Terminal Constraints using **fopc.m**

We will be utilizing the routine **fopc.m** to solve continuous-time optimization problems of the form:

Find the input $u(t),\ t_0 \le t \le t_f$ to minimize

$$J = \phi[x(t_f)] + \int_{t_0}^{t_f} L[x(t), u(t), t]dt$$

subject to the constraints

$$\begin{aligned} \dot{x}(t) &= f[x(t), u(t), t] \\ \psi[x(t_f)] &= 0 \\ x(0) &= x_0 \ (known) \end{aligned}$$

Note that we are now adding **hard** terminal constraint, i.e., we can force $x(t_f)$ to end at certain values.

In the *Mayer* formulation, the state vector is augmented by one state $q(t)$ that is the integral of L from t_0 to t:

$$q(t) = \int_{t_0}^{t} L[x(t), u(t), t]dt$$

The *Bolza* performance index

$$J = \phi[x(t_f)] + \int_{t_0}^{t_f} L\left[x(t), u(t), t\right] dt$$

becomes the *Mayer* performance index

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

where

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

and $\psi[x(t_f)]$ is another constraint (like $\dot{x}(t) = f(x(t), u(t), t)$). We usually drop the bar on x and ϕ and the problem is stated as finding a sequence $u(t),\ t_0 \le t \le t_f$ to minimize (or maximize)

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

subject to

$$\begin{aligned}\dot{x}(t) &= f[x(t), u(t), t] \\ \psi[x(t_f)] &= 0\end{aligned}$$

and t_0, $x(t_0)$ and t_f are specified.

In order to use the routine **fopc.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_fopc(u,s,t,flg)
```

Here u is the current input, $u(t)$, and s ($s(t)$)contains the current state (including the augmented state), so $\dot{s}(t) = f(s(t), u(t), t)$. Your routine should compute the following:

$$\begin{array}{ll} \text{if } \mathbf{flg} = 1 & \text{f1} = \dot{s}(t) = f(s(t), u(t), t) \\ \text{if } \mathbf{flg} = 2 & \text{f1} = \begin{bmatrix} \phi[s(t_f)] \\ \psi[s(t_f)] \end{bmatrix}, \text{f2} = \begin{bmatrix} \phi_{s(t_f)}[s(t_f)] \\ \psi_{s(t_f)}[s(t_f)] \end{bmatrix} \\ \text{if } \mathbf{flg} = 3 & \text{f1} = f_s, \text{f2} = f_u \end{array}$$

An example of the usage is:

