AMS 232 Nonlinear Optimization Hw #3

May 13, 2015

1. Given the state $x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \in \mathbb{R}^2$ and a control $u \in \mathbb{R}$, consider the optimal control problem:

$$P1: \begin{cases} \text{Minimize} & J[x(\cdot), u(\cdot)] = 4x_1(t_f) + x_2(t_f) + 4\int_{t_0}^{t_f} u^2(t)\,dt \\ \text{Subject to} & \dot{x} = \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} x_2^3 \\ u \end{bmatrix} \triangleq f(x,u) \\ & t_0 = 0 \\ & x(t_0) = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \\ & t_f = 2 \end{cases}$$

(**§1**): First we formulate the necessary conditions for (P1) by applying Pontryagin's Minimum Principle.

(a) First, we construct the control Hamiltonian. Since $N_x = 2$ we denote the costates to (P1) as $\lambda = \begin{bmatrix} \lambda_1 \\ \lambda_2 \end{bmatrix}$. By looking at the formulation of (P1), we see that the control Hamiltonian is Time Invariant, so the time argument is suppressed in the control Hamiltonian.

$$\begin{aligned} H(x, \lambda, u) &= F(x,u) + \lambda^T f(x,u) \\ &= 4u^2 + \lambda_1 x_2^3 + \lambda_2 u \end{aligned}$$

(b) Next, the adjoint equations from $\dot{\lambda} = \dfrac{\partial H}{\partial x}$.

$$\dot{\lambda} = \begin{bmatrix} \dot{\lambda}_1 \\ \dot{\lambda}_2 \end{bmatrix} = \begin{bmatrix} \dfrac{\partial H}{\partial x_1} \\ \dfrac{\partial H}{\partial x_2} \end{bmatrix} = \begin{bmatrix} 0 \\ -3\lambda_1 x_2^2 \end{bmatrix}$$

(c) Hamiltonian Minimization Condition (HMC). As the control u is unconstrained, we have that the HMC requires $\dfrac{\partial H}{\partial u} = 0$.

$$0 = \frac{\partial H}{\partial u} = 8u + \lambda_2$$

Therefore the extremal control for (P1), suppressing the superscript-star notation, is $u = -\dfrac{\lambda_2}{8}$. The lower Hamiltonian, obtained by evaluating the control Hamiltonian at the extremal control, is given as

$$\mathcal{H}(x, \lambda) = H\left(x, \lambda, -\frac{\lambda_2}{8}\right) = 4x_2^3 - \frac{\lambda_2^2}{16}$$

* To determine the Hamiltonian Value Condition as well as the Terminal Transversality Condition, we need to find the endpoint Lagrangian $\overline{E}$. Since the only endpoint condition is $t_f = 2$, we have that $N_e = 1$, and so $\nu = [\nu_1]$.

$$\begin{aligned} \overline{E}(x(t_f), \nu) &= E(x(t_f)) + \nu^T e(x(t_f)) \\ &= 4x_1(t_f) + x_2(t_f) + v_1 t_f \end{aligned}$$

(d) Hamiltonian Value Condition, (HVC), $H[@t_f] = \dfrac{\partial \overline{E}}{\partial t_f}$, gives $H[@t_f] = -\nu_1$ which does not give us any meaningful information, which isn't surprising, as the final time t_f is fixed for this problem.

(e) The Hamiltonian Evolution Equation, (HEE), is given as $\dot{\mathcal{H}} = \dfrac{\partial H}{\partial t}$. Since the control hamiltonian is time invariant, we have that $\dfrac{\partial H}{\partial t} = 0$, Therefore the HEE tells us that the lower hamiltonian (which is the control hamiltonian evaluated at the extremal control) is constant for all time.

(f) The Terminal Transversality Condition (TTC) is given as $\lambda(t_f) = \dfrac{\partial \overline{E}}{\partial x(t_f)}$, which gives

$$\begin{bmatrix} \lambda_1(t_f) \\ \lambda_2(t_f) \end{bmatrix} = \begin{bmatrix} \dfrac{\partial \overline{E}}{\partial x_1(t_f)} \\ \dfrac{\partial \overline{E}}{\partial x_2(t_f)} \end{bmatrix} = \begin{bmatrix} 4 \\ 1 \end{bmatrix}$$

(**§2**): We show that the following, (suppressing the superscript-star notation), state-control pair $\{x(\cdot), u(\cdot)\}$ satisfies the necessary conditions, and we identify the corresponding costates; that is, that $\{x(\cdot), u(\cdot)\}$ is an extremal system trajectory:

$$\begin{aligned} x_1(t) &= \frac{2}{5} - \frac{64}{5(2+t)^5} \\ x_2(t) &= \frac{4}{(2+t)^2} \\ u(t) &= -\frac{8}{(2+t)^3} \end{aligned}$$

- Costates: From the TTC and the adjoint equations, we have that $\forall\, t \in [t_0, t_f]$, $\lambda_1(t) = 4$. From the (HMC) we get λ_2 by $\lambda_2(t) = -8u(t) = \dfrac{64}{(2+t)^3}$. With these costates, indeed the TTC and the Adjoint Equations are satisfied: $\lambda_1(t_f = 2) = 4$ and $\lambda_2(t_f = 2) = 1$, and $\dot{\lambda}_1 = 0$, $\dot{\lambda}_2 = \dfrac{-192}{(2+t)^4} = -3\lambda_1 x_2^2$.
- Dynamics: With our x_1 and x_2 given above, by direct calculation, we have that $\dot{x}_1 = x_2^3$ and that $\dot{x}_2 = u = -\frac{\lambda_2}{8}$. Therefore we have that our $x_1, x_2, \lambda_1, \lambda_2$ satisfy the Hamiltonian System

$$\begin{bmatrix} \dot{x} \\ \dot{\lambda} \end{bmatrix} = \begin{bmatrix} \dfrac{\partial H}{\partial \lambda} \\ \dfrac{\partial H}{\partial \lambda} \end{bmatrix}$$

