1 Obstacle Avoidance

Lab Objective: *Solve boundary value problems that arise when using Pontryagin's Maximum principle.*

General Boundary Value Problems

A boundary value problem is a differential equation with a set of constraints. It is similar to initial value problems, which give only initial constraints. An initial value problem may look something like this

$$\begin{aligned} y'' + y' + y &= f(t) \\ y(a) &= \alpha \\ y'(a) &= \beta \\ t &\in [a, b]. \end{aligned}$$

This problem gives a differential equation with initial conditions for y and y'. A boundary value problem may look something like this

$$\begin{aligned} y'' + y' + y &= f(t) \\ y(a) &= \alpha \\ y(b) &= \beta \\ t &\in [a, b], \end{aligned}$$

where we have both right and left hand boundary conditions on y.

Formulating and solving boundary value problems is an important tool when solving many types of problems. This is especially true in the world of variational calculus and optimal control. Many optimal control problems can be formulated as a boundary value problem by using Pontryagin's Maximum Principle, which may greatly simplify the problem.

SciPy has great tools that help us solve boundary value problems. We will be using `solve_bvp` from `scipy.integrate`. Consider the following example

$$y'' + 9y = cos(t), \quad y'(0) = 5, \quad y(\pi) = -\frac{5}{3}. \tag{1.1}$$

We begin by changing this second order ODE into a first order ODE system.

$$\text{Let } y_1 = y \text{ and } y_2 = y' \text{ so that,}$$
$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix}' = \begin{bmatrix} y_2 \\ \cos t - 9y_1 \end{bmatrix}.$$

This formulation allows us to use `solve_bvp()`.

```
from scipy.integrate import solve_bvp
import numpy as np

def ode(t,y):
    ''' define the ode system '''
    return np.array([y[1], np.cos(t) - 9*y[0]])

def bc(ya,yb):
    ''' define the boundary conditions '''
    # ya are the initial values
    # yb are the final values
    # each entry of the return array will be set to zero
    return np.array([ya[1] - 5, yb[0] + 5/3])

# give the time domain
t_steps = 100
t = np.linspace(0,np.pi,t_steps)

# give an initial guess
y0 = np.ones((2,t_steps))


# solve the system
sol = solve_bvp(ode, bc, t, y0)
```

Then we can plot the solution with the following code

```
import matplotlib.pyplot as plt

plt.plot(t, sol.y[0])
plt.xlabel('t')
plt.ylabel('y(t)')
plt.show()
```

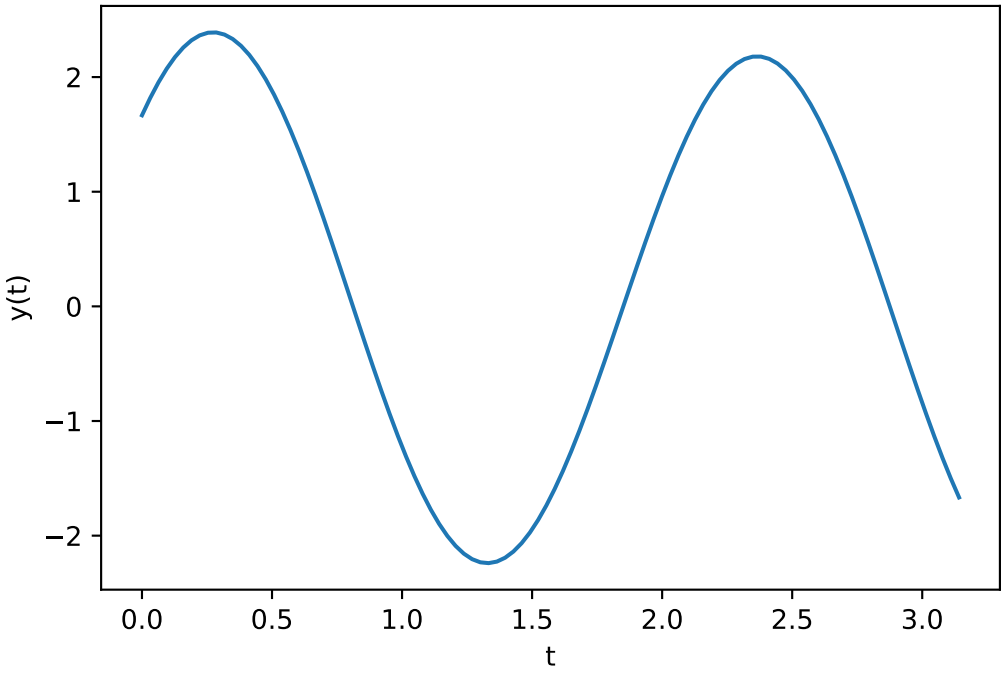

Figure 1.1: The solution to 1.1

Problem 1. Solve the following boundary value problem:

$$y'' + 3y = \sin(t)$$
$$y(0) = 0, \quad y(5\pi) = \frac{\pi}{2}.$$

Plot your solution.

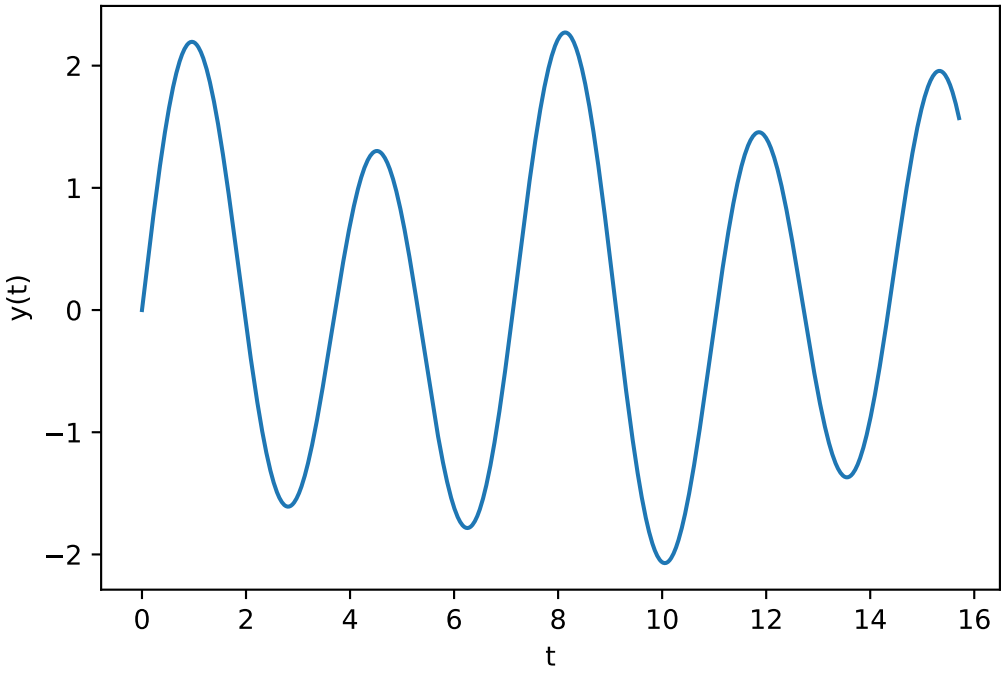

Figure 1.2: The solution to problem 1.

Pontryagin's Maximum Principle

Now that we understand how to solve boundary value problems, we can apply this to solve optimal control problems. Pontryagin's Maximum Principle is a very common way to formulate control problems as BVPs.

Fixed Time, Fixed Endpoint

We will begin with the more simple fixed time horizon problems. Fixed time horizon problems are commonly reformulated as boundary value problems, and we can apply what we have already learned about solving BVPs to make these problems easier to solve. We introduce fixed time horizon problems with a cost functional of the following form

$$J(u) = \int_{t_0}^{t_f} L(t, s(t), u(t))dt + K(t_f, s_f), \tag{1.2}$$

where t_0 and t_f are fixed. In this functional, $L(t, s(t), u(t))$ represents the cost of a certain path determined by the control u, and $K(t_f, s_f)$ is the terminal cost. We also have that

$$\dot{s} = f(t, s, u),\ s_0 = s(t_0),\ s_f = s(t_f). \tag{1.3}$$

In these equations t is time, s is the state variable, and u is the control variable. The maximum principle also uses the Hamiltonian equation

$$H(t, s, u, p) = \langle p, f(t, s, u)\rangle - L(t, s, u), \tag{1.4}$$

where p is a newly introduced variable called the costate. This Hamiltonian is then used to define an ODE system. This first equation defines a costate ODE system