```
[tu,ts,nu,la0] = fopc('bobs_fopc',tu,tf,s0,k,told,tols,mxit,eta)
```

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

- the function you just created (in single quotes).
- *tu* is an initial guess of times (first column), and control values (subsequent columns) that minimizes J. If there are multiple control signals at a given time, they are all in the same row. Note that these are just the initial time and control values, the times and control values will be modified as the program runs. The initial time should start at zero.
- the initial 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.
- *told*, the tolerance (a stopping parameter) for ode23 (differential equation solver for Matlab)
- *tols*, the tolerance (a stopping parameter), when $|\Delta u| < tols$ between iterations, the programs stops.

- *mxit*, the maximum number of iterations to try.
- *eta*, where $0 <$ eta ≤ 1, and d(psi) = -eta*psi is the desired change in the terminal constraints on the next iteration.

fopc.m returns the following:

- *tu* the optimal input sequence and corresponding times. The first column is the time, the corresponding columns are the control signals. All entries in one row correspond to one time.
- *ts* the states and corresponding times. The first column is the time, the corresponding columns are the states. All entries in one row correspond to the same time. Note that the times in *tu* and the times in *ts* may not be the same, and they may not be evenly spaced.
- *nu*, the Lagrange multipliers on psi
- *la*0 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 **fopc.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 Assume the continuous-time equations of motion

$$\begin{aligned}
\dot{u}(t) &= a\cos(\theta(t)) \\
\dot{v}(t) &= a\sin(\theta(t)) \\
\dot{y}(t) &= v(t) \\
\dot{x}(t) &= u(t)
\end{aligned}$$

We want to find $\theta(t)$ to maximize $u(t_f)$ and we want $v(t_f) = 0$, and $y(t_f) = y_f$, where $a = 1$, $y_f = 0.2$, and $t_f = 1.0$.

Let's define the state vector as

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

and let's define

$$\Phi[s(t_f)] \;=\; \left[\begin{array}{c} \phi[s(t_f)] \\ \psi[s(t_f)] \end{array}\right]$$

For this problem we want to **maximize** $u(t_f)$, so we have

$$\phi[s(t_f)] \;=\; u(t_f), \quad L = 0$$

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

We also have the hard terminal constraints

$$\psi[s(t_f)] \;=\; \left[\begin{array}{c} v(t_f) \\ y(t_f) - y_f \end{array}\right]$$

so we can write

$$\Phi[s(t_f)] \;=\; \left[\begin{array}{c} u(t_f) \\ v(t_f) \\ y(t_f) - y_f \end{array}\right]$$

We can then write

$$f \;=\; \left[\begin{array}{c} f_1 \\ f_2 \\ f_3 \\ f_4 \end{array}\right] = \left[\begin{array}{c} a\cos(\theta(t)) \\ a\sin(\theta(t)) \\ v(t) \\ u(t) \end{array}\right]$$

Next we need

$$\begin{aligned}
\Phi_{s(t_f)}[s(t_f)] \;&=\; \left[\begin{array}{c} \phi_{s(t_f)}[s(t_f)] \\ \psi_{s(t_f)}[s(t_f)] \end{array}\right] \\
&=\; \left[\begin{array}{cccc} \frac{\partial\phi[s(t_f)]}{\partial u(t_f)} & \frac{\partial\phi[s(t_f)]}{\partial v(t_f)} & \frac{\partial\phi[s(t_f)]}{\partial y(t_f)} & \frac{\partial\phi[s(t_f)]}{\partial x(t_f)} \\ \frac{\partial\psi[s(t_f)]}{\partial u(t_f)} & \frac{\partial\psi[s(t_f)]}{\partial v(t_f)} & \frac{\partial\psi[s(t_f)]}{\partial y(t_f)} & \frac{\partial\psi[s(t_f)]}{\partial x(t_f)} \end{array}\right] \\
&=\; \left[\begin{array}{cccc} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{array}\right]
\end{aligned}$$

Then we need

$$f_s \;=\; \left[\begin{array}{cccc} \frac{\partial f}{\partial u(t)} & \frac{\partial f}{\partial v(t)} & \frac{\partial f}{\partial y(t)} & \frac{\partial f}{\partial x(t)} \end{array}\right] = \left[\begin{array}{cccc} \frac{\partial f_1}{\partial u(t)} & \frac{\partial f_1}{\partial v(t)} & \frac{\partial f_1}{\partial y(t)} & \frac{\partial f_1}{\partial x(t)} \\ \frac{\partial f_2}{\partial u(t)} & \frac{\partial f_2}{\partial v(t)} & \frac{\partial f_2}{\partial y(t)} & \frac{\partial f_2}{\partial x(t)} \\ \frac{\partial f_3}{\partial u(t)} & \frac{\partial f_3}{\partial v(t)} & \frac{\partial f_3}{\partial y(t)} & \frac{\partial f_3}{\partial x(t)} \\ \frac{\partial f_4}{\partial u(t)} & \frac{\partial f_4}{\partial v(t)} & \frac{\partial f_4}{\partial y(t)} & \frac{\partial f_4}{\partial x(t)} \end{array}\right]$$

$$= \left[\begin{array}{cccc} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{array}\right]$$

and finally

$$f_u = f_\theta = \begin{bmatrix} \frac{\partial f_1}{\partial \theta(t)} \\ \frac{\partial f_2}{\partial \theta(t)} \\ \frac{\partial f_3}{\partial \theta(t)} \\ \frac{\partial f_4}{\partial \theta(t)} \end{bmatrix} = \begin{bmatrix} -a\sin(\theta(t)) \\ a\cos(\theta(t)) \\ 0 \\ 0 \end{bmatrix}$$

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

Discrete-Time Problems with Hard Terminal Constraints using **dopc.m**

We will be utilizing the routine **dopc.m** to solve 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

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

Note that we are now adding **hard** terminal constraint, i.e., we can force $x(N)$ to end at certain values.

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

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

and $\psi[x(N)]$ is another constraint (like $x(i+1) = f(x(i), u(i), i)$). 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

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

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

In order to use the routine **dopc.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_dopc(u,s,dt,t,flg)
```