- HMC: Evaluating $\dfrac{\partial H}{\partial u}$ at the given u and λ_2 above, we see that the HMC is satisfied (that it should be zero along the optimal control):

$$\frac{\partial H}{\partial u} = 8u + \lambda_2 = -8\left(\frac{8}{(2+t)^3}\right) + \frac{64}{(2+t)^3} = 0$$

- Lastly, the necessary conditions gives that the Lower Hamiltonian must be a constant. A direct calculation shows that this is true:

$$\mathcal{H} = 4x_2^3 - \left(\frac{1}{16}\right)\lambda_2^2 = \frac{4^4}{(2+t)^6} - \left(\frac{1}{16}\right)\left(\frac{4^6}{(2+t)^6}\right) = 0$$

(**§2**): See the appendix for the implementation of P1 in Dido. The following plots are from running this code. These plots were produced by beginning with a 16 node run with a guess of a straight line and then using that run as an initial guess for a 32 node run.

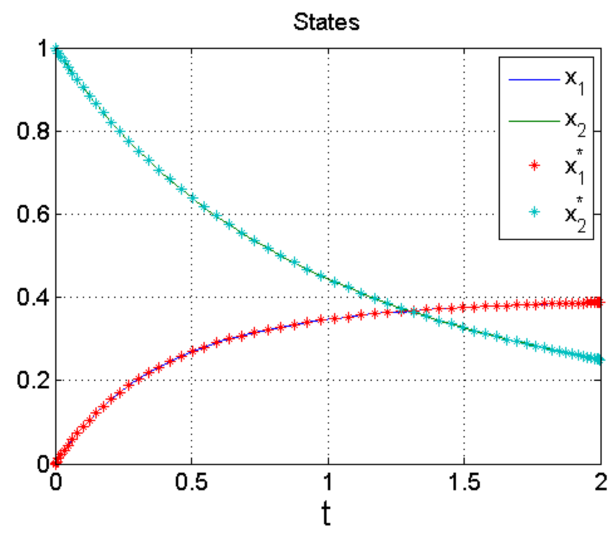

Figure 1: These are the states returned by Dido; the analytical solutions are plotted as as stars. We see that the analytical solutions match the ones returned by Dido. Also we see that Dido has satisfied the initial conditions specified in the problem statement.

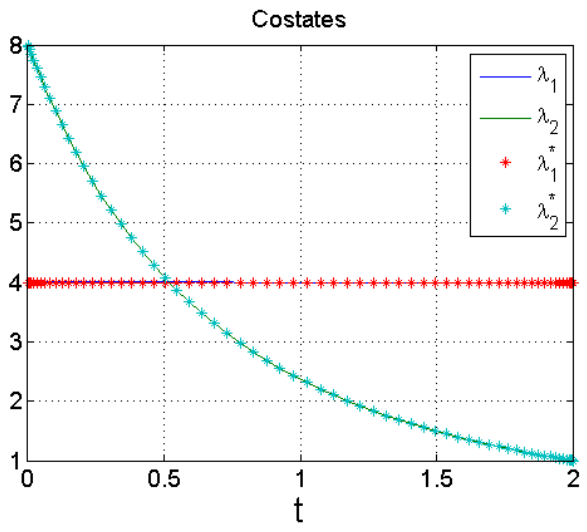

Figure 2: These are the costates returned by Dido, with the analytical solutions added to the graph. We see that the Dido solution and the analytical solution match. Also, the adjoint equations from Pontryagin's Necessary Condition states that the costate associated to x_1 is constant for all time, which is the case. Similarly, the TTC gives that necessarily $\lambda_1 = 4$ for all time, and that λ_2 at the final time must be 1; both of which are satisfied on this plot

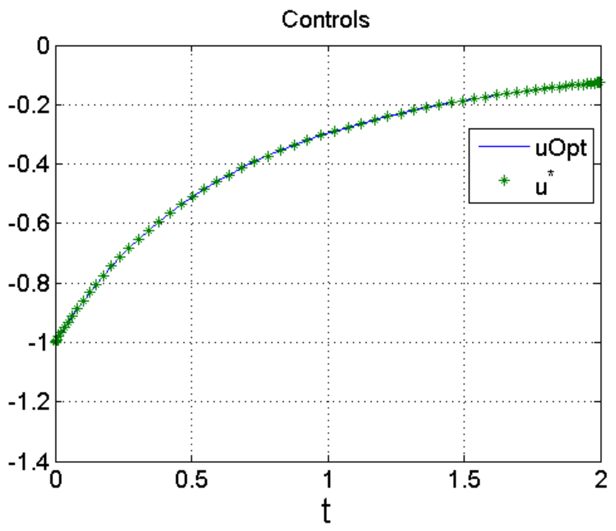

Figure 3: This is the control returned by dido compared to the analytical solution given at the beginning of this section. We see that the analytical solution matches with the control returned by Dido.

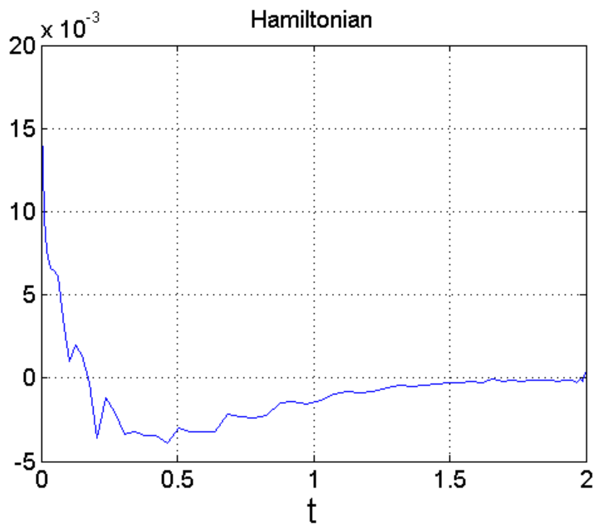

Figure 4: Pontryagin's Minimization Principal gives a necessary condition that the Hamiltonian must be constant for all time, and in this plot, we see that the Hamiltonian returned from Dido is nearly zero; we would say that this necessary condition has been met.