$$\dot{p}^* = -H_s(t, s^*, u^*, p^*), \tag{1.5}$$

where a variable marked with an asterisk is the optimal choice of that variable, meaning that equation 1.5 is only true for the optimal state s^*, costate p^*, and control u^* functions. This next equation will allow us to solve for the control in terms of the state and costate

$$0 = H_u(t, s^*, u^*, p^*), \quad \forall t \in [t_0, t_f]. \tag{1.6}$$

The combination of these equations will allow us to create a BVP that will solve for the optimal control u^* and the associated states s^*. Our ODE comes from 1.3, 1.5, and 1.6, and the boundary values will come from our initial and final conditions on s.

Avoiding Collision

One area of application that relies heavily on optimal control is autonomous driving. A common problem in autonomous driving is the avoidance of obstacles. In this section we will outline a naïve solution to obstacle avoidance with a fixed time horizon.

First we can begin by defining our state variable s. We will want to understand the position and velocity at a given time so we will define the following state variable

$$s(t) = \begin{bmatrix} x(t) \\ y(t) \\ \dot{x}(t) \\ \dot{y}(t) \end{bmatrix} = \begin{bmatrix} s_1(t) \\ s_2(t) \\ s_3(t) \\ s_4(t) \end{bmatrix}, \tag{1.7}$$

which allows us to track those states in $\mathbb{R}^2$.

We can then establish the ODE defined in equation 1.3 by examining $\dot{s}(t)$

$$\dot{s}(t) = \begin{bmatrix} \dot{s_1}(t) \\ \dot{s_2}(t) \\ \dot{s_3}(t) \\ \dot{s_4}(t) \end{bmatrix} = \begin{bmatrix} \dot{x}(t) \\ \dot{y}(t) \\ \ddot{x}(t) \\ \ddot{y}(t) \end{bmatrix},$$

and if we define our control u_1 and u_2 to be acceleration in the x and y directions respectively, then we have

$$\dot{s}(t) = f(t, s, u) = \begin{bmatrix} s_3(t) \\ s_4(t) \\ u_1(t) \\ u_2(t) \end{bmatrix}. \tag{1.8}$$

Next we will define an obstacle. Since we are using integration to define cost, a reasonable way to model an obstacle in this problem would be to use a function. It would be helpful if this function is malleable, allowing us to reposition and resize it, based on the needs of the specific situation. This function also needs to have a large, preferably positive, value in a concentrated location, and it needs to vanish relatively quickly. A decent selection could be a function based on an ellipse, such as this function

$$C(x, y) = \frac{W_1}{((x - c_x)^2/r_x + (y - c_y)^2/r_y)^\lambda + 1}. \tag{1.9}$$

With the function 1.9 we can manipulate the center by changing c_x and c_y, and we can control the size by changing r_x and r_y. Changing the constant W_1 allows us to change the relative penalty of occupying the same location as the obstacle, and a reasonable value for λ will control the vanishing rate. We will also include a term in the cost functional that weights against high acceleration. This will allow us to model the real world more accurately, though the term we will be using is not a perfect representation of real world acceleration limitations. Our cost functional is the following

$$J(u) = \int_{t_0}^{t_f} 1 + C(x(t), y(t)) + W_2 \left|u(t)\right|^2 dt, \tag{1.10}$$

where $W_2 > 0$ defines the relative penalty of high acceleration. This functional will penalize passing near the obstacle and high levels of acceleration.

With the cost functional defined, we can now create the Hamiltonian and the rest of our BVP. We get the following Hamiltonian

$$H(t, p, s, u) = p_1 s_3 + p_2 s_4 + p_3 u_1 + p_4 u_2 - \left(1 + C(x, y) + W_2 \left|u(t)\right|^2\right), \tag{1.11}$$

which gives the following costate ODE by equation 1.5

$$\dot{p} = \begin{bmatrix} \dot{p}_1 \\ \dot{p}_2 \\ \dot{p}_3 \\ \dot{p}_4 \end{bmatrix} = \begin{bmatrix} C_x(x, y) \\ C_y(x, y) \\ -p_1 \\ -p_2 \end{bmatrix}. \tag{1.12}$$

Since we're given $H_u = 0$ in equation 1.6, then we also have the following relations

$$\begin{aligned} u_1(t) &= \frac{1}{2W_2} p_3(t) \\ u_2(t) &= \frac{1}{2W_2} p_4(t). \end{aligned} \tag{1.13}$$