Here u is the current input, $u(i)$, and s ($s(i)$)contains the current state (including the augmented state) 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:

$$\begin{array}{ll} \text{if } \textbf{flg} = 1 & \text{f1} = s(i+1) = f(s(i), u(i), \Delta T) \\ \text{if } \textbf{flg} = 2 & \text{f1} = \begin{bmatrix} \phi[s(N)] \\ \psi[s(N)] \end{bmatrix}, \text{f2} = \begin{bmatrix} \phi_{s(N)}[s(N)] \\ \psi_{s(N)}[s(N)] \end{bmatrix} \\ \text{if } \textbf{flg} = 3 & \text{f1} = f_s, \text{f2} = f_u \end{array}$$

An example of the usage is:

```
[u,s,nu,la0] = dopc('bobs_dopc',u,s0,tf,k,tol,mxit,eta)
```

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.
- *eta*, where 0 < eta ≤ 1, and d(psi) = -eta*psi is the desired change in the terminal constraints on the next iteration.

dopc.m returns the following:

- u the optimal input sequence
- s the corresponding states
- nu, the Lagrange multipliers on psi
- $la0$ 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 **dopc.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.

<u>**Example B**</u> Assume the discrete-time equations of motion (these are the discretized equations from **Example A**)

$$\begin{aligned}
u(i+1) &= u(i) + a\Delta T\cos(\theta(i)) \\
v(i+1) &= v(i) + a\Delta T\sin(\theta(i)) \\
y(i+1) &= y(i) + \Delta T v(i) + \frac{a}{2}\Delta T^2\sin(\theta(i)) \\
x(i+1) &= x(i) + \Delta T u(i) + \frac{a}{2}\Delta T^2\cos(\theta(i))
\end{aligned}$$

We want to find $\theta(i)$ to maximize $u(N)$ and we want $v(N) = 0$ and $y(N) = y_f$, where $a = 1$, $y_f = 0.2$, $t_f = 1.0$, and $\Delta T = t_f/N$.

Let's define the state vector as

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

and let's define

$$\Phi[s(N)] \;=\; \begin{bmatrix} \phi[s(N)] \\ \psi[s(N)] \end{bmatrix}$$

For this problem we want to **<u>maximize</u>** $u(N)$, so we have

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

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

We also have the hard terminal constraints

$$\psi[s(N)] \;=\; \begin{bmatrix} v(N) \\ y(N) - y_f \end{bmatrix}$$

so we can write

$$\Phi[s(N)] \;=\; \begin{bmatrix} u(N) \\ v(N) \\ y(N) - y_f \end{bmatrix}$$

We can then write

$$f \;=\; \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \end{bmatrix} = \begin{bmatrix} u(i) + a\Delta T\cos(\theta(i)) \\ v(i) + \Delta T a\sin(\theta(i)) \\ y(i) + \Delta T v(i) + \frac{1}{2}\Delta T^2 a\sin(\theta(i)) \\ x(i) + \Delta T u(i) + \frac{1}{2}\Delta T^2 a\cos(\theta(i)) \end{bmatrix}$$

