Optimal Control Bonus

Andres Pulido

In [1]:
```
import numpy as np

import matplotlib.pyplot as plt
plt.style.use('seaborn-poster')
%matplotlib inline

from scipy.integrate import solve_ivp
from scipy.optimize import fsolve
```

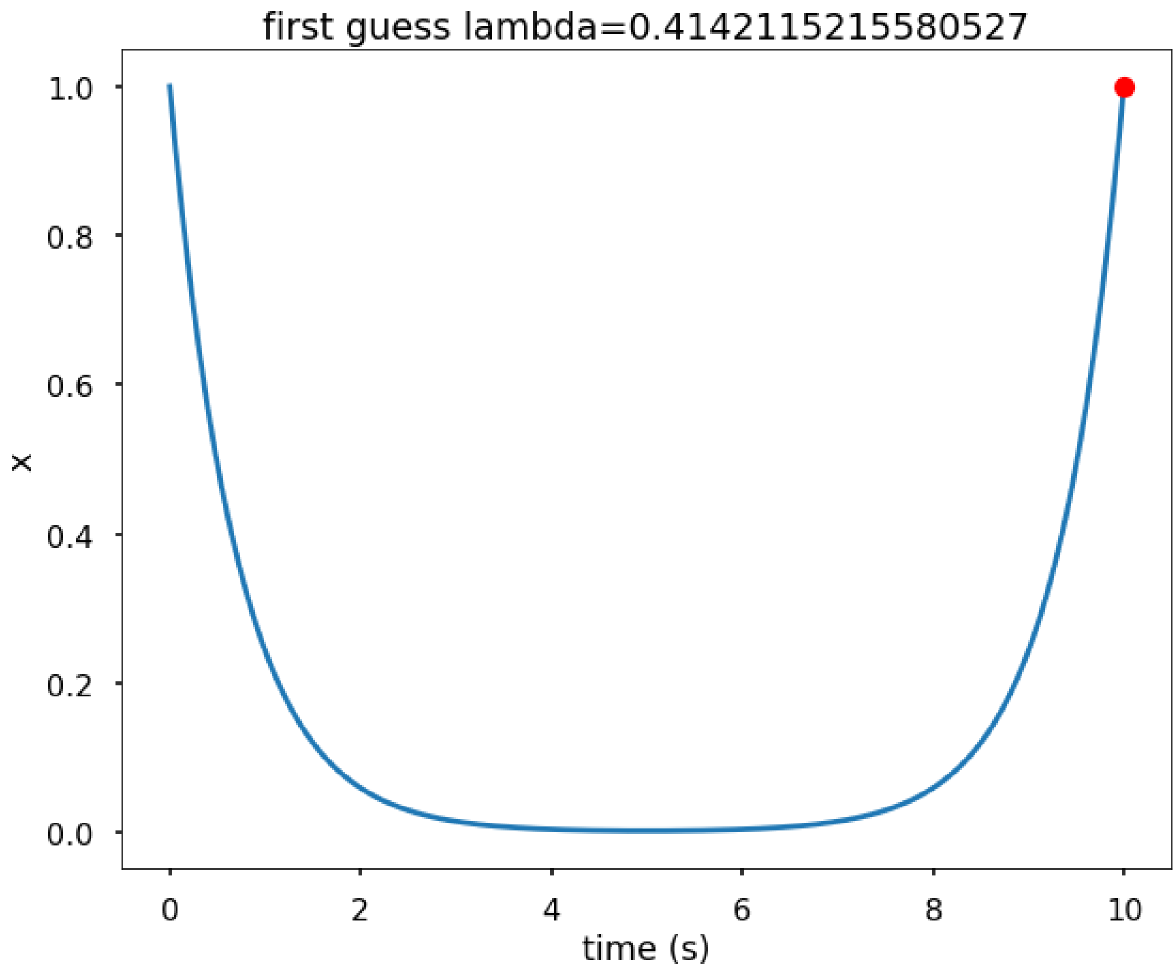
Problem 1

$$\begin{bmatrix} \dot{x} \\ \dot{\lambda} \end{bmatrix} = \begin{bmatrix} -1 & -1 \\ -1 & +1 \end{bmatrix} \begin{bmatrix} x \\ \lambda \end{bmatrix}$$