2. Define the state $x^T = \begin{bmatrix} x & y & \theta \end{bmatrix} \in \mathbb{R}^3$ with a control $u = \omega \in \mathbb{R}$, we consider the motion planing problem for Dubin's

Vehicle:

$$P1: \begin{cases} \text{Minimize} & J[x(\cdot),\omega(\cdot),t_f] = \int_{t_0}^{t_f} \omega^2(t)\,dt \\ \text{Subject to} & \dot{x} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \theta \end{bmatrix} = \begin{bmatrix} v\cos\theta \\ v\sin\theta \\ \omega \end{bmatrix} \triangleq f(x,\omega) \\ & t_0 = 0 \\ & x(t_0) = x^0 \in \mathbb{R}^3 \\ & x(t_f) = x^f \in \mathbb{R}^3 \end{cases}$$

Where v is a constant velocity and x^0, x^f are given fixed values for the initial state and final state, and t_f is free.

(a) First we design a scaling to scale-out the constant velocity v. Consider the following approach to scaling out v from problem P1; consider $\hat{t} = \dfrac{t}{K_t}$ and $\hat{\omega} = \dfrac{\omega}{K_\omega}$ where K_t and K_ω are constants to be determined (indeed, constants that will be chosen to scale-out v from P1). We must write all of the problem formulation in terms of hats:
Time: We have that $\hat{t}_0 = \dfrac{t_0}{K_t} = 0$, and $\hat{t}_f = \dfrac{t_f}{K_t}$. The end **Time**: Endpoint Events. Nothing changes here as the state variables are not scaled. What does change, is just the time: $x(\hat{t}_0) = x^0$ and $x(\hat{t}_f) = x^f$ **Dynamics**: Since no element of the state has been scaled, we need to only need to find $\dfrac{d\cdot}{d\hat{t}}$ for each of the state components: x, y, θ.

$$\frac{dx}{d\hat{t}} = \left(\frac{dx}{dt}\right)\left(\frac{dt}{d\hat{t}}\right) = K_t v\cos\theta \tag{1}$$
$$\frac{dy}{d\hat{t}} = \left(\frac{dy}{dt}\right)\left(\frac{dt}{d\hat{t}}\right) = K_t v\sin\theta \tag{2}$$
$$\frac{d\theta}{d\hat{t}} = \left(\frac{d\theta}{dt}\right)\left(\frac{dt}{d\hat{t}}\right) = K_t\omega = K_t K_\omega \hat{\omega} \tag{3}$$

Choosing K_t, K_ω: Since we want to remove the constant v from the dynamics, by looking at 1 we see that by choosing $K_t = \dfrac{1}{v}$ and $K_\omega = v$, we scale-out v from the dynamics. **Cost Function**: J must now be in terms of the scaled variables, so J is a function of $x(\cdot), \hat{\omega}(\cdot)$ and $\hat{t}_f$. Now $d\hat{t} = \frac{1}{K_t}\,dt$ and $\omega^2 = k_\omega^2\hat{\omega}^2$ which gives us

$$\int_{t_0}^{t_f} \omega^2(t)\,dt \longrightarrow \int_{t_0/K_t}^{t_f/K_t} K_\omega^2 \hat{\omega}^2(\hat{t}) K_t\,d\hat{t}$$
$$= \int_{\hat{t}_0}^{\hat{t}_f} \hat{\omega}^2(\hat{t})\,d\hat{t}$$

Where the scalars were dropped since minimization is invariant under scalar multiplication. That is, minimizing J is equivalent to minimizing αJ where $\alpha > 0$ is a scalar.
Synthesizing the above, we have the the new formulation of P1 as

$$P1Scaled: \begin{cases} \text{Minimize} & J[x(\cdot),\hat{\omega}(\cdot),\hat{t}_f] = \int_{\hat{t}_0}^{\hat{t}_f} \hat{\omega}^2(\hat{t})\,d\hat{t} \\ \text{Subject to} & \dfrac{dx}{d\hat{t}} = \begin{bmatrix} \dfrac{dx}{d\hat{t}} \\ \dfrac{dx}{d\hat{t}} \\ \dfrac{dx}{d\hat{t}} \end{bmatrix} = \begin{bmatrix} \cos\theta \\ \sin\theta \\ \hat{\omega} \end{bmatrix} \triangleq f(x,\omega) \\ & \hat{t}_0 = 0 \\ & x(\hat{t}_0) = x^0 \in \mathbb{R}^3 \\ & x(\hat{t}_f) = x^f \in \mathbb{R}^3 \end{cases}$$

(b) We consider the following question: Given an optimal solution $S_1 = \{x(\cdot), \omega(\cdot), t_f\}$ to P1 with $v = v_1$ (i.e.: a specific velocity). Can we construct an optimal solution to P1 where the velocity constant is different, say $v = v_2$ ($v_1 \neq v_2$)? Yes, the answer is yes, and the work in part (a) of this problem hints at how this is true. Given problem P1 with $v = v_1$, we can scale it to problem $P1Scaled$ with $\hat{t} = \frac{t}{K_t}$ and $\hat{\omega} = \frac{\omega}{K_\omega}$ where $K_t = \frac{1}{v_1}$ and $K_\omega = v_1$. Solving $P1Scaled$, we receive the (scaled) state trajectory $S_s = \{x(\cdot), \omega(\cdot), t_f\}$ Then, to recover the solution to S_1 we just have an un-scaling of $\omega = v_1\hat{\omega}$ and $t = \left(\frac{1}{v_1}\right)\hat{t}$. To get the solution when $v = v_2$, we need but change the un-scaling: $\omega = v_2\hat{\omega}$ and $t = \left(\frac{1}{v_2}\right)\hat{t}$. Attached to the appendix, is the scaled version of the Dubin's Vehicle with free end-time.