Problem 2. Using the ODEs found in 1.8 and 1.12, the obstacle function 1.9, and the following boundary conditions and parameters solve for and plot the optimal path.

$$\begin{aligned} t_0 = 0, \quad t_f &= 20 \\ (c_x, c_y) &= (4, 1) \\ (r_x, r_y) &= (5, .5) \\ \lambda &= 20 \end{aligned}$$

$$s_0 = \begin{bmatrix} 6 \\ 1.5 \\ 0 \\ 0 \end{bmatrix}, \quad s_f = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

You will need to choose a W_1 and W_2 which allow the solver to find a valid path. If these parameters are not chosen correctly, the solver may find a path which goes through the obstacle, not around it. Plot the obstacle using `plt.contour()` to see be certain path doesn't pass through the obstacle.

Hint: The default for a parameter of `solve_bvp` called `max_nodes` is not large enough.

Try at least `max_nodes = 30000`. You may also find it helpful to use the function `partial` from the module `functools` to preset the parameters for the functions you will be using.

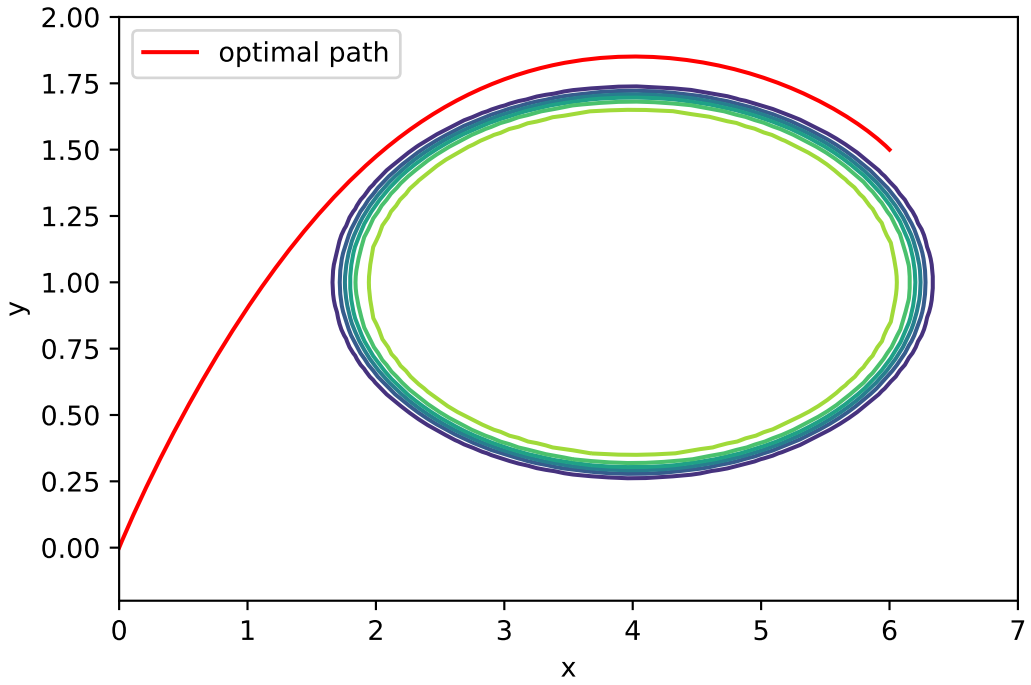

Figure 1.3: Solution to problem 2 for certain choice of parameters

Free Time Horizon Problems

In the previous sections and problems, we were working with BVPs that had a fixed start time t_0, and a fixed end time t_f. However, we may also encounter systems that have a free end time. In order to solve these problems we will need to make some alterations to the problem. First we will perform a change of basis so that we can work with a fixed end time. Consider the following system

$$\dot{x}(t) = f(x(t), t) \quad t \in [0, t_f],$$

we can do the following change of basis for the time variable

$$\begin{aligned} t &= t_f \hat{t} \\ \implies \frac{d}{dt} &= \frac{d}{d\hat{t}} \frac{d\hat{t}}{dt} \\ \implies \frac{d}{dt} &= \frac{d}{d\hat{t}} \frac{1}{t_f}. \end{aligned}$$

We can now define $z(\hat{t}) := x(t_f \hat{t})$ which gives us the following new system

$$\dot{z}\left(\hat{t}\right) = t_f f\left(z\left(\hat{t}\right), \hat{t}\right) \quad \hat{t} \in [0, 1].$$

This system can be solved in the same way we solve the fixed time horizon problems. But you may notice that we now have an extra unknown parameter, the final time. Because of this, a free time horizon problem will need one more boundary value to make the system solvable.

So lets examine the earlier example as a free time horizon problem. We start with the ODE system we derived from the second order equation, replacing the fixed final time with a free final time and including the needed third boundary condition

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix}' = \begin{bmatrix} y_2 \\ cos(t) - 9y_1 \end{bmatrix}, \quad y_1(0) = 5/3, \quad y_2(0) = 5, \quad y_1(t_f) = -\frac{5}{3}.$$

Now we make the coordinate change giving the following system

$$\begin{bmatrix} z_1 \\ z_2 \end{bmatrix}' = t_f \begin{bmatrix} z_2 \\ cos(\hat{t}) - 9z_1 \end{bmatrix}, \quad z_1(0) = 5/3, \quad z_2(0) = 5, \quad z_1(1) = -\frac{5}{3}. \tag{1.14}$$

Now we can solve this system using `solve_bvp` in python. The new argument `p` that we have included in `ode()` and `bc()` is an `ndarray` that contains our parameter t_f.

```
def ode(t,y,p):
    ''' define the ode system '''
    return p[0]*np.array([y[1], np.cos(t) - 9*y[0]])

def bc(ya,yb,p):
    ''' define the boundary conditions '''
    return np.array([ya[0] - (5/3), ya[1] - 5, yb[0] + 5/3])

# give the time domain
t_steps = 100
t = np.linspace(0,1,t_steps)

# give an initial guess
y0 = np.ones((2,t_steps))
p0 = np.array([6])

# solve the system
sol = solve_bvp(ode, bc, t, y0, p0)
```

The attribute `sol.p[0]` will give the final time the solver found.

When plotting we need to make sure that we remember that $x(t_f\hat{t}) = z(\hat{t})$, so we plot in the following way

```
plt.plot(sol.p[0]*t,sol.sol(t)[0])
plt.xlabel('t')
plt.ylabel('y(t)')
plt.show()
```

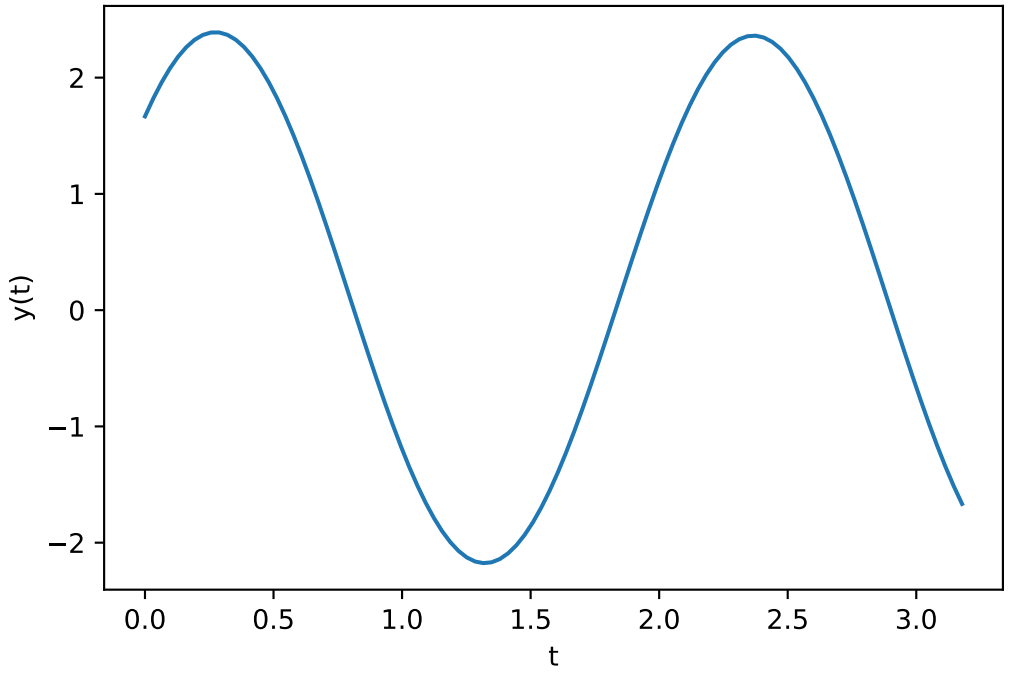

Figure 1.4: The solution to 1.14

Problem 3. Solve the following boundary value problem:

$$y'' + 3y = \sin(t)$$
$$y(0) = 0, \quad y(t_f) = \frac{\pi}{2}, \quad y'(t_f) = \frac{1}{2}\left(\sqrt{3}\pi \cot(\pi\sqrt{75}) - 1\right).$$

Plot your solution. What t_f did the solver find?

Free Time, Fixed Endpoint Control Problems

Now that we understand how to formulate free time horizon problems, we can modify our optimal control BVP to become a free time horizon problem. This is actually the best way to formulate many optimal control problems, as we usually don't know exactly how long it takes to traverse the optimal path. The methodology is exactly the same as we used in the last problem, we only need to find the extra boundary value which will allow us to make the end time a free variable.

To find this extra boundary value we will use the fact that the Hamiltonian is 0 for all t along the optimal path. It is standard to use the final time as the representative so we will assert that

$$H(t_f, p(t_f), s(t_f), u(t_f)) = 0. \tag{1.15}$$

You may notice that when you solve an optimal control problem as a free end time BVP, the optimal path you get is different than what you found when it was a fixed end time BVP. This is

because the free end time solution actually arrives faster. The solution found in the fixed end time formulation is the optimal path for a certain fixed end time, but it may not be the overall fastest path that avoids the obstacle.

Problem 4. Refactor your code from problem 2 to create a free end time BVP and use a new boundary value derived from 1.15. Plot the solution you found. What is the optimal time?

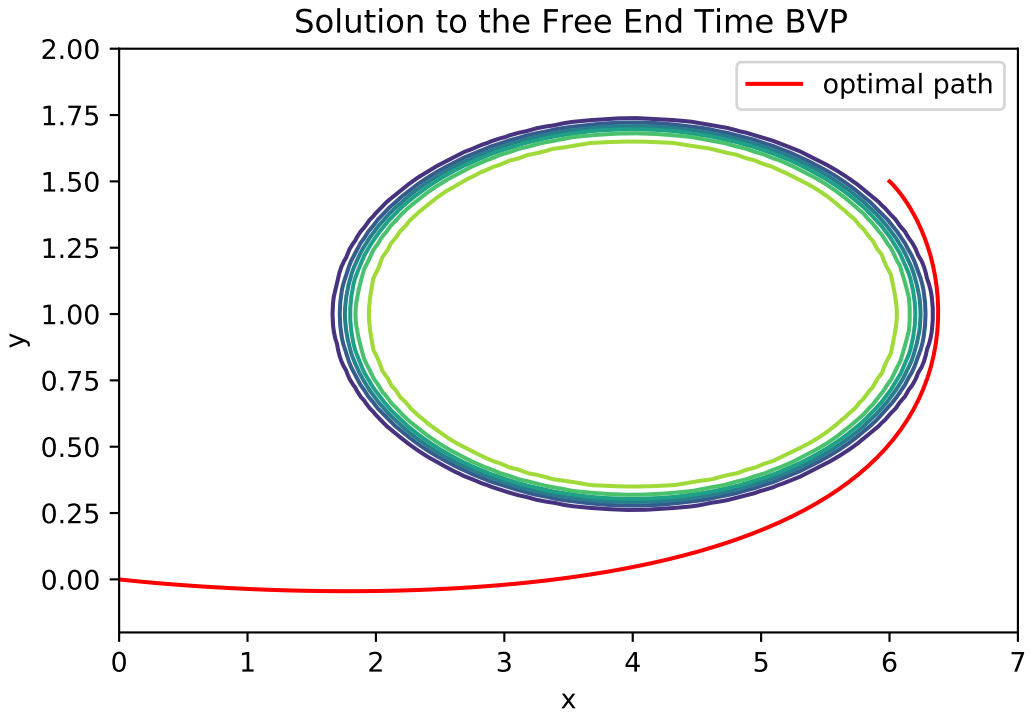

Figure 1.5: The solution to 4