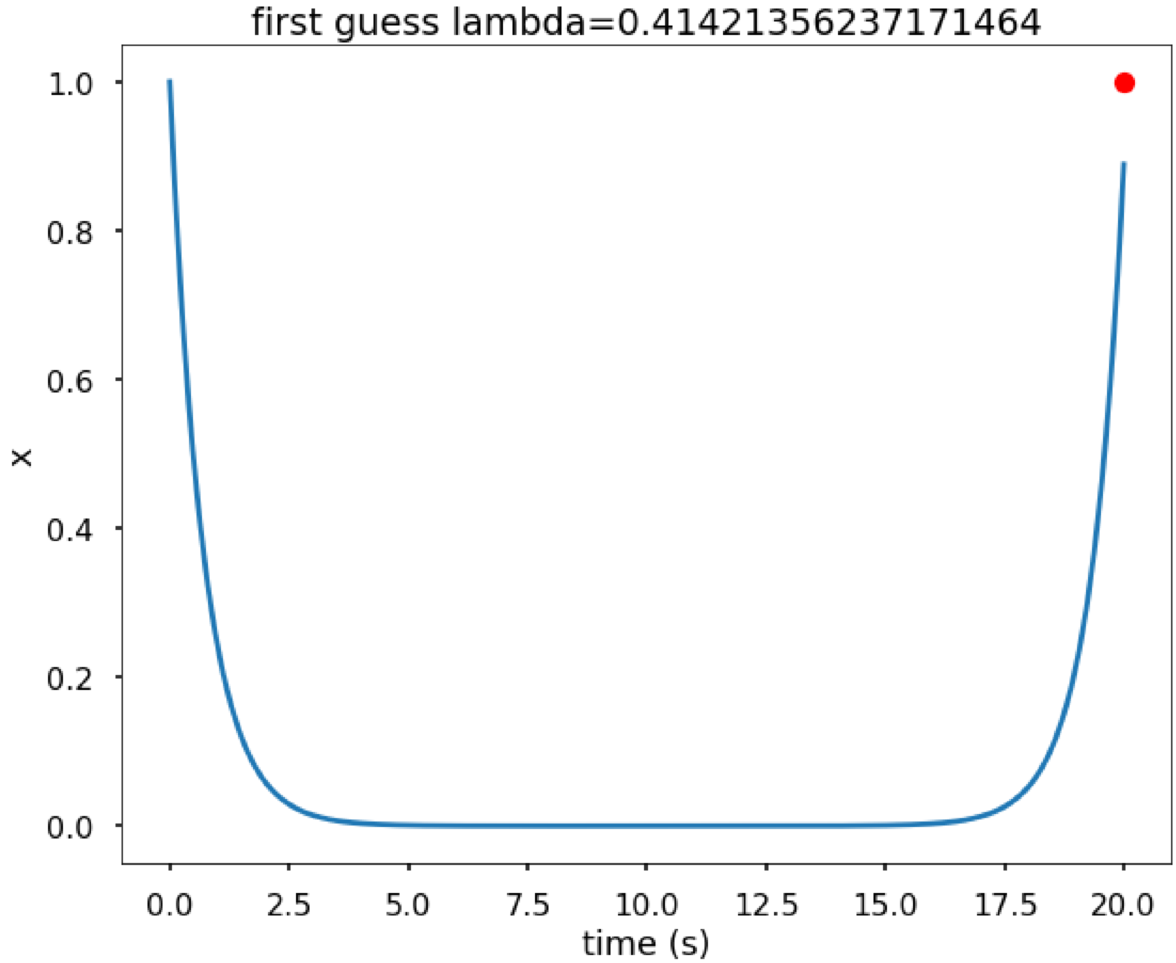
with the two boundary conditions are: $x(0) = 1$ and $x(t_f) = 1$.

In [2]:
```
def shooting_method(F, x0, lamb0_guess, xf, tf):
    t_eval = np.linspace(0, tf, 1000)

    def objective(lamb0):
        sol = solve_ivp(F, [0, tf], \
                [x0, lamb0], t_eval = t_eval)
        x = sol.y[0]
        return x[-1] - xf  # Error

    lamb0, = fsolve(objective, lamb0_guess)
    sol = solve_ivp(F, [0, tf], [x0, lamb0], t_eval = t_eval)

    plt.figure(figsize = (10, 8))
    plt.plot(sol.t, sol.y[0])
    plt.plot(tf, xf, 'ro')
    plt.xlabel('time (s)')
    plt.ylabel('x')
    plt.title(f'first guess lambda={lamb0}')
    plt.show()
```