3. In the last problem, we consider the Brachistochrone problem where $e : \mathbb{R} \times \mathbb{R} \longrightarrow \mathbb{R}$ specifies the target set and $(x^0, y^0, v^0) \in \mathbb{R}^3$. The state is $x = \begin{bmatrix} x \\ y \\ v \end{bmatrix} \in \mathbb{R}^3$ and the control $u = \theta \in \mathbb{R}$, and g is acceleration due to gravity:

$$Brach: \begin{cases} \text{Minimize} & J[x(\cdot), u(\cdot), t_f] = t_f \\ \text{Subject to} & \dot{x} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{v} \end{bmatrix} = \begin{bmatrix} v\sin\theta \\ v\cos\theta \\ g\cos\theta \end{bmatrix} \triangleq f(x,u) \\ & t_0 = 0 \\ & x(t_0) = \begin{bmatrix} x^0 \\ y^0 \\ v^0 \end{bmatrix} \\ & e(x(t_f)) = 0 \end{cases}$$

(a) Suppose that $\{x(\cdot), \theta(\cdot), t_f\}$ is the optimal solution. We will show that there exists $k \in \mathbb{R}$ such that

$$\begin{bmatrix} \dot{x}(t_f) \\ \dot{y}(t_f) \end{bmatrix} = \begin{bmatrix} v(t_f)\sin\theta(t_f) \\ v(t_f)\cos\theta(t_f) \end{bmatrix} = k \begin{bmatrix} \frac{\partial e}{\partial x(t_f)} \\ \frac{\partial e}{\partial y(t_f)} \end{bmatrix} \tag{4}$$

We denote the costates of the Brach problem as $\lambda^T = \begin{bmatrix} \lambda_x & \lambda_y & \lambda_v \end{bmatrix}$. Since H is time invariant, we have that $\frac{\partial H}{\partial t} = 0$, so $H[@t]$ is a constant for all time $t \in [t_0, t_f]$. The HVC gives $H[@t_f] = -1$. Therefore, we have that for all time $t \in [t_0, t_f]$, $H[@t] = -1$. By the TTC, $\lambda_v(t_f) = 0$, and combining this with the fact that $H[@t] = -1$ for all time, we have

$$\lambda_x(t_f)v(t_f)\sin\theta(t_f) + \lambda_y(t_f)v(t_f)\cos\theta(t_f) = -1 \tag{5}$$

Combining $\lambda_v(t_f) = 0$, with the HMC (which gives $\frac{\partial H}{\partial \theta} = 0$), we have

$$\lambda_x(t_f)v(t_f)\cos\theta(t_f) + \lambda_y(t_f)v(t_f)\sin\theta(t_f) = 0 \tag{6}$$

From 5 and 6, we get

$$\begin{aligned} \lambda_x(t_f)v(t_f) &= -\sin\theta(t_f) \\ \lambda_y(t_f)v(t_f) &= -\cos\theta(t_f) \end{aligned}$$

Now, we can write the TTC as $\lambda_x(t_f) = \nu\frac{\partial e}{\partial x(t_f)}$, $\lambda_y(t_f) = \nu\frac{\partial e}{\partial y(t_f)}$, where $\nu \in \mathbb{R}$ as our endpoint Lagrangian as $N_e = 1$. Multiplying the above each by $-v(t_f)$ we have (where $k = -v(t_f)^2\nu$)

$$v(t_f)\sin\theta(t_f) = k\frac{\partial e}{\partial x(t_f)} \tag{7}$$

$$v(t_f)\cos\theta(t_f) = k\frac{\partial e}{\partial y(t_f)} \tag{8}$$

And, since $\{x(\cdot), \theta(\cdot), t_f\}$ is the optimal solution, the states satisfy the dynamics: That is, for all time $t \in [t_0, t_f]$ $\dot{x} = f(x, \theta)$. Therefore

$$\begin{bmatrix} \dot{x}(t_f) \\ \dot{y}(t_f) \end{bmatrix} = \begin{bmatrix} v(t_f) \sin\theta(t_f) \\ v(t_f) \cos\theta(t_f) \end{bmatrix} \tag{9}$$

And combining 7 and 9, we have 4.

(b) The meaning of 4 is that at the endpoint (i.e.: at the end time) the velocity vector (of the state) is parallel to the gradient of the target set at the end time. Geometrically, this means that the curve of the brachistochrone strikes the target set orthogonally at the end time. We demonstrate this by implementing the Brachistochrone problem in Dido with an initial condition $(x(t_0), y(t_0), v(t_0)) = (0, 0, 0)$ and a target set of

$$e(x(t_f), y(t_f)) = (x(t_f) - 5)^2 + (y(t_f) - 1)^2 - 0.1^2$$

which describes the target set as a circle. The following plots were generated from the code given in the appendix. All of the following figures were generated first by running 16 nodes with a guess (which is a straight line from the origin, the to center of the terminal circle), and then increasing the nodes, eventually up to 64 nodes, by using the guess as the previous run.