Next we need

$$\begin{aligned}
\Phi_{s(N)}[s(N)] \;&=\; \begin{bmatrix} \phi_{s(N)}[s(N)] \\ \psi_{s(N)}[s(N)] \end{bmatrix} \\
&=\; \begin{bmatrix} \frac{\partial\phi[s(N)]}{\partial u(N)} & \frac{\partial\phi[s(N)]}{\partial v(N)} & \frac{\partial\phi[s(N)]}{\partial y(N)} & \frac{\partial\phi[s(N)]}{\partial x(N)} \\ \frac{\partial\psi[s(N)]}{\partial u(N)} & \frac{\partial\psi[s(N)]}{\partial v(N)} & \frac{\partial\psi[s(N)]}{\partial y(N)} & \frac{\partial\psi[s(N)]}{\partial x(N)} \end{bmatrix} \\
&=\; \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}
\end{aligned}$$

Then we need

$$f_s \;=\; \begin{bmatrix} \frac{\partial f}{\partial u(i)} & \frac{\partial f}{\partial v(i)} & \frac{\partial f}{\partial y(i)} & \frac{\partial f}{\partial x(i)} \end{bmatrix} = \begin{bmatrix} \frac{\partial f_1}{\partial u(i)} & \frac{\partial f_1}{\partial v(i)} & \frac{\partial f_1}{\partial y(i)} & \frac{\partial f_1}{\partial x(i)} \\ \frac{\partial f_2}{\partial u(i)} & \frac{\partial f_2}{\partial v(i)} & \frac{\partial f_2}{\partial y(i)} & \frac{\partial f_2}{\partial x(i)} \\ \frac{\partial f_3}{\partial u(i)} & \frac{\partial f_3}{\partial v(i)} & \frac{\partial f_3}{\partial y(i)} & \frac{\partial f_3}{\partial x(i)} \\ \frac{\partial f_4}{\partial u(i)} & \frac{\partial f_4}{\partial v(i)} & \frac{\partial f_4}{\partial y(i)} & \frac{\partial f_4}{\partial x(i)} \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & \Delta T & 1 & 0 \\ \Delta T & 0 & 0 & 1 \end{bmatrix}$$

and finally

$$f_u \quad = \quad f_\theta = \begin{bmatrix} \frac{\partial f_1}{\partial \theta(i)} \\ \frac{\partial f_2}{\partial \theta(i)} \\ \frac{\partial f_3}{\partial \theta(i)} \\ \frac{\partial f_4}{\partial \theta(i)} \end{bmatrix} = \begin{bmatrix} -a\Delta T \sin(\theta(i)) \\ a\Delta T \cos(\theta(i)) \\ \frac{a}{2}\Delta T^2 \cos(\theta(i)) \\ -\frac{a}{2}\Delta T^2 \sin(\theta(i)) \end{bmatrix}$$

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

1) We will solve the same problem here in both continuous-time and discrete-time form.
a) Assume the continuous time equations of motion

$$\begin{aligned}\dot{u}(t) &= a\cos(\theta(t))\\ \dot{v}(t) &= a\sin(\theta(t)) - g\\ \dot{y}(t) &= v(t)\\ \dot{x}(t) &= u(t)\end{aligned}$$

We want to find $\theta(t)$ to maximize $u(t_f)$ and we want $v(t_f) = 0$, $y(t_f) = y_f$, $x(t_f) = x_f$ and $v(t_f) = 0$, where $a = 1$, $g = 1/3$, $y_f = 0.2$, $x_f = 0.15$ and $t_f = 1.0$. Solve the problem using **fopc.m** and include your code and final plot.

b) Assume the discrete-time equations of motion

$$\begin{aligned}u(i+1) &= u(i) + a\Delta T\cos(\theta(i))\\ v(i+1) &= v(i) + \Delta T\left[a\sin(\theta(i)) - g\right]\\ y(i+1) &= y(i) + \Delta T v(i) + \frac{1}{2}\Delta T^2\left[a\sin(\theta(i)) - g\right]\\ x(i+1) &= x(i) + \Delta T u(i) + \frac{1}{2}\Delta T^2 a\cos(\theta(i))\end{aligned}$$

We want to find $\theta(i)$ to maximize $u(N)$ and we want $v(N) = 0$, $y(N) = y_f$, $x(N) = x_f$ and $v(N) = 0$, where $a = 1$, $g = 1/3$, $y_f = 0.2$, $x_f = 0.15$, $t_f = 1.0$, and $\Delta T = t_f/N$. Solve the problem using **dopc.m** and include your code and final plot.