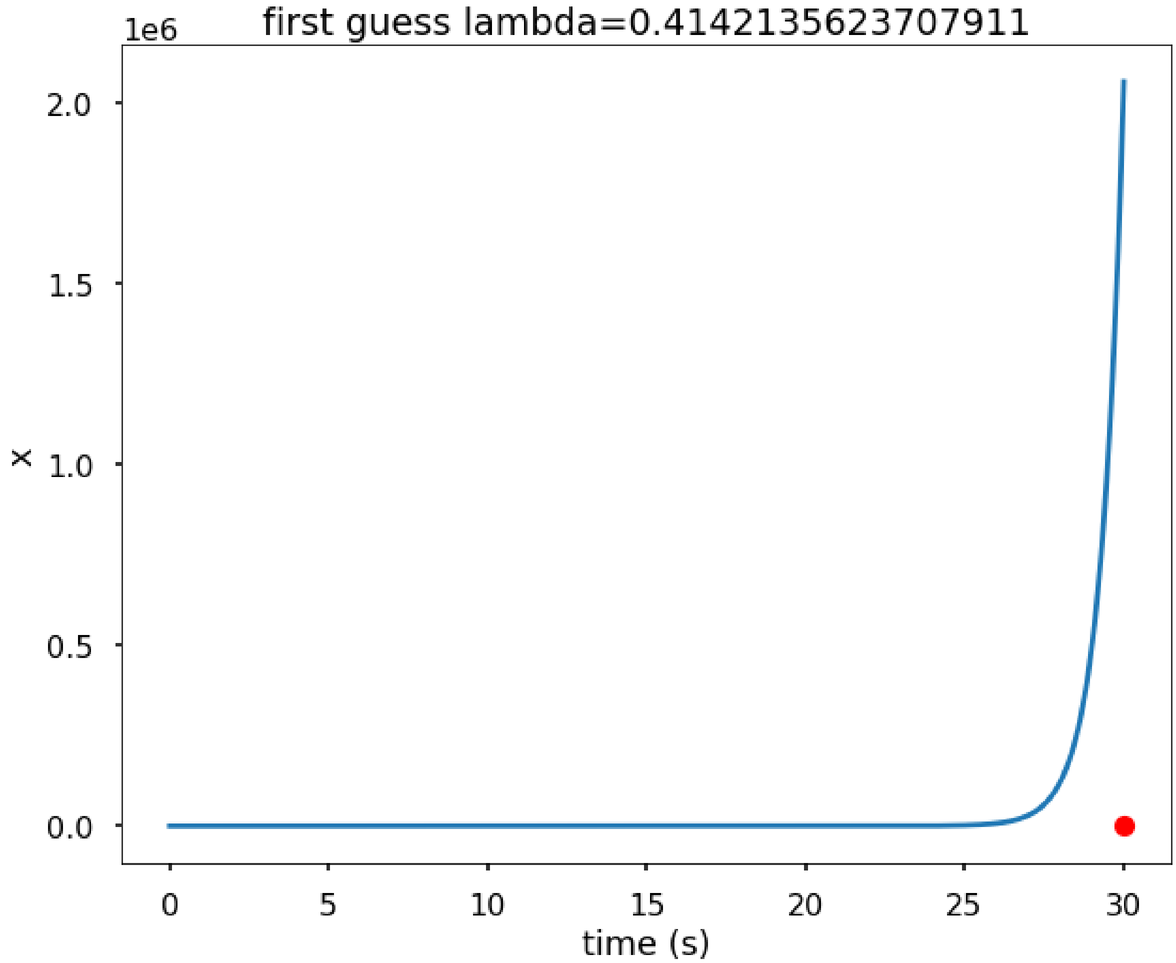
In [3]:
```
F = lambda t, s: \
  np.dot(np.array([[-1,-1],[-1,1]]),s)

x0 = 1
xf = 1
lamb0_guess = 0  # guess


tf = [10, 20, 30, 40, 50]

shooting_method(F, x0, lamb0_guess, xf, tf[0])
```