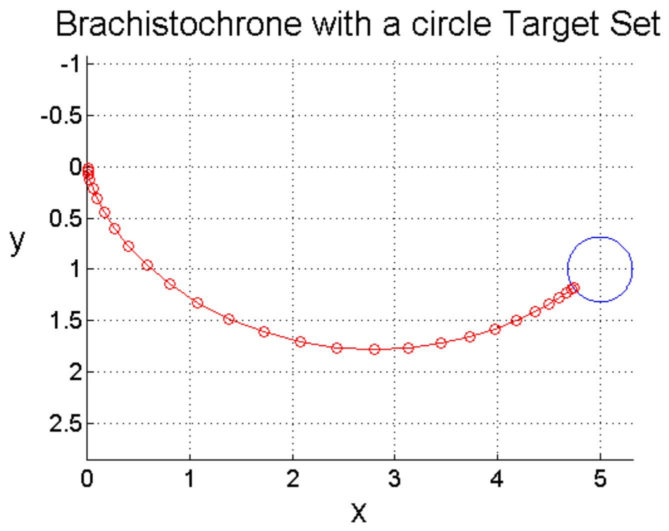

Figure 5: This is the curve returned by Dido that minimizes the transfer time of a particle starting at the origin at rest, and ending on the circle $(x-5)^2 + (y-1)^2 - \left(\frac{1}{\sqrt{10}}\right)^2$. From this plot, we can see that the curve strikes the terminal set in a perpendicular manner.

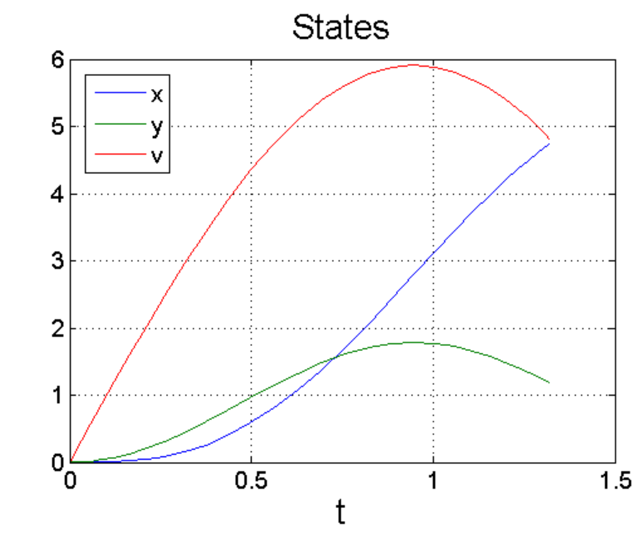

Figure 6: The states to the Brachistochrone problem over time.

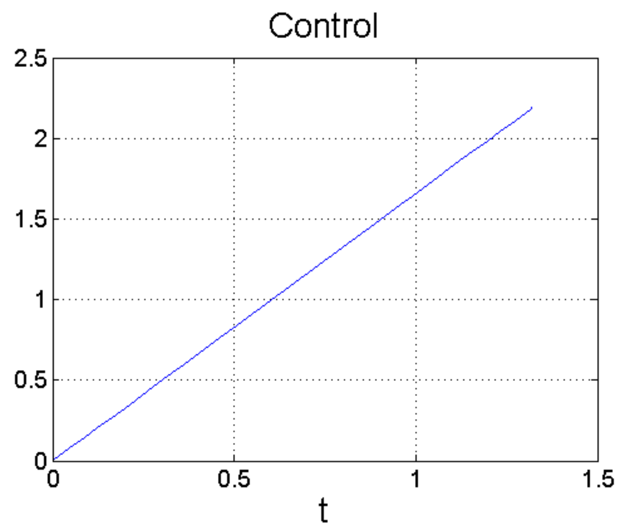

Figure 7: The control θ returned by Dido, which is linear with respect to time t. This matches the derivation in section 3.1 of Ross' text.

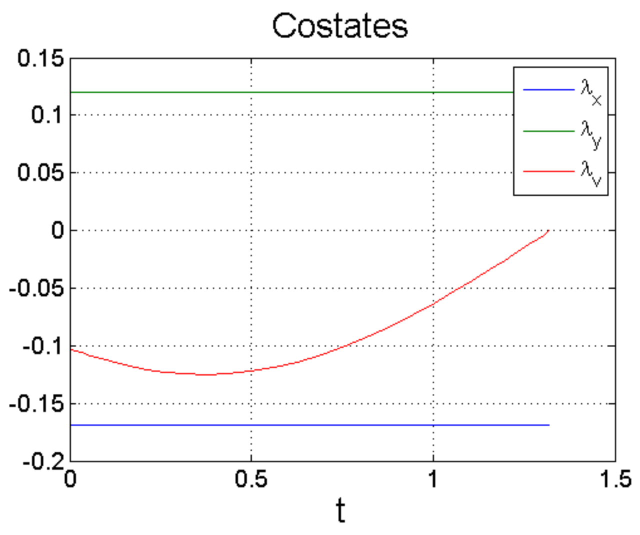

Figure 8: This plot are the costates returned by Dido. Applying Pontryagin's Principal to the Brach problem, gives that extremal solutions necessarily have the costate associated to x and y to be constant for all time. The TTC states that $\lambda_v(t_f) = 0$: We see that the solution returned by Dido satisfies the necessary conditions associated to the costates.