2) We will solve the same problem here in both continuous-time and discrete-time form.
a) Assume the continuous time equations of motion

$$\begin{aligned}\dot{v}(t) &= a - g\sin(\gamma(t))\\ \dot{y}(t) &= v(t)\sin(\gamma(t))\\ \dot{x}(t) &= v(t)\cos(\gamma(t))\end{aligned}$$

We want to find $\gamma(t)$ to maximize $x(t_f)$ and we want $y(t_f) = y_f$, where $a = 0.5$, $g = 1$, $y_f = 0.1$, and $t_f = 1.0$. Solve the problem using **fopc.m** and include your code and final plot.

b) Assume the discrete-time equations of motion

$$\begin{aligned}v(i+1) &= v(i) + \Delta T\left[\tilde{a} - \sin(\gamma(i))\right]\\ y(i+1) &= y(i) + \Delta l(i)\sin(\gamma(i))\\ x(i+1) &= x(i) + \Delta l(i)\cos(\gamma(i))\\ \Delta l(i) &= v(i)\Delta T + \frac{1}{2}\left[\tilde{a} - \sin(\gamma(i))\right]\Delta T^2\end{aligned}$$

We want to find $\theta(i)$ to maximize $x(N)$ and we want $y(N) = y_f$, where $\tilde{a} = 0.5$, $y_f = 0.1$, $t_f = 1.0$, and $\Delta T = t_f/N$. Solve the problem using **dopc.m** and include your code and final plot.

3) Assume we have a model for heating a room

$$\dot{\theta}(t) = -a(\theta(t) - \theta_a) + bu(t)$$

where $\theta(t)$ is the room temperature at time t, θ_a is the ambient room temperature, a and b are constants, and $u(t)$ is the rate of heat supplied to the room. Defining the state variable

$$x(t) = \theta(t) - \theta_a$$

we have the state equations

$$\dot{x}(t) = -ax(t) + bu(t)$$

Suppose we want to heat the room to a desired final temperature using the minimum energy. We would have

$$J = \frac{1}{2}\int_0^T u^2(t)dt$$

subject to

$$\begin{aligned} \dot{x}(t) &= -ax(t) + bu(t) \\ x(0) &= \theta(0) - \theta_a \\ x_f &= \theta_f - \theta_a \end{aligned}$$

where T is the given final time.

a) Show that

$$\lambda_x(t) = \nu e^{-a(T-t)}$$

b) Show that

$$u(t) = -b\lambda_x(t)$$

and that

$$\frac{d}{dt}\left(x(t)e^{at}\right) = -b^2\lambda_x(t)e^{at}$$

c) Defining

$$\gamma = -b^2\nu e^{-aT}$$

show that

$$x(t) = x(0)e^{-at} + \frac{\gamma}{a}\sinh(at)$$

d) Equating $x(T)$ with x_f, show that we get

$$\nu = \frac{2a}{b^2(1-e^{-2aT})}\left[x_0 e^{-aT} - x_f\right]$$

e) Using the Matlab function **fopc.m** to simulate this system for the case $a = 10, b = 2, x_0 = 10, T = 2, x_f = 20$ Try small values of k (near 10^{-6}), set both tolerances to about 5×10^{-3}, and a maximum of 25 simulations. Compare the true and estimated inputs and states.

f) As the results show, this may not be a very good way to try and heat up a room. Devise a different performance index that still penalizes the energy, but does not produce the rapid drop in temperature we got in part (e). Rerun your simulation for this performance index. (You do not need an analytical solution!) Try to be creative and don't just copy something from a friend.

4) From the text, 3.1.2. Don't worry about part (b). Note: the final answer in (a) is correct, but I think there is an error in the solution.

5) From the text, 3.3.7 (a and b only) Follow the steps in the solutions.