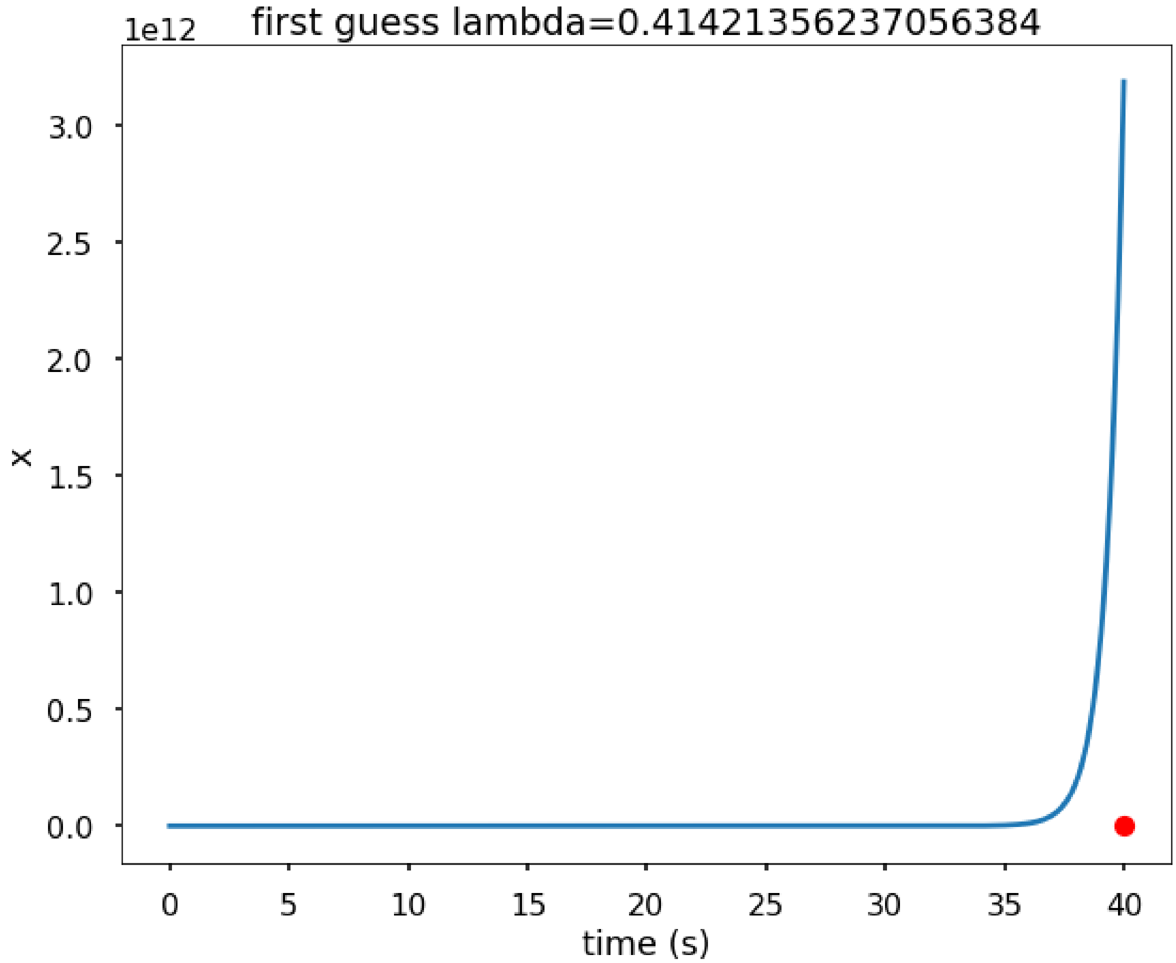
In [4]:
```
shooting_method(F, x0, 10, xf, tf[1])
```



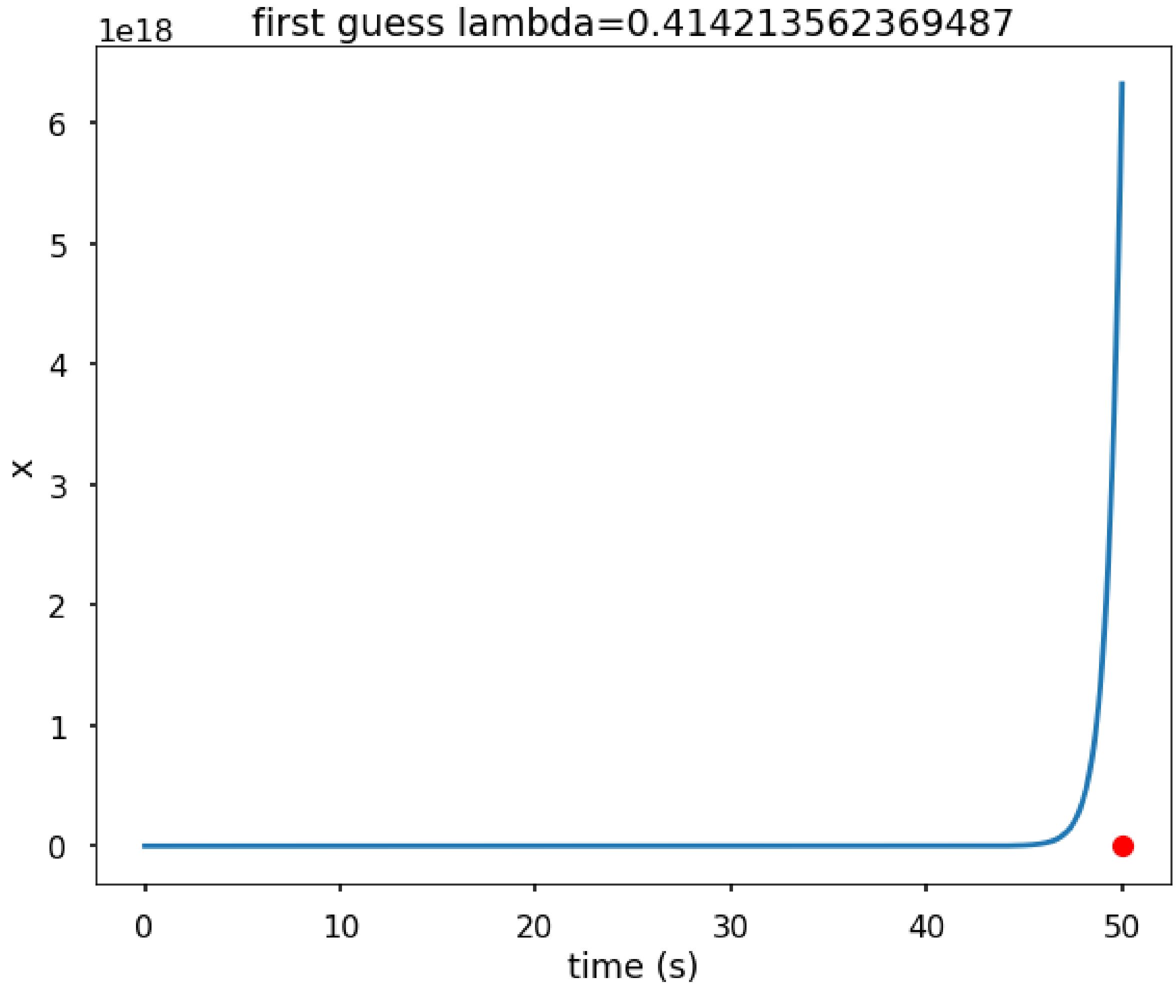
In [5]:
```
shooting_method(F, x0, 11.5, xf, tf[2])
```