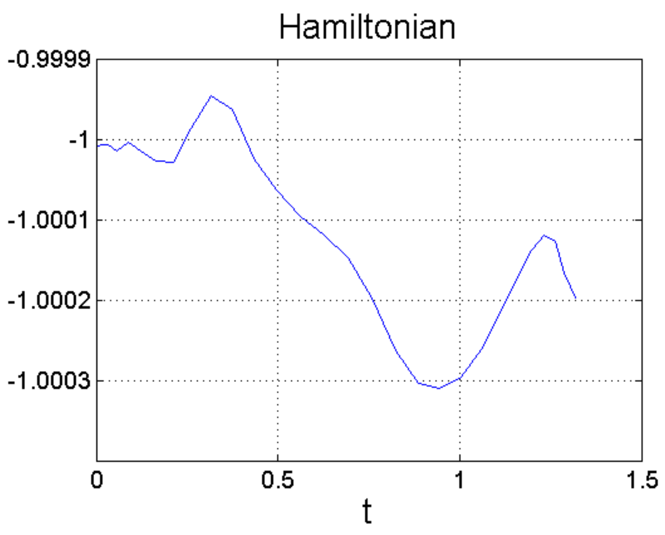

Figure 9: This is the Hamiltonian returned by Dido. Pontryagin's Principal give that, Necessarily the Hamiltonian must be -1 for all time $t \in [t_0, t_f]$, and we see in this figure that the result returned by Dido satisfies this Necessary condition.

Matlab Code

To run each problem set, just make a sperate function-file for each function: costFun, dynamicsFun, eventFun, and one script file for the mainProblemFile.m

Problem1

```
function [ endPointCost, runningCost ] = costFun( primal )

    %Grab the final point of the states x1 and x2. i.e.: x1(tf) and x2(tf)
    x1f = primal.states(1, end);
    x2f = primal.states(2, end);

    u = primal.controls;

    endPointCost = 4*x1f + x2f;
    runningCost = 4*u.^2;

end

function [ XDot ] = dynamicsFun( primal )

    x2 = primal.states(2,:);
    u = primal.controls;

    x1Dot = x2.^3;
    x2Dot = u;

    %Dynamics go into XDot
    XDot = [ x1Dot; x2Dot ];

end

function [ endPointConstraints ] = eventFun( primal )

    %pull out the initial and final value of the state variable x
    x10 = primal.states(1,1);
    x20 = primal.states(2,1);

    endPointConstraints = [x10; x20];

end

%{
    Using Dido, we solve the optCtrl Problem 1 in HW #3
%}
clear all; close all;

%Analytical Extremal Solutions provided in the Homework
x1Star = @(t) 2/5 - 64./(5*(2+t).^5);
x2Star = @(t) 4./(2+t).^2;
uStar  = @(t) -8./(2+t).^3;
lambda1Star = @(t) 0*t+4;
lambda2Star = @(t) 64./(2+t).^3;

Nn = 32;
makeGuess = 0;
loadGuess = 1;


%Setting the box constraints to the states
x1L = -10; x2L = -10;
x1U =  10; x2U =  10;
bounds.lower.states = [x1L; x2L];
bounds.upper.states = [x1U; x2U];


%set the box constraints for the control
uL = -100; uU = 100;
bounds.lower.controls = [uL];
bounds.upper.controls = [uU];


%set the bounds for the initial and final time.  This problem is a fixed time problem.
t0 = 0;  tf = 2;
bounds.lower.time = [t0; tf];
bounds.upper.time = [t0; tf];


%Endpoint constraint (i.e.: 'e' in the problem formulation).
x10 = 0;
x20 = 1;
bounds.lower.events = [x10; x20];
bounds.upper.events = bounds.lower.events;


%Pack the problem data into the problem structure 'optCtrlProb'
optCtrlProb.cost     = 'costFun';
optCtrlProb.dynamics   = 'dynamicsFun';
optCtrlProb.events     = 'eventFun';
optCtrlProb.bounds     = bounds;
```

```
%To use the previous run as an initial guess for the new run
if( loadGuess )
    load optCtrlPrimal primal;
    algorithm.guess = primal;
end

%A guesss to the problem: useful on a cold-run.  Our guess gives initial and final values
%for each state: x: 0 -> L, y:0->0, z:0->L
if( makeGuess )
    guess.states(1,:) = [x10, x1U];
    guess.states(2,:) = [x20, x2U];

    guess.controls(1,:) = [0, 0];

    guess.time     = [t0, tf];

    algorithm.guess = guess;
end

%Here is how to run DIDO in accurate mode.
% algorithm.mode = 'accurate';

%The number of nodes Nn is Nn+1 sample points taken inbetween the initial and
%final time bounds.  Increase Nn for greater accuracy.  Hence Nn+1 points on the
%interval [0,1]
algorithm.nodes = [Nn];

% call dido and record the execution time
tStart= cputime;
    [cost, primal, dual] = dido(optCtrlProb, algorithm);
runTime = cputime-tStart;

%%%%%%%%%%%%%%%%%-----Pretty Plot Code Follows-----%%%%%%%%%%%%%%%%%

disp(strcat('Dido executed in:', ' ' ,num2str(runTime), ' seconds.'));
disp(strcat('Minimized Cost:', ' ' ,num2str(cost), ' cost units.'));

%Unpack the optimal solution from DIDO's primal structure
xOpt = primal.states;
uOpt = primal.controls;
tNodes = primal.nodes;
costates = dual.dynamics;


%%%Pretty Plot Code%%%

figure;
    plot(tNodes, xOpt, ...
         tNodes, x1Star(tNodes), '*', ...
         tNodes, x2Star(tNodes), '*');
    title('States');
    xlabel('t', 'FontSize', 20);
    legend('x_1', 'x_2', 'x_1^*', 'x_2^*');
    grid on;

figure
    plot(tNodes, uOpt, ...
         tNodes, uStar(tNodes), '*');
    title('Controls');
    xlabel('t', 'FontSize', 20);
    legend('uOpt', 'u^*');
    grid on;

figure;
    plot(tNodes, dual.Hamiltonian);
    title('Hamiltonian');
    xlabel('t', 'FontSize', 20);
    grid on;


figure;
    plot(tNodes, costates, ...
         tNodes, lambda1Star(tNodes), '*', ...
         tNodes, lambda2Star(tNodes), '*' );
    title('Costates')
    xlabel('t', 'FontSize', 20);
    legend('\lambda_1', '\lambda_2', '\lambda_1^*', '\lambda_2^*');
    grid on;


save optCtrlPrimal;

%
% end mainProblemFile.m
%
```