In [6]:
```
shooting_method(F, x0, 11.5, xf, tf[3])
```



In [7]:
```
shooting_method(F, x0, 11.5, xf, tf[4])
```



Analysis

The analytic solution for this can be found when the control u is replaced by $u = \dot{x} + x$. The full solution is shown in `Pulido_bonus.pdf` . In the pdf is shown that the solution of the x_f is solved by computing $e^{\sqrt{2}t_f}$ so that demonstrates why after $t_f = 30$ the computer cannot precisely calculate the solution, which is why the graph looks likke the curve is not hitting the red dot after converging.

Problem 2

$$
\begin{gathered}
\dot{x} = v\cos\theta \\
\dot{y} = v\sin\theta \\
\dot{v} = g\cos\theta \\
\dot{\lambda}_x = 0 \\
\dot{\lambda}_y = 0 \\
\dot{\lambda}_v = -\lambda_x \cos\theta - \lambda_y \cos\theta
\end{gathered}
$$

with the intitial conditions: $x(0) = 0$, $y(0) = 0$, $v(0) = 0$.
Equation:

$$
\begin{gathered}
x(t_f) - x_f = x(t_f) - 2 = 0 \\
y(t_f) - y_f = y(t_f) - 2 = 0 \\
\lambda_v(t_f) = 0 \\
H(t_f) + 1 = 0
\end{gathered}
$$

In [12]:
```
def brachistocrone_shooting_method(initial_states, final_states, guesses
    def objective(obj0):
        global theta_array
        print(" - lambdax = ", obj0[0])
        print(" - lambday = ", obj0[1])
        print(" - lambdav = ", obj0[2])
        print(" - tf = ", obj0[3])
        print("")
        tf = obj0[3]
        t_eval = np.linspace(0, tf, 1000)

        sol = solve_ivp(F1, [0, tf],
                        [x0, y0, v0, obj0[0], obj0[1], obj0[2]], t_eval=
        x = sol.y[0]
        y = sol.y[1]
        v = sol.y[2]
        lambx = sol.y[3]
        lamby = sol.y[4]
        lambv = sol.y[5]

        H = lambx[-1]*v[-1]*np.sin(theta_array[-1]) + lamby[-1] * \
            v[-1]*np.cos(theta_array[-1]) + lambv[-1]*g*np.cos(theta_ar
        theta_array = []

        eq1 = x[-1] - xf
        eq2 = y[-1] - yf
        eq3 = lambv[-1]
        eq4 = H + 1
        #print([eq1, eq2, eq3, eq4])
        return [eq1, eq2, eq3, eq4]

    def F1(t, s):
        global theta, theta_array

        v_t = s[2]
        lambx_t = s[3]
        lamby_t = s[4]
        lambv_t = s[5]

        def solve_control(th):
            ht = lambx_t*v_t*np.cos(th) - (lamby_t*v_t + lambv_t*g)*np.s
            # print(ht)
            return ht

        theta, = fsolve(solve_control, theta_guess)
        theta_array.append(theta)

        x_dot = v_t*np.sin(theta)
        y_dot = v_t*np.cos(theta)
        v_dot = g*np.cos(theta)
        lambx_dot = 0
        lamby_dot = 0
        lambv_dot = -s[3]*np.cos(theta) - s[4]*np.cos(theta)
        return [x_dot, y_dot, v_dot, lambx_dot, lamby_dot, lambv_dot]

    global x0, y0, v0, xf, yf, theta_guess, theta_array
```

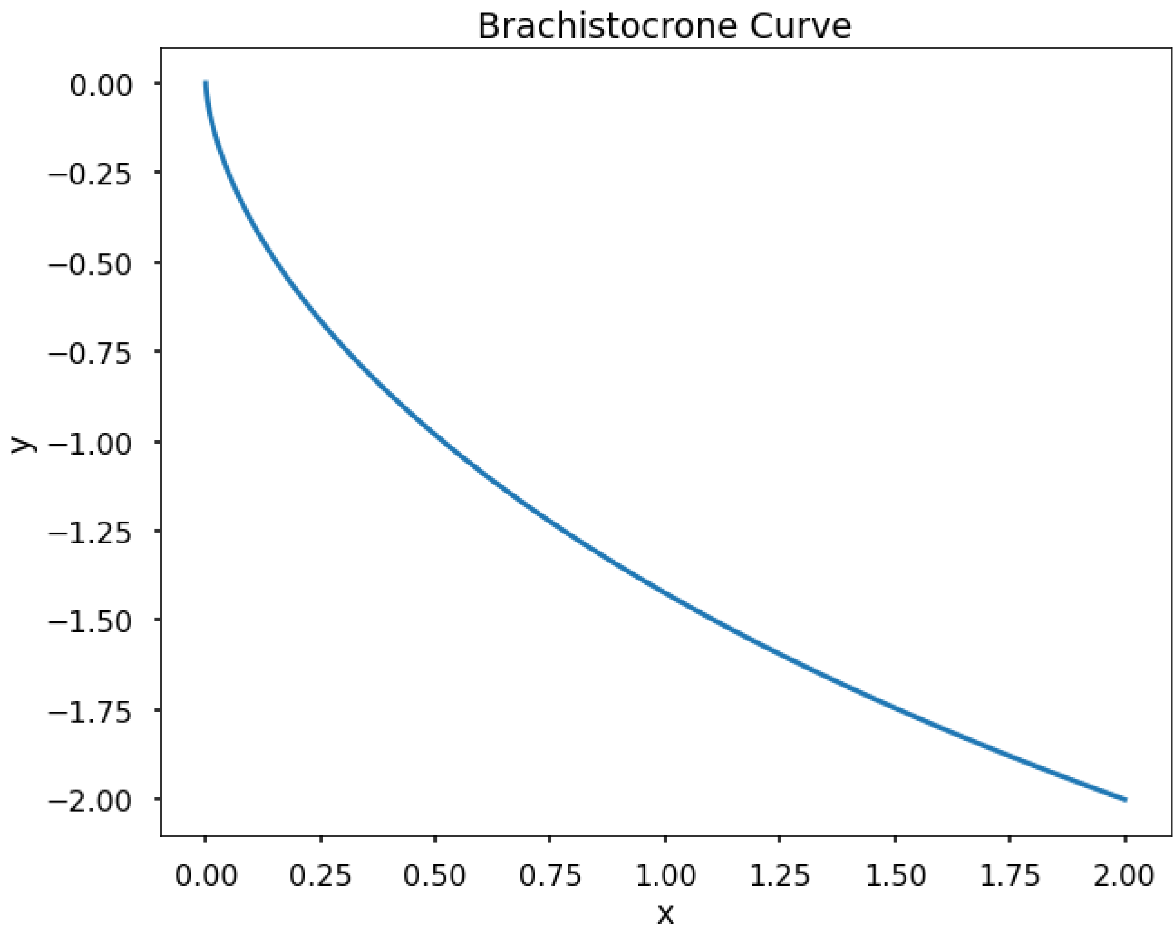
```
    theta_guess = guesses[4]
    theta_array = []

    xf = final_states[0]
    yf = final_states[1]
    x0 = initial_states[0]
    y0 = initial_states[1]
    v0 = initial_states[2]

    obj_sol = fsolve(objective, guesses[0:4])

    # x0, y0, v0, lambx0, lamby0, lambv0
    sol_initial_states = [x0, y0, v0, obj_sol[0], obj_sol[1], obj_sol[2

    tf = obj_sol[3]
    t_eval = np.linspace(0, tf, 100)

    sol = solve_ivp(F1, [0, tf], sol_initial_states, t_eval=t_eval)

    plt.figure(figsize=(10, 8))
    plt.plot(sol.y[0], -1*sol.y[1])
    plt.xlabel('x')
    plt.ylabel('y')
    plt.title(f'Brachistocrone Curve')
    plt.show()

    plt.figure(figsize=(10, 8))
    plt.plot(sol.t, sol.y[0], label='x')
    plt.plot(sol.t, sol.y[1], label='y')
    plt.plot(sol.t, sol.y[2], label='v')
    plt.xlabel('t')
    plt.ylabel('Value')
    plt.title(f'States')
    plt.legend()
    plt.show()
```