Problem2

```
function [ endPointCost, runningCost ] = costFun( primal )

    omegaHat = primal.controls(1,:);

    endPointCost = 0;
    runningCost = omegaHat.^2;

end


function [ XDot ] = dynamicsFun( primal )

    theta = primal.states(3,:);
    omegaHat = primal.controls;

    xDot = cos(theta);
    yDot = sin(theta);
    thetaDot = omegaHat;

    %Dynamics go into XDot
    XDot = [ xDot; yDot; thetaDot ];
end


function [ endPointConstraints ] = eventFun( primal )

    %pull out the initial and final value of the state variable x
    initialStates = primal.states(:,1);
    finalStates = primal.states(:,end);

    %The order of the endpoint constraints is [x0;v0;xf;vf], this has been set in the main
    %file.
    endPointConstraints = [initialStates; finalStates];


end


%{
    Here we look at a Dubins Vehicle: We control the turning rate and minimize the L2 norm
    of the turning rate (smooth ride)
    State: x y theta
    Control: omega (turning rate)
    Dynamics:
         xDot = v*cos(theta)
         yDot = v*sin(theta)
         thetaDot = omega
    So, we have a 3dim state and the control is one dim.

    We are given the initial position and orientation of the car, and we are given the
    final position and and orientation.  We are also given the start and end times.  Given
    this information we look for a curve that satisfies the endpoint constraints (initial
    and final) while minimizing the rate of change of the car turning.
%}
clear all; close all;

global Kt Komega;

v=0.01;

Kt = 1/v;
Komega = v;

Nn = 16;
makeGuess = 1;
loadGuess = 0;
runInAccurateMode = 0;

%Setting the box constraints to the OptCtrl problem.
xL=-10;  yL=-10;  thetaL=-2*pi;
xU=10;   yU=10;   thetaU=2*pi;
bounds.lower.states = [xL;yL;thetaL];
bounds.upper.states = [xU;yU;thetaU];


%Set the box constraints for the control u.
omegaLHat = -100/Komega;
omegaUHat = 100/Komega;
bounds.lower.controls = [omegaLHat];
bounds.upper.controls = [omegaUHat];


%set the bounds for the initial and final time.
t0Hat = 0/Kt;  tfHat = 300/Kt;
bounds.lower.time = [t0Hat; t0Hat];
bounds.upper.time = [t0Hat; tfHat];


%Endpoint constraint (i.e.: 'e' in the problem formulation).  We start and end at a
%specific point and orientation
x0 = 0; y0 = 0;
xf = 1; yf = 1;
theta0 = pi/4;
thetaf = pi/4;
bounds.lower.events = [x0; y0; theta0; xf; yf; thetaf];
bounds.upper.events = bounds.lower.events;


dubinsVeh.cost     = 'costFun';
dubinsVeh.dynamics = 'dynamicsFun';
dubinsVeh.events   = 'eventFun';
dubinsVeh.bounds   = bounds;

%To use the previous run as an initial guess for the new run
if( loadGuess )
    load optCtrlPrimal primal;
    algorithm.guess = primal;
end

%A guesss to the problem: useful on a cold-run.  Our guess gives initial and final values
%for each state: x: 0 -> L, y:0->0, z:0->L
if( makeGuess )
    guess.states(1,:) = [x0, xf];
    guess.states(2,:) = [y0, yf];
    guess.states(3,:) = [theta0, thetaf];

    guess.controls(1,:) = [0, 0];

    guess.time     = [t0Hat, .5*tfHat];

    algorithm.guess = guess;
end

if( runInAccurateMode )
    algorithm.mode = 'accurate';
end

%The number of nodes Nn is Nn+1 sample points taken inbetween the initial and
%final time bounds.  Increase Nn for greater accuracy.  Hence Nn+1 points on the
%interval [0,1]
algorithm.nodes = [Nn];

% call dido and record the execution time
tStart= cputime;
    [cost, primal, dual] = dido(dubinsVeh, algorithm);
runTime = cputime-tStart;

%%%%%%%%%%%%%%%%%-----Pretty Plot Code Follows-----%%%%%%%%%%%%%%%%%

disp(strcat('Dido executed in:', ' ' ,num2str(runTime), ' seconds.'));
disp(strcat('Cost Minimized:', ' ' ,num2str(cost), ' (cost units).'));

%Unpack the optimal solution from DIDO's primal structure
xOpt = primal.states;
omegaOpt = Komega*primal.controls;
tNodes = Kt*primal.nodes;

figure;
    plot(xOpt(1,:), xOpt(2,:));
    title( 'Curve Minimizing Engergy', 'FontSize', 20 );
    xlabel('x', 'FontSize', 20);
    ylabel('y', 'FontSize', 20);
    axis equal;

figure;
    plot(tNodes, xOpt);
    title('States', 'FontSize', 20);
    legend('x', 'y', '\theta');
    xlabel('t');

figure;
    plot(tNodes, omegaOpt)
    title('Unscaled Control', 'FontSize', 20);
    xlabel('t');
    ylabel('u');

figure;
    plot(primal.nodes, primal.controls)
    title('Scaled Control', 'FontSize', 20);
    xlabel('t');
    ylabel('u');

%Plot the Control Hamiltonian.
figure;
plot(tNodes, dual.Hamiltonian);
    title('Hamiltonian', 'FontSize', 20);
    legend('H');
    xlabel('time', 'FontSize', 20);
    ylabel('Hamiltonian Value', 'FontSize', 20);


%Plot the costates
figure;
plot(tNodes, Kt*dual.dynamics);
    title('Costates', 'FontSize', 20)
    xlabel('time', 'FontSize', 20);
    ylabel('costates', 'FontSize', 20);
    legend('\lambda_x', '\lambda_y', '\lambda_\theta');

save optCtrlPrimal;

%
% end mainProblemFile.m
%
```

Problem3

```
function [ endPointCost, runningCost ] = costFun( primal )

    tf = primal.nodes(end);

    endPointCost = tf;
    runningCost = 0;

end


function [ XDot ] = dynamicsFun( primal )

    global g;

    v = primal.states(3,:);
    theta = primal.controls;

    xDot = v.*sin(theta);
    yDot = v.*cos(theta);
    vDot = g*cos(theta);

    %Dynamics go into XDot
    XDot = [xDot; yDot; vDot ];
end


function [ endPointConstraints ] = eventFun( primal )

    global e;

    %pull out the initial and final value of the state variable x
    x0 = primal.states(1,1);
    xf = primal.states(1,end);
    y0 = primal.states(2,1);
    yf = primal.states(2,end);
    v0 = primal.states(3,1);


    %The order of the endpoint constraints, this has been set in the main file.
    endPointConstraints = zeros(4,1);

    endPointConstraints(1) = x0;
    endPointConstraints(2) = y0;
    endPointConstraints(3) = v0;
    endPointConstraints(4) = e(xf,yf);

end


%{
    Using Dido, we solve the brachistochrone problem where we must end on a circle.
%}
clear all; close all;

%{
    g: Gravit Constant , Units m/s^2
    e: Function representing the circle the particle must land upon at the final time.
%}
global g e;


g = 9.8;
h = 5; k = 1; r =1/sqrt(10);
e = @(xf, yf) (xf-h)^2 + (yf-k)^2 - r^2;


Nn = 32;
makeGuess = 0;
loadGuess = 1;

%Setting the box constraints to the states
xL = 0; xU = 20;
yL = 0; yU = 20;
vL = 0; vU = 20;
bounds.lower.states = [xL; yL; vL];
bounds.upper.states = [xU; yU; vU];


%set the box constraints for the control
thetaL = 0; thetaU = pi;
bounds.lower.controls = [thetaL];
bounds.upper.controls = [thetaU];


%set the bounds for the initial and final time.  The final time is free, and the initial
%time is fixed
t0 = 0;  tfMax = 10;
bounds.lower.time = [t0; t0];
bounds.upper.time = [t0; tfMax];


%Endpoint constraints: We are given an initial condition, and must end on a circle.  The
%last constraint defines e(x(tf),y(tf))=0
x0 = 0; y0=0; v0=0;
```

```
bounds.lower.events = [x0; y0; v0; 0];
bounds.upper.events = bounds.lower.events;


%Pack the problem data into the problem structure 'optCtrlProb'
optCtrlProb.cost     = 'costFun';
optCtrlProb.dynamics   = 'dynamicsFun';
optCtrlProb.events     = 'eventFun';
optCtrlProb.bounds     = bounds;

%To use the previous run as an initial guess for the new run
if( loadGuess )
    load optCtrlPrimal primal;
    algorithm.guess = primal;
end

%A guesss to the problem: useful on a cold-run.  Our guess is a straight line to the
%center of the circle we must land upon.
if( makeGuess )
    guess.states(1,:) = [x0, h];
    guess.states(2,:) = [y0, k];
    guess.states(3,:) = [v0, vU];

    guess.controls(1,:) = [0, 0];

    guess.time     = [t0, tfMax];

    algorithm.guess = guess;
end

%Here is how to run DIDO in accurate mode.
% algorithm.mode = 'accurate';

%The number of nodes Nn is Nn+1 sample points taken inbetween the initial and
%final time bounds.  Increase Nn for greater accuracy.  Hence Nn+1 points on the
%interval [0,1]
algorithm.nodes = [Nn];

% call dido and record the execution time
tStart= cputime;
    [cost, primal, dual] = dido(optCtrlProb, algorithm);
runTime = cputime-tStart;

%%%%%%%%%%%%%%%%-----Pretty Plot Code Follows-----%%%%%%%%%%%%%%%%

disp(strcat('Dido executed in:', ' ' ,num2str(runTime), ' seconds.'));
disp(strcat('Minimized Cost:', ' ' ,num2str(cost), ' cost units.'));

%Unpack the optimal solution from DIDO's primal structure
xOpt = primal.states;
uOpt = primal.controls;
tNodes = primal.nodes;


%%% Pretty Plot Code %%%

%Plot the Brach curve, as well as the terminal circle
figure;
hold on;
    tDom = 0:0.01:2*pi;
    plot(h+r*cos(tDom), k+r*sin(tDom));
    plot(xOpt(1,:),xOpt(2,:),'ro-');
    title('Brachistochrone with a circle Target Set','fontSize',20);
    xlabel('x','fontSize',20);
    ylabel('y   ','fontSize',20, 'Rotation', 0);
    set(gca,'YDir','reverse');
    grid on;
    axis equal;
hold off;

figure;
    plot(tNodes, xOpt);
    title('States', 'fontSize',20);
    xlabel('t','fontSize',20);
    grid on;
    legend('x','y','v');

figure
    plot(tNodes, uOpt);
    title('Control', 'fontSize',20);
    grid on;
    xlabel('t','fontSize',20);

figure;
    plot(tNodes, dual.Hamiltonian);
    title('Hamiltonian', 'fontSize',20);
    grid on;
    xlabel('t','fontSize',20);

figure;
    plot(tNodes, dual.dynamics);
    title('Costates', 'fontSize',20);
    xlabel('t','fontSize',20);
    grid on;
    legend('\lambda_x', '\lambda_y', '\lambda_v');
```

```
save optCtrlPrimal;

%
% end mainProblemFile.m
%
```