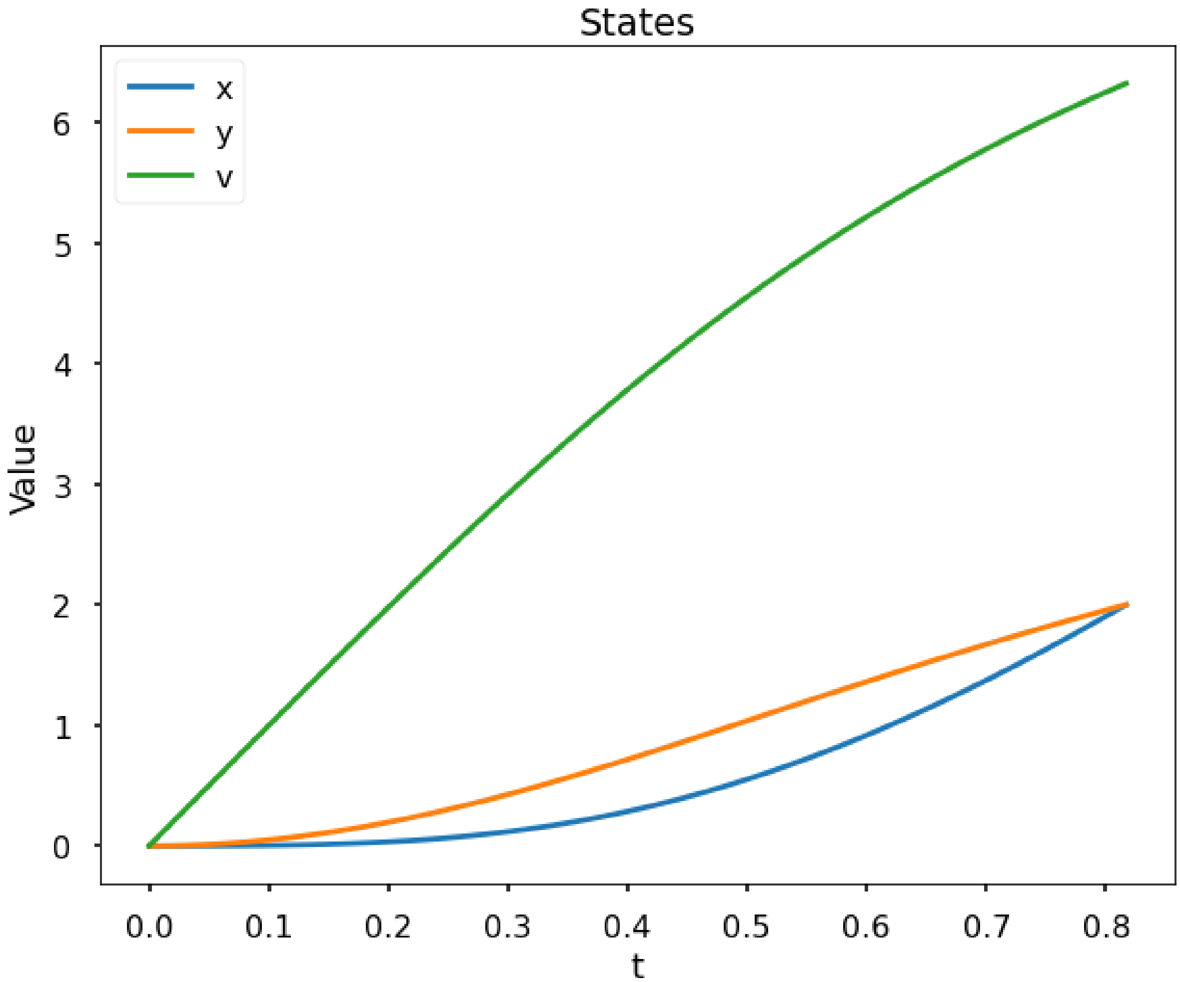
In [13]:
```
global g
g = 10

x0 = 0
xf = 2
y0 = 0
yf = 2
v0 = 0
#vf = free

initial_states = [x0, y0, v0]
final_states = [xf, yf]

lamb0_guess = [1, 1, 1]
tf_guess = [1]
theta_guess = [0.5]

guesses = lamb0_guess + tf_guess + theta_guess  # list of element guesse

brachistocrone_shooting_method(initial_states, final_states, guesses)
```

```
 - lambdax =  1
 - lambday =  1
 - lambdav =  1
 - tf =  1

 - lambdax =  1.0
 - lambday =  1.0
 - lambdav =  1.0
 - tf =  1.0

 - lambdax =  1.0
 - lambday =  1.0
 - lambdav =  1.0
 - tf =  1.0

 - lambdax =  1.0000000149011612
 - lambday =  1.0
 - lambdav =  1.0
 - tf =  1.0

 - lambdax =  1.0
 - lambday =  1.0000000149011612
 - lambdav =  1.0
 - tf =  1.0

 - lambdax =  1.0
 - lambday =  1.0
 - lambdav =  1.0000000149011612
 - tf =  1.0

 - lambdax =  1.0
 - lambday =  1.0
 - lambdav =  1.0
 - tf =  1.0000000149011612

 - lambdax =  -0.024285284332479895
```

```
- lambday =  0.1255488892558394
- lambdav =  -0.10559319742920659
- tf =  0.8312964657341566

- lambdax =  0.4254083770734707
- lambday =  -0.37265539160848166
- lambdav =  -0.34957108816065663
- tf =  1.0002353267818003

- lambdax =  1.7164650847406846
- lambday =  -0.4251132650260758
- lambdav =  0.5619267117839291
- tf =  1.2343633010644164

- lambdax =  -0.02428528397060096
- lambday =  0.1255488892558394
- lambdav =  -0.10559319742920659
- tf =  0.8312964657341566

- lambdax =  -0.024285284332479895
- lambday =  0.12554889112666365
- lambdav =  -0.10559319742920659
- tf =  0.8312964657341566

- lambdax =  -0.024285284332479895
- lambday =  0.1255488892558394
- lambdav =  -0.10559319585574534
- tf =  0.8312964657341566

- lambdax =  -0.024285284332479895
- lambday =  0.1255488892558394
- lambdav =  -0.10559319742920659
- tf =  0.8312964781214393

- lambdax =  -0.11510342777752704
- lambday =  -0.07293458976210376
- lambdav =  -0.172245214284687
- tf =  0.7053843571998742

- lambdax =  -0.1876371646964321
- lambday =  -0.09943429713319331
- lambdav =  -0.1803871775863795
- tf =  0.7164560739139574

- lambdax =  -0.2164834211676947
- lambday =  -0.06357786747048325
- lambdav =  -0.1517067776332392
- tf =  0.7765547701538128

- lambdax =  -0.1938307740105947
- lambday =  -0.057405390590041516
- lambdav =  -0.13655880460692704
- tf =  0.8184462614061122

- lambdax =  -0.07331500136749781
- lambday =  -0.10263088481548611
- lambdav =  -0.13993479684212998
```

```
- tf =  0.8327108173199322

- lambdax =  -0.15476641455612547
- lambday =  -0.05703649069828534
- lambdav =  -0.12557994308223847
- tf =  0.8484653913180362

- lambdax =  -0.13442521076425082
- lambday =  -0.07238005895813605
- lambdav =  -0.13604120669671094
- tf =  0.810049475065594

- lambdax =  -0.14388399371303937
- lambday =  -0.06650636516852312
- lambdav =  -0.13303628108069374
- tf =  0.8182643010087328

- lambdax =  -0.1440291795329469
- lambday =  -0.06617609412283805
- lambdav =  -0.13300674453684666
- tf =  0.8161078937378686

- lambdax =  -0.14371283877892754
- lambday =  -0.06623453965659519
- lambdav =  -0.13281898099981082
- tf =  0.8166004937983459

- lambdax =  -0.14350088676249106
- lambday =  -0.06635030596649731
- lambdav =  -0.13273048436497292
- tf =  0.8167705030611013

- lambdax =  -0.14351385261500474
- lambday =  -0.0663582216098168
- lambdav =  -0.1327369110009303
- tf =  0.8167522537035999

- lambdax =  -0.143513893951255
- lambday =  -0.06636180144075435
- lambdav =  -0.13273699184607327
- tf =  0.816751878008503

- lambdax =  -0.14351390369838887
- lambday =  -0.06636275801478299
- lambdav =  -0.13273700301234218
- tf =  0.8167518416113585

- lambdax =  -0.1435139024551872
- lambday =  -0.06636284216272598
- lambdav =  -0.13273700275478856
- tf =  0.8167518426713121

- lambdax =  -0.14351390244855963
- lambday =  -0.0663628554250985
- lambdav =  -0.13273700282564077
- tf =  0.8167518424567212
```


Brachistocrone Curve
y
0.00
−0.25
−0.50
−0.75
−1.00
−1.25
−1.50
−1.75
−2.00
x
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00



In [ ]: