# Modeling Traffic with Metering Control

Broderik Craig, Sheridan Harding, Joe Housley,
Adam Skousen, Jonah Sundrud

April 19, 2023

**Abstract**

In our project, we model the flow of traffic through a mile long strip of highway. We assume that we have control over the meter that lets new cars onto the highway and we seek to find the optimal way to let cars in to not cause traffic jams but still let on enough people to so that everyone gets where they need to go in a timely manner.

## 1 Background

This is an important problem in the planning of highways as the city needs to decide when the meter will be on and when it will be off. Normally during the hours where there is little traffic, an on-ramp has no meter and during rush hour it is metered. This allows the cars that are on the road to travel at full speed without the road becoming congested. If too many cars are sent in at once, the road is congested and it takes longer for everyone to get where they need to go. If too few cars are sent, it also takes longer for people to get where they need to go because they are spending unnecessary time waiting at the meter when they could be on the road already. Thus, this problem is important because it seeks to find that balance- send enough to keep things moving but not too much so that there is congestion. According to INRIX, the average person in Chicago spends 155 hours per year in traffic [1]. If we could find ways to minimize the time we spend in transit, the world would be far better off.

# 2 Mathematical Representation

## 2.1 First Attempt

Our first attempt at a cost function took into account bikes, buses, and cars, and is given by

$$J[u] = \int_0^{t_f} \alpha A(t)^2 + \beta B(t)^2 + \gamma C(t)^2 \, dt \tag{1}$$

where

$A(t)$ is number of cars (automobiles) on the road at time t

$B(t)$ is number of buses on the road at time t

$C(t)$ is number of cyclists on the road at time t

$\alpha$ is density of car passengers

$\beta$ is density of bike passengers

$\gamma$ is density of bus passengers

This measures how much space is being used by the vehicle and how many passengers the vehicle uses. We are attempting to minimize the number of vehicles while maximizing the number of passengers.

The evolution equation modeling the change in the number of cars within the stretch of highway we are modeling is

$$x' = E(t) - \hat{s} \odot [\frac{F - x}{F}] + u \tag{2}$$

where

$x = [A, B, C]$

$E(t)$ is existing highway traffic

$\hat{s}$ is the functional speed limit of each vehicle

$F$ is the maximum capacity of the highway

$u$ is the control, which is how many of each vehicle we let on the highway

Finally the demand for the on-ramp is modeled by

$$D' = -u\dot{k} + f(t) \tag{3}$$

where

$u$ is our control

$k$ is the number of passengers each vehicle can take

$f(t)$ is the number of passengers waiting at time $t$.

We choose initial and endpoint conditions for the state and demand as follows:

$A(0) = 100$

$B(0) = 5$

$C(0) = 20$

$D(0) = [1000, 3000, 100]$

$D(t_f) = [0, 0, 0]$

### 2.2 PDE Adaptation

As a second attempt, we decided to only model cars entering the highway and instead to find out how to maximize traffic flow while taking into account how too much traffic slows down traffic. Our control here is only how many cars we let on at each time step.
We adapted a traffic flow PDE found in the volume 4 textbook to our situation. The original PDE is $u_t + u_x(1 - u^2)t - 2u^2u_x = t$. We modify this by assuming $u_x$ to be 0 for all $x$, meaning that we don't care about the density at each point on the highway, just the density on the entire stretch of highway. This yields the ODE $u_t = r$ where $u$ is the density and $r$ is the rate of entry onto the highway. Our cost functional for this version of the problem is:

$$J[r] = \int_0^{t_f} r^2 + (1 - u)^2 dt \tag{4}$$

$$\text{subject to} \quad u'(t) = r(t) \tag{5}$$

### 2.3 Model including waiting ramp

For our final and most interesting attempt, we left the PDE adaptation behind and instead increased our state space to include another state that is the number of cars waiting on the ramp, ready to be let onto the highway. The optimization problem here is as follows:

$$J[r] = \int_0^{t_f} x^2 - u^2/8 + y^2 dt \tag{6}$$

$$\text{subject to} \quad x' = (|x - 100| + 100)/25 + u + E(t) \tag{7}$$

$$y' = f(t) - u \tag{8}$$

$$x(0) = 20 \tag{9}$$

$$y(0) = 0 \tag{10}$$

where

$x$ is the number of cars on the road

$u$ is the number of cars let on from the ramp

$y$ is the number of cars waiting on the ramp

$E(t)$ is existing highway traffic, taken to be a constant 10

$f(t)$ is the number of cars arriving at the ramp, taken to be a constant 100

The $(|x - 100| + 100)/25$ represents the flow of traffic and how if there are more cars on the road then less will be leaving proportionally. The $u$ adds to $x$ as does the $E(t)$ because those are the cars flowing in. The $y$ is affect by cars flowing in $f(t)$ and cars let off onto the highway $u$. We seek to minimize the number of cars on the road $x$ and cars on the ramp $y$ while maximizing how many cars we let onto the highway $u$.

# 3 Solution

## 3.1 First Attempt

Here we will derive the co-state and optimal control using Pontraygin's maximum principle.

$$H = p \cdot f - L = p_1(e_1 - s_1 \cdot A + u_1) + p_2(e_2 - s_2 \cdot B + u_2) + p_3(e_3 - s_3 \cdot C + u_3) + p_4(f_1 - u_1 k_1) + p_5(f_2 - u_2 k_2) + p_6(f_3 - u_3 k_3) + \alpha A^2 + \beta B^2 + \gamma C^2 + \frac{\alpha}{k_1} D_1^2 + \frac{\beta}{k_2} D_2^2 + \frac{\gamma}{k_3} D_3^2 \quad (11)$$

Then Pontraygin's Maximum Principle states that the optimal control maximizes the Hamiltonian.

$$\frac{DH}{Du_1} = p_1 - p_4 k_1$$
$$\frac{DH}{Du_2} = p_2 - p_5 k_2$$
$$\frac{DH}{Du_3} = p_3 - p_6 k_3$$

Which yields the following optimal controls. Note that the solution is a bang-bang.

$$\tilde{u_1}(t) = \begin{cases} U_{max} & \text{if } p_1 > p_4 k_1 \\ U_{min} & \text{if } p_1 < p_4 k_1 \end{cases}$$
$$\tilde{u_2}(t) = \begin{cases} U_{max} & \text{if } p_2 > p_5 k_2 \\ U_{min} & \text{if } p_2 < p_5 k_2 \end{cases}$$
$$\tilde{u_3}(t) = \begin{cases} U_{max} & \text{if } p_3 > p_6 k_3 \\ U_{min} & \text{if } p_3 < p_6 k_3 \end{cases}$$

We also find the co-state equations from the Hamiltonian

$$\begin{aligned}
p_1' &= \frac{-p_1 s_1}{F} - \alpha \\
p_2' &= \frac{-p_2 s_2}{F} - \beta \\
p_3' &= \frac{-p_3 s_4}{F} - \gamma \\
p_4' &= \frac{\alpha}{k_1} - p_4 \\
p_5' &= \frac{\beta}{k_2} - p_5 \\
p_6' &= \frac{\gamma}{k_3} - p_6
\end{aligned}$$

With the following endpoint conditions

$$\begin{aligned}
p_1(t_f) &= 0 \\
p_2(t_f) &= 0 \\
p_3(t_f) &= 0
\end{aligned}$$

### 3.2 PDE Adaptation

Here we use a simplified version of the PDE discussed in Section 2.2. Notice that here, we use different notation. For example, $u$ is our state density function and is one-dimensional. $r$ represents the rate at which cars enter the highway and is our control. The problem is as follows:

$$\begin{aligned}
J[r] = \int_0^{t_f} r^2 - (1-u)^2 dt \\
\text{subject to } u' = r
\end{aligned}$$

This has a Hamiltonian

$$H = pr + r^2 + (1-u)^2, \tag{12}$$

as we are trying to maximize $J[r]$. We solve the problem as follows:

$$\begin{aligned}
p' = -\frac{DH}{Du} &= 2(1-u) = 2 - 2u \\
\frac{DH}{Dr} &= 0 = p + 2r \\
p &= -2r \\
u' &= -\frac{1}{2}p
\end{aligned}$$

With Wolfram Alpha, we found the solution

$$u = \frac{c_1}{2}(e^t + e^{-t}) - \frac{c_2}{4}(e^t - e^{-t}) + 1 \tag{13}$$

$$p = c_1(e^t - e^{-t}) + \frac{c_2}{2}(e^t + e^{-t}) \tag{14}$$

Using $u(0) = .8$ and $p(0) = 0$, we see that $c_1 = .8$ and $c_2 = 0$, so

$$\begin{aligned} u &= .4(e^t + e^{-t}) + 1 \\ p &= -.8(e^t - e^{-t}) \\ r &= .4(e^t - e^{-t}). \end{aligned}$$

### 3.3 Model including waiting ramp

We begin by finding the Hamiltonian

$$H = p_1 x' + p_2 y' - y^2 - x^2 + u^2/8, \tag{15}$$

so by Pontraygin's maximum principle:

$$p_1' = -\frac{DH}{Dx} = p_1(x - 100)/|x - 100|/25 + 2x \tag{16}$$

$$p_1(t_f) = 0 \tag{17}$$

$$p_2' = -\frac{DH}{Dx} = 2y \tag{18}$$

$$p_2(t_f) = 0 \tag{19}$$

$$\frac{DH}{Du} = 0 = p_1 - p_2 + u/4 \tag{20}$$

$$u = 4(p_2 - p_1) \tag{21}$$

$$\tag{22}$$

## 4 Interpretation

### 4.1 First Attempt

The solution to the first attempt was sensitive to the initial conditions, but follows a consistent form. The first graph shows the number of vehicles on the highway at each point in time, the second graph shows the number of vehicles waiting on the on-ramp to enter the highway, and the third

graph shows the control sequence of the meter. In this we assumed that the maximum control was 70 vehicles for each type of vehicle.

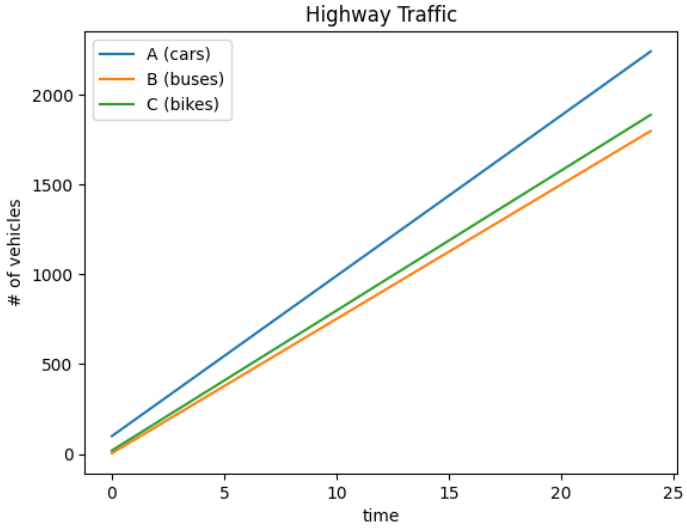



Figure 1: Vehicles on the highway over time

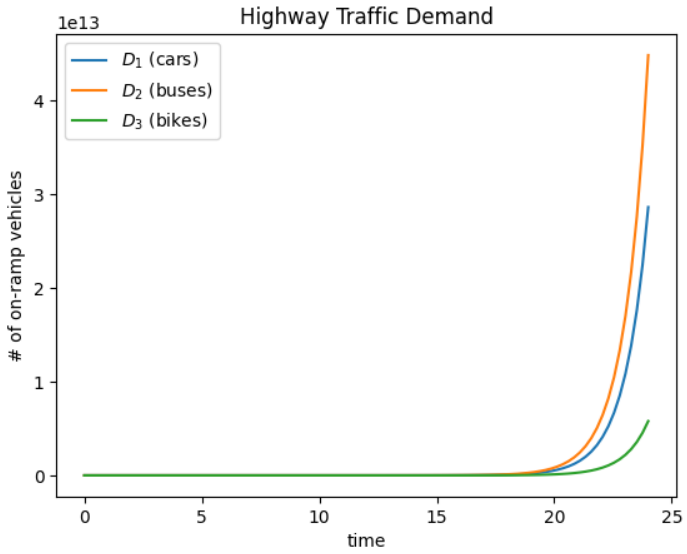



Figure 2: Highway demand over time

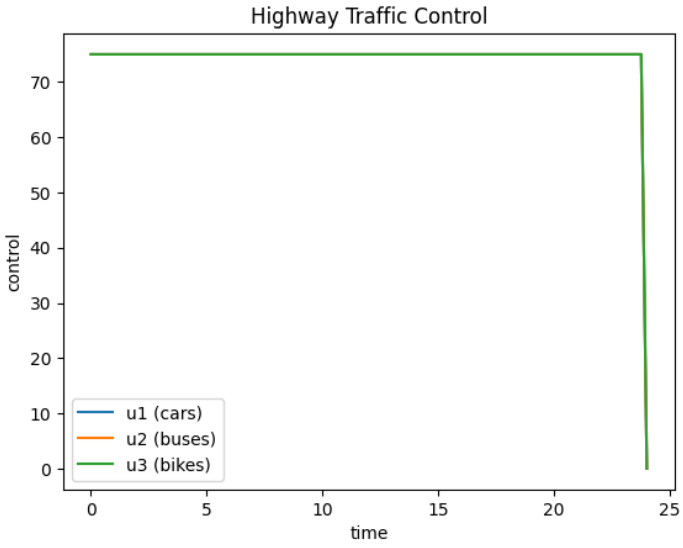



Figure 3: Control of the metered on-ramp over time

For the first iteration, we found a solution that yielded trivial but plausible results. The solution was to allow as many vehicles onto the highway at all times. The reason that the graph of the control goes to 0 at the end of the time interval is because in the derivation of the co-state equations found from Pontraygin's Maximum Principle, we introduced an endpoint condition on the co-state which also forces the control to 0 at the endpoint. We were dissatisfied with this solution because it does not actually implement the control we added. We attempted to modify the cost function to penalize traffic density but were unsuccessful in producing a viable model.

### 4.2 PDE Adaptation

The solution to the iteration that was adapted from the traffic flow PDE is shown in the following graph.

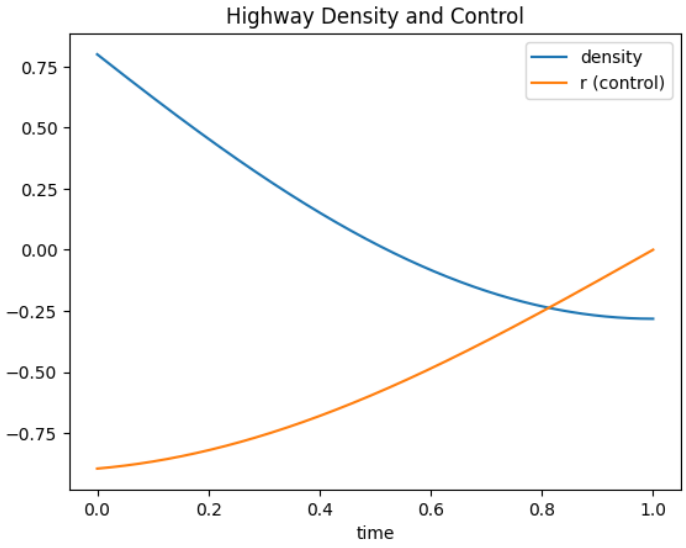



Figure 4: Solution of the cost functional $J[r] = \int_0^{t_f} r^2 + (1-u)^2 dt$ subject to the state equation $u_t = r$.

In this iteration, we reduced the complexity to only consider cars. The solution of this simplified ODE looked more reasonable in its shape because traffic declines over time while the number of cars we allow onto the highway increases. One major concern is that the actual values of the control were negative, which doesn't make sense in the context of our project.

### 4.3 Model including waiting ramp

When we plugged this setup into solve_bvp using $E(t) = 10$ and $f(t) = 100$ with initial conditions $x(0) = 20$ and $y(0) = 0$ we got figure 5. We find this to be the most interesting of the solutions because it shows the oscillating nature of the problem. One notes that the solution is nearly bang-bang in that is goes through periods of turning the meter completely off and then allows many cars in at once. It seems to be doing this according to a rule of letting the highway run for a while with whoever is on there, letting them keep a good speed. Then when the ramp gets too full compared to how empty the highway is, another batch of waiting cars is allowed on. This decreases the number of cars on the ramp and increases the number on the highway and the cycle repeats. During the first section the number of cars on the highway dips into the negative, which is obviously nonsensical. However, this behavior only happens on the first bit and the solution settles

into a steadily increasing oscillatory pattern, which leads us to believe it is still valid in the long term. If more time were available we could incorporate inequality constraints to bound the solution and likely get improved results.

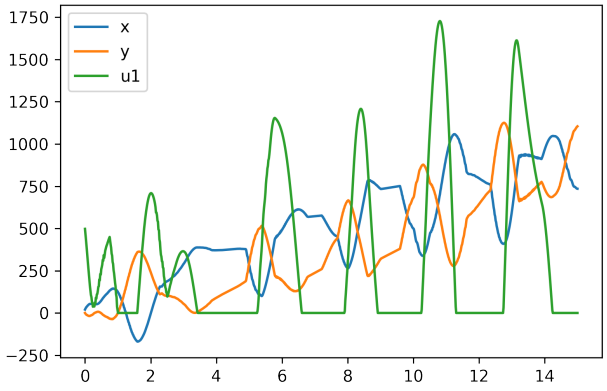



Figure 5: Solution of the cost functional $J[r] = \int_0^{t_f} x^2 - u^2/8 + y^2 dt$ subject to the state equations $x' = (|x - 100| + 100)/25 + u + E(t)$ and $y' = f(t) - u$.

## References

[1] INRIX. Inrix 2022 traffic scorecard, 2022.

We give Dr. Barker, Dr. Whitehead, and other instructors at BYU teaching ACME classes permission to share our project as an example of a good (or bad) project in future classes they teach.

# 5 Appendix 1: Code

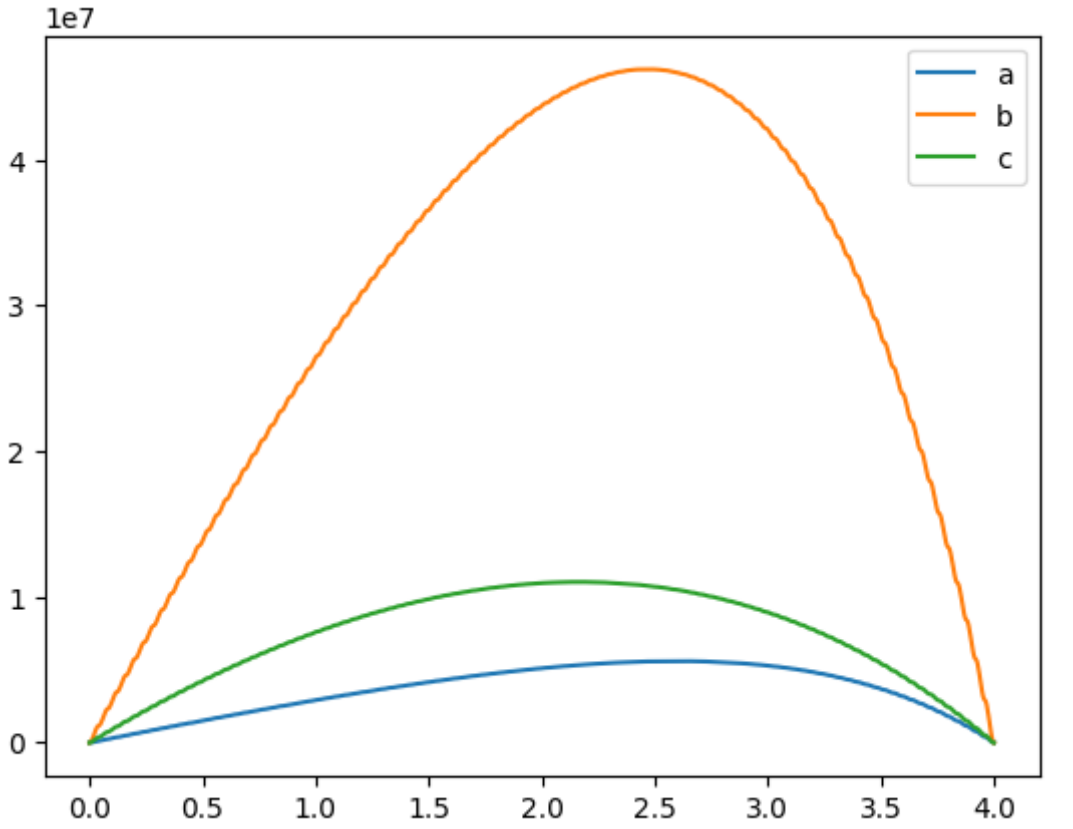
In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import solve_bvp
```

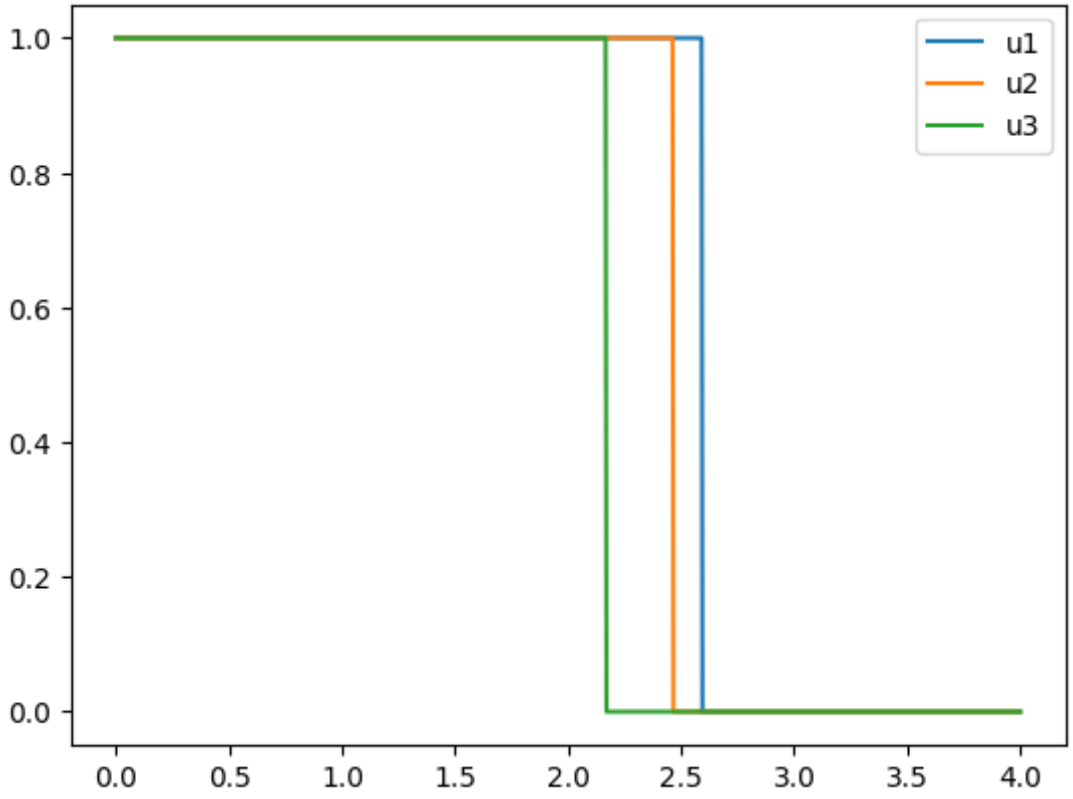
$a =$ number of automobiles
$b =$ number of buses
$c =$ number of cyclists
$\alpha =$ passengers / car length
$\beta =$ passengers / bus length
$\gamma =$ passengers / bicycle length
$d_1 =$ demand for cars to get on the highway
$d_2 =$ demand for buses to get on the highway
$d_3 =$ demand for bicycles to get on the highway
$s_1 =$ speed of cars
$s_2 =$ speed of buses
$s_3 =$ speed of bicycles

maximize $J[u] = \int_0^{t_f} (\delta \cdot x - u \cdot k + f(t))\, dt$
subject to $x' = (1 - s)x + (d - u)$
where $x = (a, b, c)^T$
and $\delta = (\alpha, \beta, \gamma)^T$
and $u = (u_1, u_2, u_3)^T$
and $s = (s_1, s_2, s_3)^T$
and $d = (d_1, d_2, d_3)^T$

In [39]:
```
# x' = c - s*(F-x)/F + u
# a' = c1 - s1*(F-a)/F + u1
# b' = c2 - s2*(F-b)/F + u2
# c' = c3 - s3*(F-c)/F + u3
# D' = -u*k + f(t)
# d1' = -u1*k1 + f1(t)
# d2' = -u2*k2 + f2(t)
# d3' = -u3*k3 + f3(t)
# p' = s*p - delta
# p1' = (s1)*p1 - alpha
# p2' = (s2)*p2 - beta
# p3' = (s3)*p3 - gamma
k1, k2, k3 = 1.5, 30, 1.
s1, s2, s3 = 1., .75, .25
c1, c2, c3 = 15., 1/3, 3.
F = 10_000
f = lambda t: 1000
alpha, beta, gamma = .1, 30./39, 1./6
U_max, U_min = 1, 0
def u_hat(p):
    u = np.zeros_like(p)
    for i in range(p.size):
        if p[i] > 0:
            u[i] = U_max
        elif p[i] < 0:
```

```
            u[i] = U_min
    return u
def ode_fun(t,y):
    a,b,c,p1,p2,p3 = y
    return np.vstack((c1 - s1*(F-a)/F + u_hat(p1),
                      c2 - s2*(F-b)/F + u_hat(p2),
                      c3 - s3*(F-c)/F + u_hat(p3),
                      s1*p1 - alpha,
                      s2*p2 - beta,
                      s3*p3 - gamma))

def bc(ya,yb):
    return np.array([ya[0], yb[0]-10, ya[1], yb[1]-5, ya[2], yb[2]-20])

# initial guess
t = np.linspace(0, 4, 100)
y = np.zeros((6, t.size))
sol = solve_bvp(ode_fun, bc, t, y)

# plot solution
t_plot = np.linspace(0, 4, 1000)
y_plot = sol.sol(t_plot)
plt.plot(t_plot, y_plot[0], label='a')
plt.plot(t_plot, y_plot[1], label='b')
plt.plot(t_plot, y_plot[2], label='c')
plt.legend()
plt.show()

# plot control
u1 = u_hat(y_plot[3])
u2 = u_hat(y_plot[4])
u3 = u_hat(y_plot[5])
plt.plot(t_plot, u1, label='u1')
plt.plot(t_plot, u2, label='u2')
plt.plot(t_plot, u3, label='u3')
plt.legend()
plt.show()
```



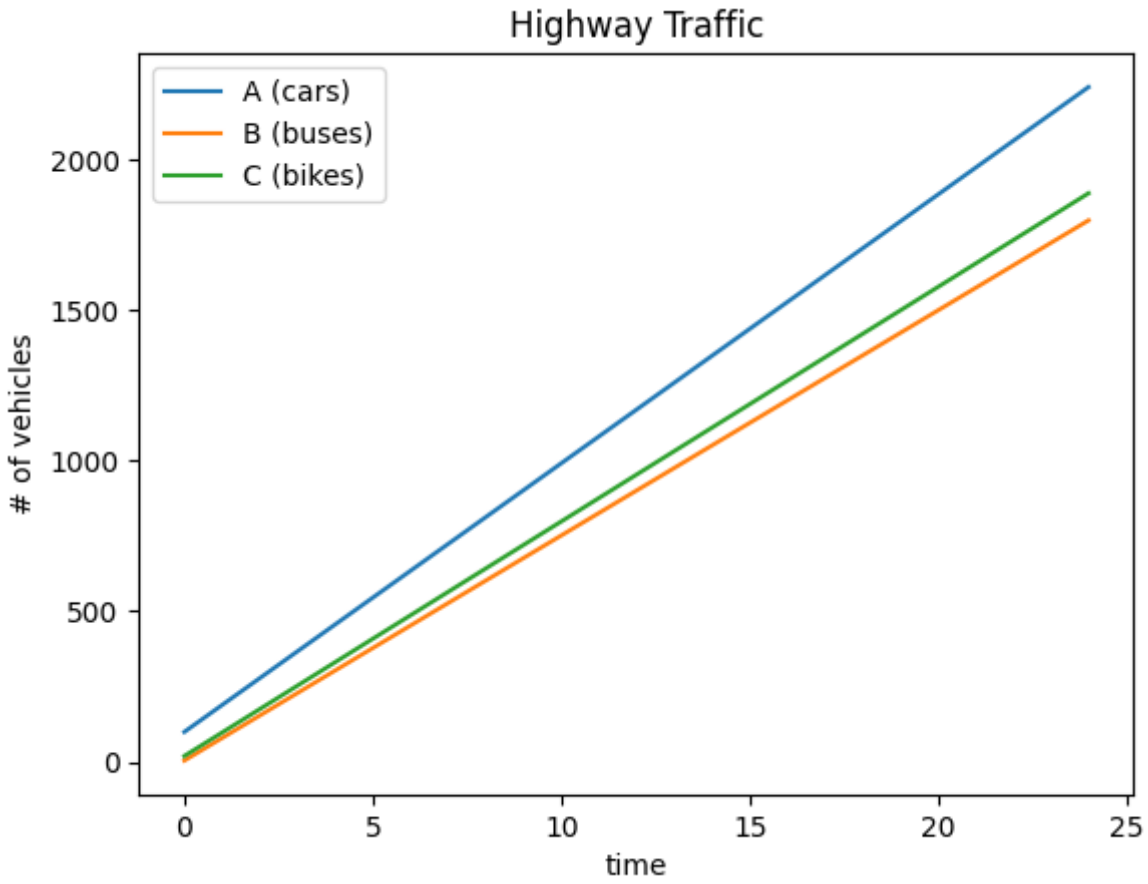



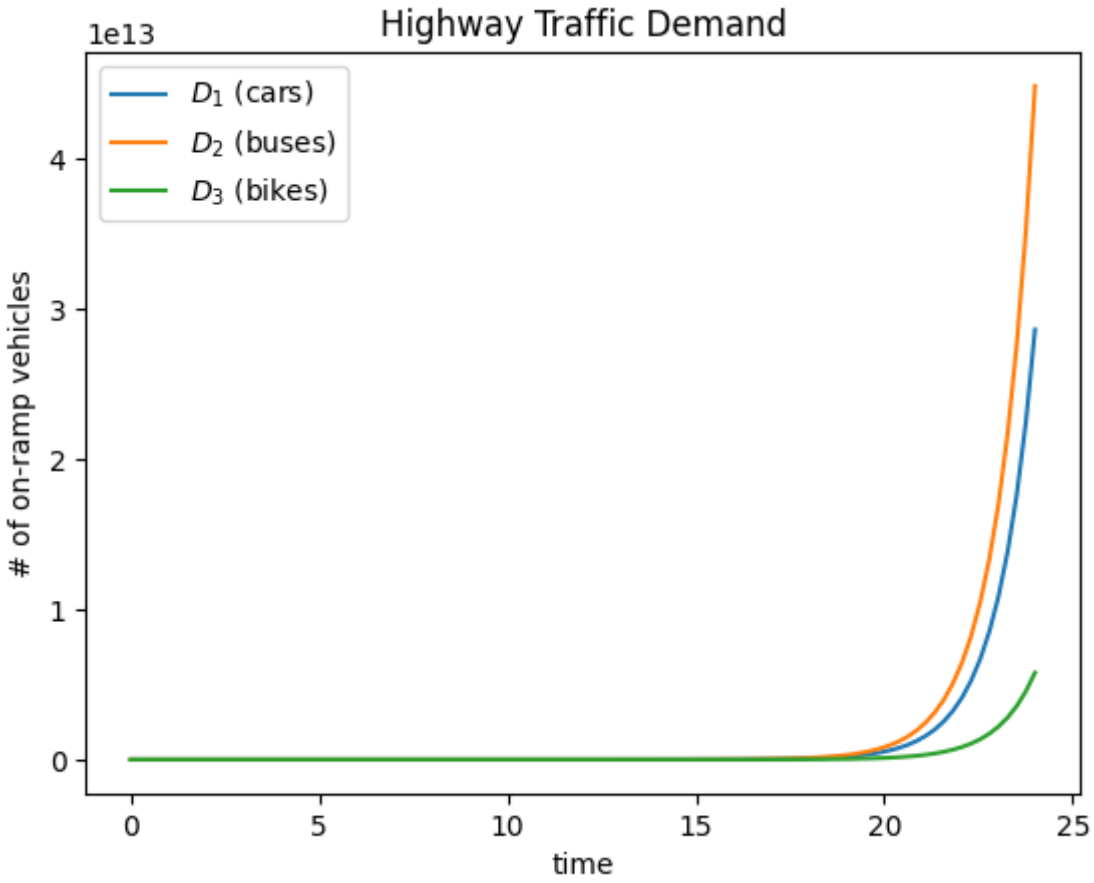

In [7]:
```
# x' = c - s*(F-x)/F + u
# a' = c1 - s1*(F-a)/F + u1
# b' = c2 - s2*(F-b)/F + u2
# c' = c3 - s3*(F-c)/F + u3
# y' = y - u*k + f(t)
# y1' = y1 - u1*k1 + f1(t)
# y2' = y2 - u2*k2 + f2(t)
```

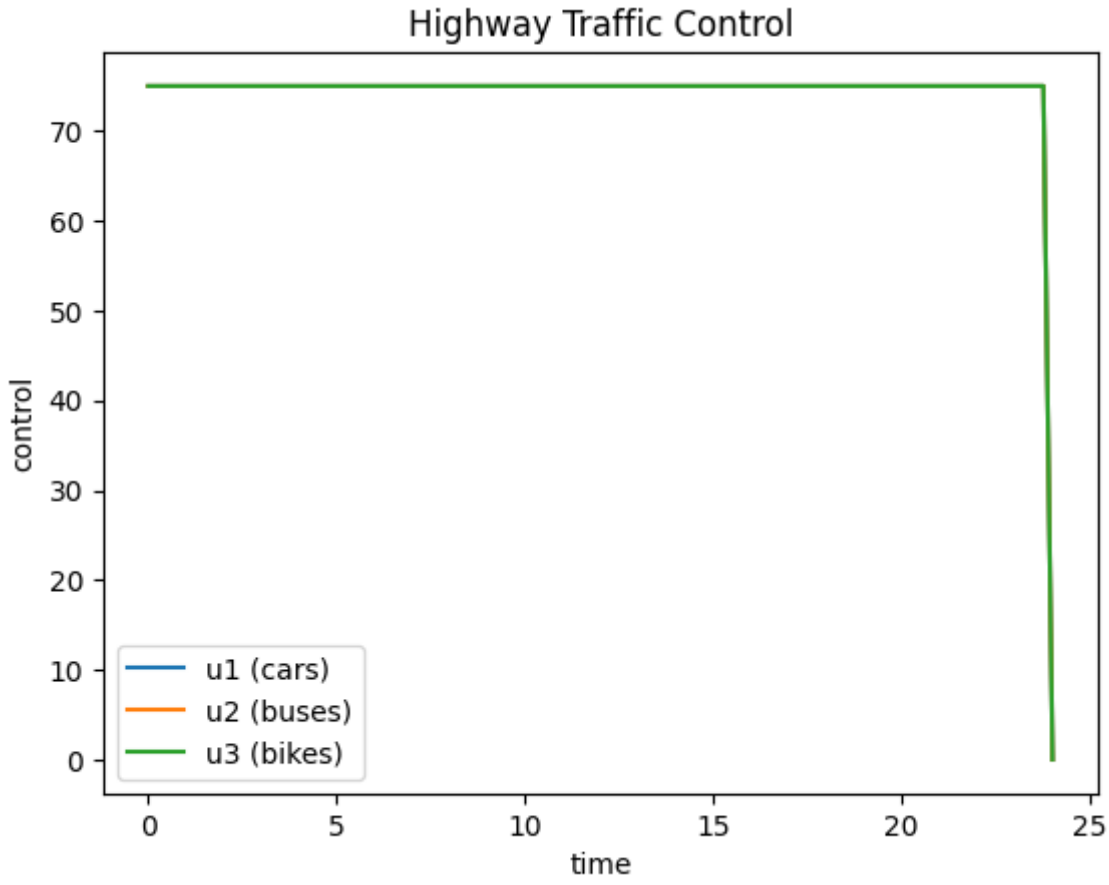
```
# y3' = y3 - u3*k3 + f3(t)
# p' = s*p - delta
# p1' = -(s1)*p1/F - alpha
# p2' = -(s2)*p2/F - beta
# p3' = -(s3)*p3/F - gamma
# p4' = alpha/k1 - p4
# p5' = beta/k2 - p5
# p6' = gamma/k3 - p6
k1, k2, k3 = 1.5, 20, 1.
s1, s2, s3 = 1., .75, .25
c1, c2, c3 = 15., 1/3, 3.
F = 10000
f1 = lambda t: 100*np.cos(np.pi*t/12) + 100
f2 = lambda t: 100*np.cos(np.pi*t/12) + 100
f3 = lambda t: 100*np.cos(np.pi*t/12) + 100

alpha, beta, gamma = .1, 30./39, 1./6
U_max, U_min = 75, 0
def u1_hat(p1,p4):
    u1 = np.zeros_like(p1)
    for i in range(p1.size):
        if p1[i] > p4[i] * k1:
            u1[i] = U_max
        elif p1[i] < p4[i] * k1:
            u1[i] = U_min
    return u1
def u2_hat(p2,p5):
    u2 = np.zeros_like(p2)
    for i in range(p2.size):
        if p2[i] > p5[i] * k2:
            u2[i] = U_max
        elif p2[i] < p5[i] * k2:
            u2[i] = U_min
    return u2
def u3_hat(p3,p6):
    u3 = np.zeros_like(p3)
    for i in range(p3.size):
        if p3[i] > p6[i] * k3:
            u3[i] = U_max
        elif p3[i] < p6[i] * k3:
            u3[i] = U_min
    return u3
def ode_fun(t,y):
    a,b,c,y1,y2,y3,p1,p2,p3,p4,p5,p6 = y
    u1 = u1_hat(p1,p4)
    u2 = u2_hat(p2,p5)
    u3 = u3_hat(p3,p6)
    return np.vstack((c1 - s1*(F-a)/F + u1,
                      c2 - s2*(F-b)/F + u2,
                      c3 - s3*(F-c)/F + u3,
                      y1 + f1(t) - u1*k1,
                      y2 + f2(t) - u2*k2,
                      y3 + f3(t) - u3*k3,
                      -s1*p1/F - alpha,
                      -s2*p2/F - beta,
                      -s3*p3/F - gamma,
                      alpha/k1 - p4,
                      beta/k2 - p5,
                      gamma/k3 - p6))
```

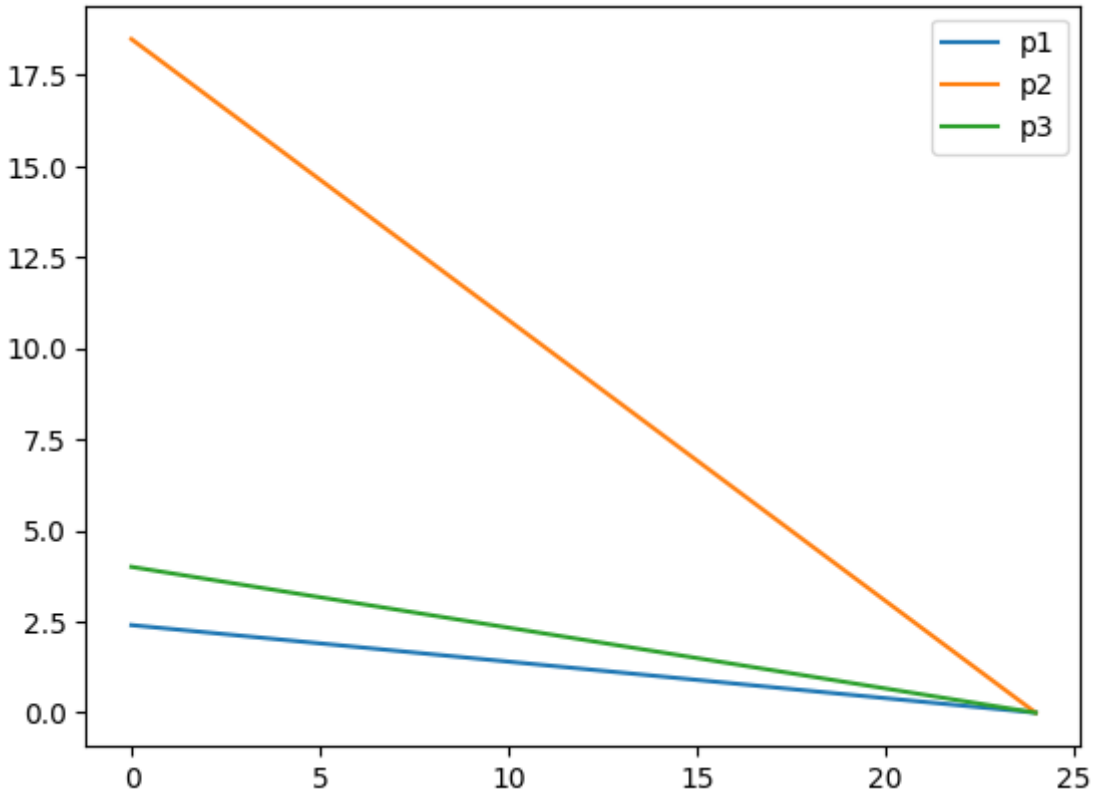
```
def bc(ya,yb):
    return np.array([ya[0]-100,
                     ya[1]-5,
                     ya[2]-20,
                     ya[3]-1000,
                     ya[4]-3000,
                     ya[5]-100,
                     yb[9],
                     yb[10],
                     yb[11],
                     yb[6],
                     yb[7],
                     yb[8]])


# initial guess
t = np.linspace(0, 24, 100)
y = np.zeros((12, t.size))
sol = solve_bvp(ode_fun, bc, t, y)

# plot solution
t_plot = np.linspace(0, 24, 100)
y_plot = sol.sol(t_plot)
plt.plot(t_plot, y_plot[0], label='A (cars)')
plt.plot(t_plot, y_plot[1], label='B (buses)')
plt.plot(t_plot, y_plot[2], label='C (bikes)')
plt.xlabel('time')
plt.ylabel('# of vehicles')
plt.title('Highway Traffic')
plt.legend()
plt.show()

# plot demand
plt.plot(t_plot, y_plot[3], label=r'$D_1$ (cars)')
plt.plot(t_plot, y_plot[4], label=r'$D_2$ (buses)')
plt.plot(t_plot, y_plot[5], label=r'$D_3$ (bikes)')
plt.xlabel('time')
plt.ylabel('# of on-ramp vehicles')
plt.title('Highway Traffic Demand')
plt.legend()
plt.show()

# plot control
u1 = u1_hat(y_plot[6],y_plot[9])
u2 = u2_hat(y_plot[7],y_plot[10])
u3 = u3_hat(y_plot[8],y_plot[11])
plt.plot(t_plot, u1, label='u1 (cars)')
plt.plot(t_plot, u2, label='u2 (buses)')
plt.plot(t_plot, u3, label='u3 (bikes)')
plt.xlabel('time')
plt.ylabel('control')
plt.title('Highway Traffic Control')
plt.legend()
plt.show()

# plot p
plt.plot(t_plot, y_plot[6], label='p1')
plt.plot(t_plot, y_plot[7], label='p2')
plt.plot(t_plot, y_plot[8], label='p3')
```

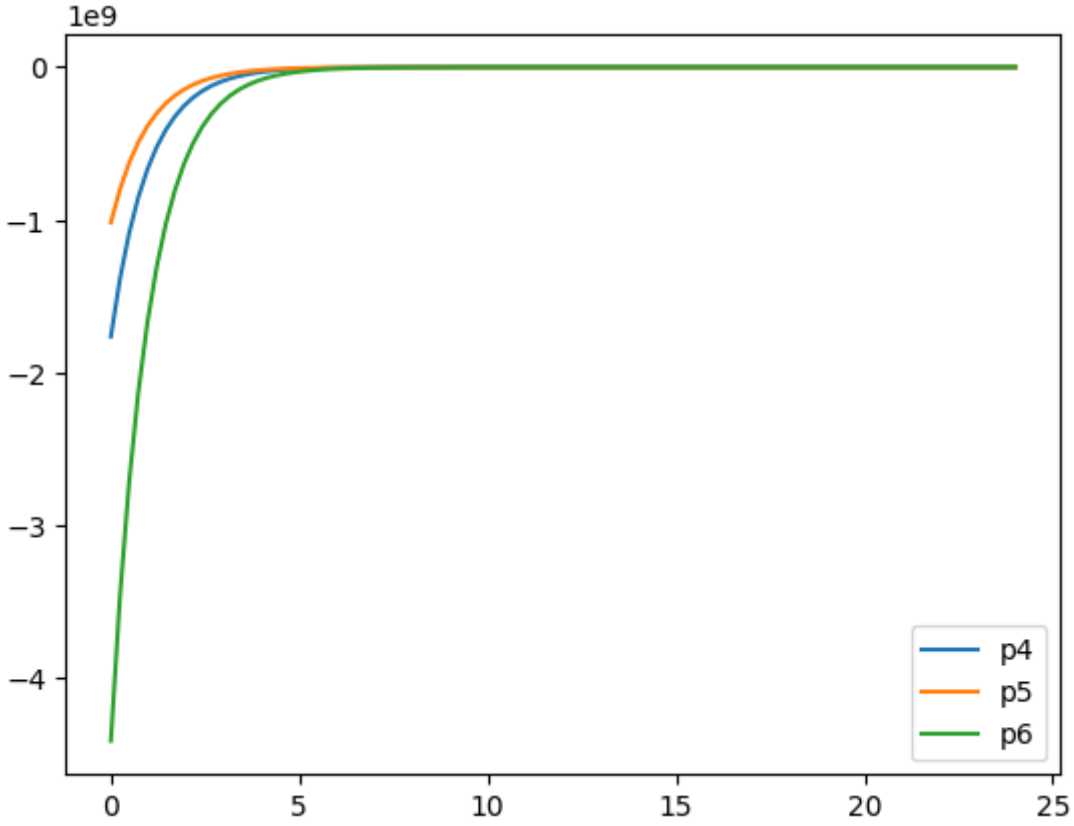
```
plt.legend()
plt.show()
plt.plot(t_plot, y_plot[9], label='p4')
plt.plot(t_plot, y_plot[10], label='p5')
plt.plot(t_plot, y_plot[11], label='p6')
plt.legend()
plt.show()
```




Highway Traffic Demand
1e13
$D_1$ (cars)
$D_2$ (buses)
$D_3$ (bikes)
# of on-ramp vehicles
time
0
1
2
3
4
0
5
10
15
20
25


Highway Traffic Control
u1 (cars)
u2 (buses)
u3 (bikes)
control
time
0
10
20
30
40
50
60
70
0
5
10
15
20
25

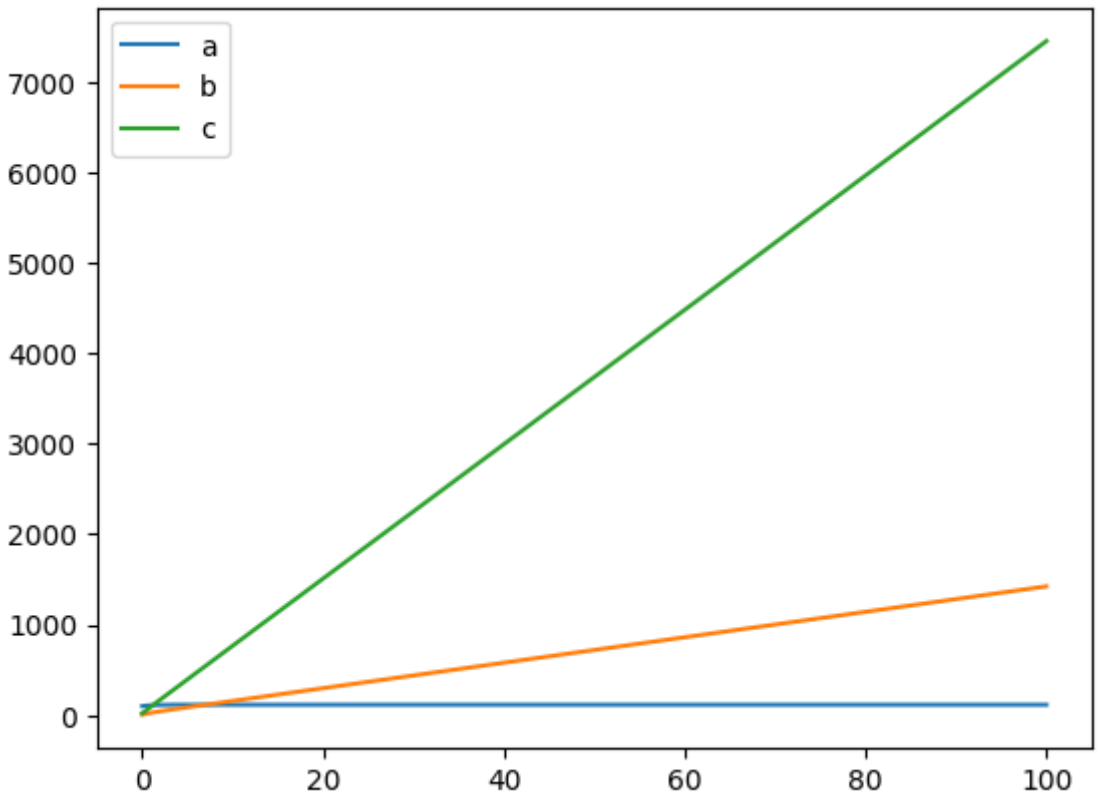


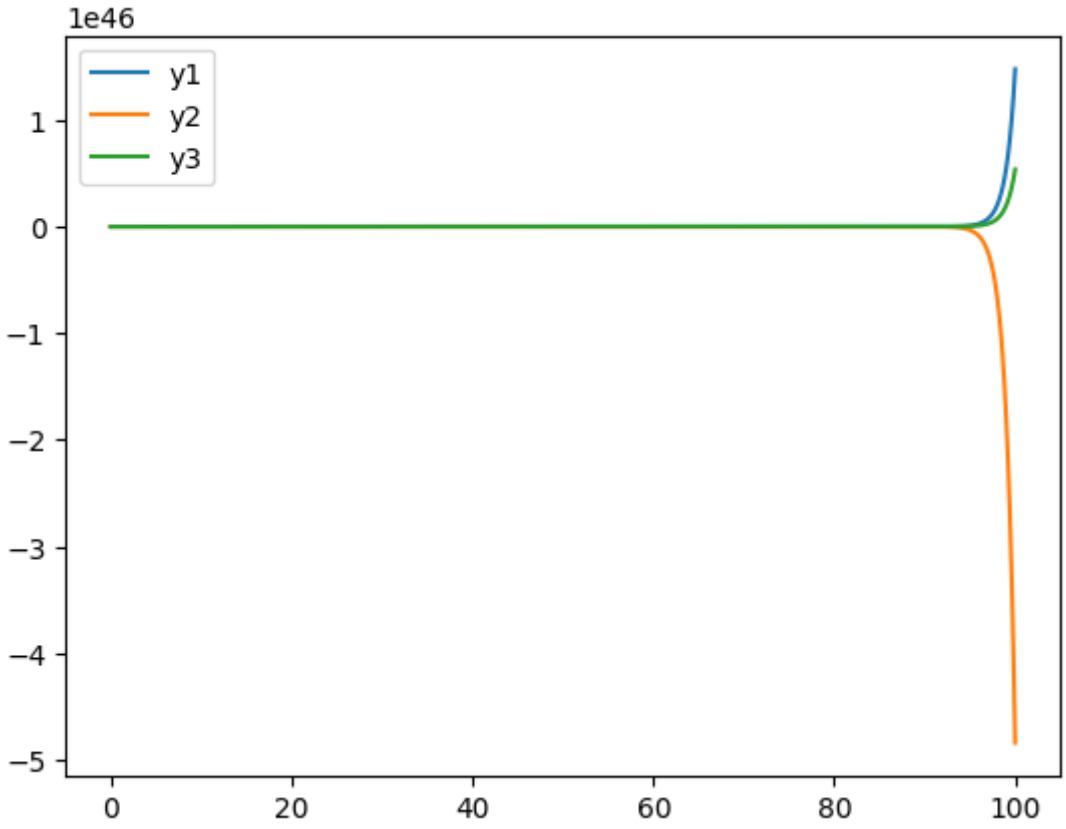



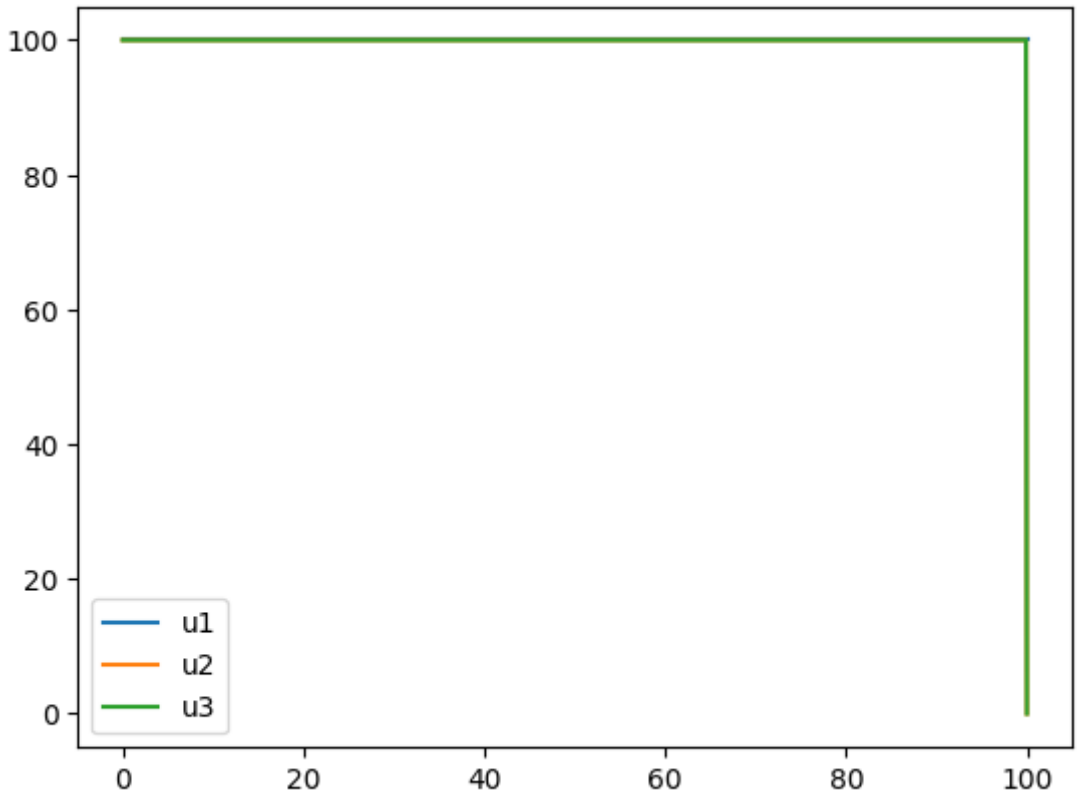

In [63]:
```
k1, k2, k3 = 1.5, 30, 1.
s1, s2, s3 = 1., .75, .25
c1, c2, c3 = 15., 1/3, 3.
F = 1000
f1 = lambda t: 100*np.cos(t/8) + 101
f2 = lambda t: 100*np.cos(t/8) + 101
f3 = lambda t: 100*np.cos(t/8) + 101
```

```
alpha, beta, gamma = .1, 30./39, 1./6
U_max, U_min = 100, 0
def u1_hat(p1,p4):
    u1 = np.zeros_like(p1)
    for i in range(p1.size):
        if p1[i] > p4[i] * k1:
            u1[i] = U_max
        elif p1[i] < p4[i] * k1:
            u1[i] = U_min
    return u1
def u2_hat(p2,p5):
    u2 = np.zeros_like(p2)
    for i in range(p2.size):
        if p2[i] > p5[i] * k2:
            u2[i] = U_max
        elif p2[i] < p5[i] * k2:
            u2[i] = U_min
    return u2
def u3_hat(p3,p6):
    u3 = np.zeros_like(p3)
    for i in range(p3.size):
        if p3[i] > p6[i] * k3:
            u3[i] = U_max
        elif p3[i] < p6[i] * k3:
            u3[i] = U_min
    return u3
def ode_fun(t,y):
    a,b,c,y1,y2,y3,p1,p2,p3,p4,p5,p6 = y
    u1 = u1_hat(p1,p4)
    u2 = u2_hat(p2,p5)
    u3 = u3_hat(p3,p6)
    return np.vstack((c1 - s1*a + u1,
                      c2 - s2*a + u2,
                      c3 - s3*a + u3,
                      y1 + f1(t) - u1*k1,
                      y2 + f2(t) - u2*k2,
                      y3 + f3(t) - u3*k3,
                      s1*p1 - 2*a*alpha,
                      s2*p2 - 2*b*beta,
                      s3*p3 - 2*c*gamma,
                      alpha/k1 - p4,
                      beta/k2 - p5,
                      gamma/k3 - p6))


def bc(ya,yb):
    return np.array([ya[0]-100,
                     ya[1]-5,
                     ya[2]-20,
                     ya[3]-500,
                     ya[4]-1000,
                     ya[5]-100,
                     yb[9],
                     yb[10],
                     yb[11],
                     yb[6],
                     yb[7],
                     yb[8]])
```

```
# initial guess
t = np.linspace(0, 100, 1000)
y = np.zeros((12, t.size))
sol = solve_bvp(ode_fun, bc, t, y)

# plot solution
t_plot = np.linspace(0, 100, 1000)
y_plot = sol.sol(t_plot)
plt.plot(t_plot, y_plot[0], label='a')
plt.plot(t_plot, y_plot[1], label='b')
plt.plot(t_plot, y_plot[2], label='c')
plt.legend()
plt.show()

# plot demand
plt.plot(t_plot, y_plot[3], label='y1')
plt.plot(t_plot, y_plot[4], label='y2')
plt.plot(t_plot, y_plot[5], label='y3')
plt.legend()
plt.show()

# plot control
u1 = u1_hat(y_plot[6],y_plot[9])
u2 = u2_hat(y_plot[7],y_plot[10])
u3 = u3_hat(y_plot[8],y_plot[11])
plt.plot(t_plot, u1, label='u1')
plt.plot(t_plot, u2, label='u2')
plt.plot(t_plot, u3, label='u3')
plt.legend()
plt.show()
```

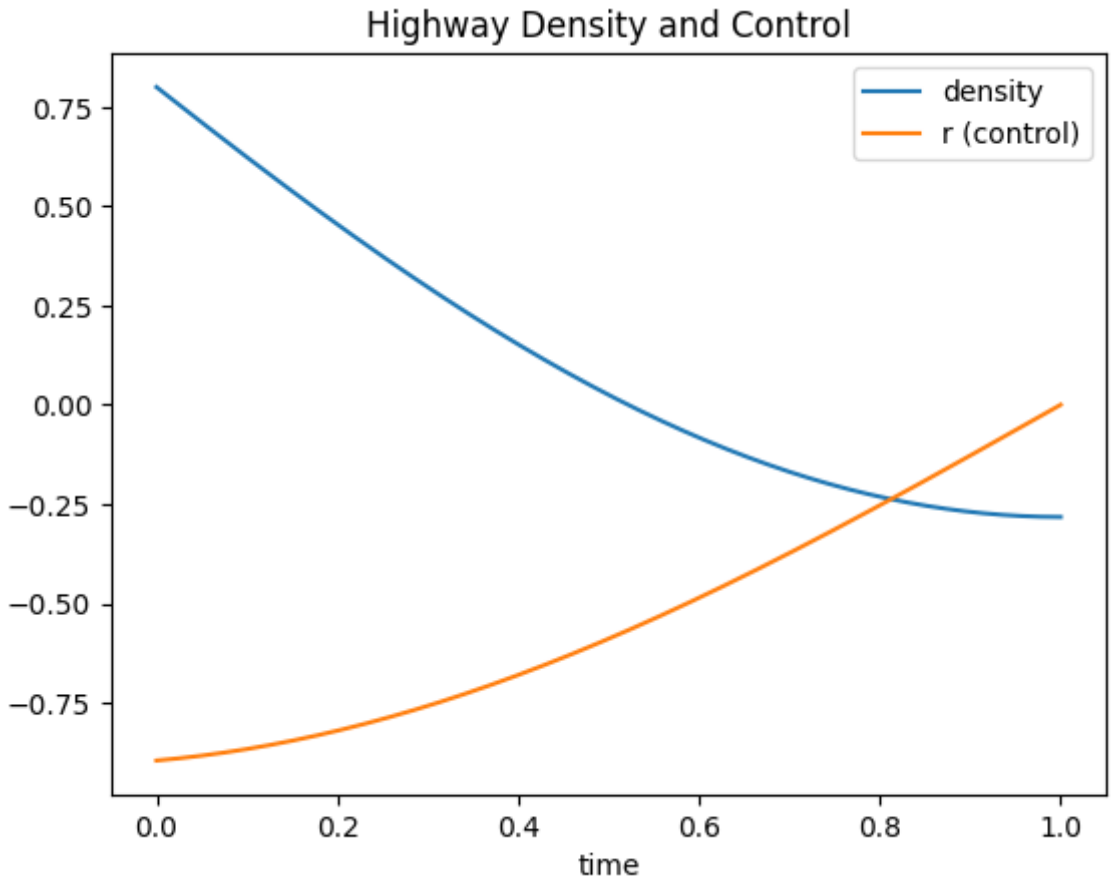




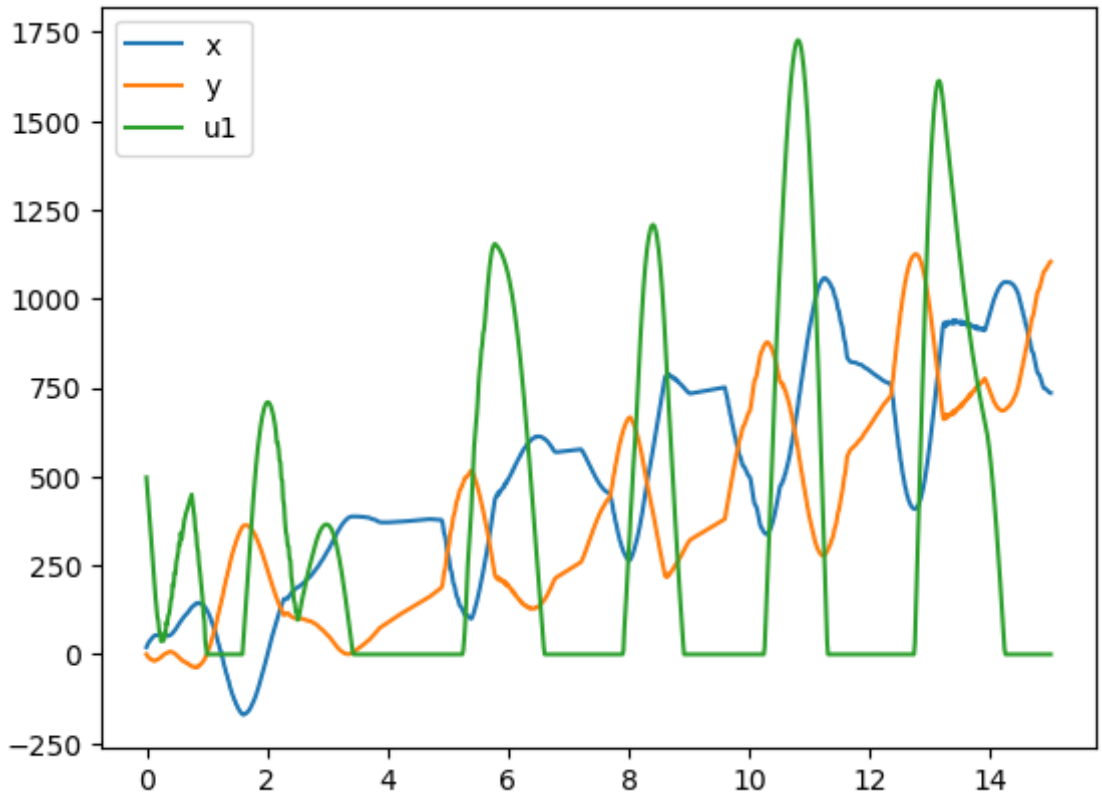

In [6]:
```
u = lambda p: p/2
xstart = .8
def ode(t,x):
    return np.vstack((x[1], 2-2*x[0]))

def bc(ya, yb):
    return np.array([ya[0]-xstart, yb[1]])
```

```
t = np.linspace(0,1)
y = np.zeros((2,t.size))
sol = solve_bvp(ode, bc, t, y)

t_plot = np.linspace(0,1,1000)
y_plot = sol.sol(t_plot)
plt.plot(t_plot, y_plot[0], label='density')

# plt.legend()
# plt.show()

uhat = u(y_plot[1])
plt.plot(t_plot, uhat, label='r (control)')
plt.xlabel('time')
plt.title('Highway Density and Control')
plt.legend()
plt.show()
```




In [2]:
```
E = 10
f = lambda t: 100
def uhat(p1,p2):
     return np.maximum((p2-p1)*4,0)
def ode(t, X):
    x = X[0]
    y = X[1]
    p1 = X[2]
    p2 = X[3]
    return np.vstack(((np.abs(x-100) - 100)/25 + uhat(p1,p2) + E,
                      f(t) - uhat(p1,p2),
                      p1*(x-100)/abs(x-100)/25 + x*2,
```

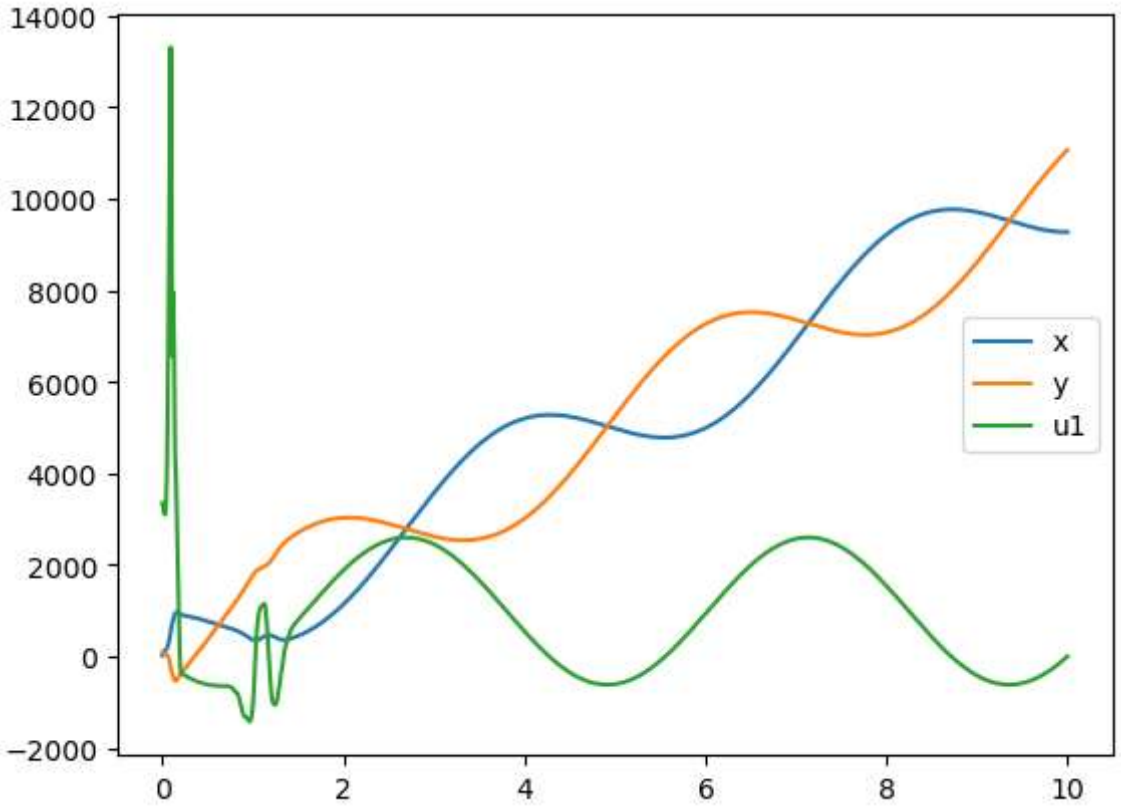
```
                                2*y))
def bc(ya,yb):
    return np.array([ya[0]-20, ya[1], yb[2], yb[3]])
t = np.linspace(0,15)
y = np.zeros((4,t.size))
sol = solve_bvp(ode, bc, t, y)
t_plot = np.linspace(0,15,1000)
y_plot = sol.sol(t_plot)
plt.plot(t_plot, y_plot[0], label='x')
plt.plot(t_plot, y_plot[1], label='y')
u1 = []
for i in range(len(y_plot[2])):
    u1.append(uhat(y_plot[2][i],y_plot[3][i]))
plt.plot(t_plot, u1, label='u1')
plt.legend()
plt.savefig('final_solution.png',dpi=300)
plt.show()
```




In [ ]:

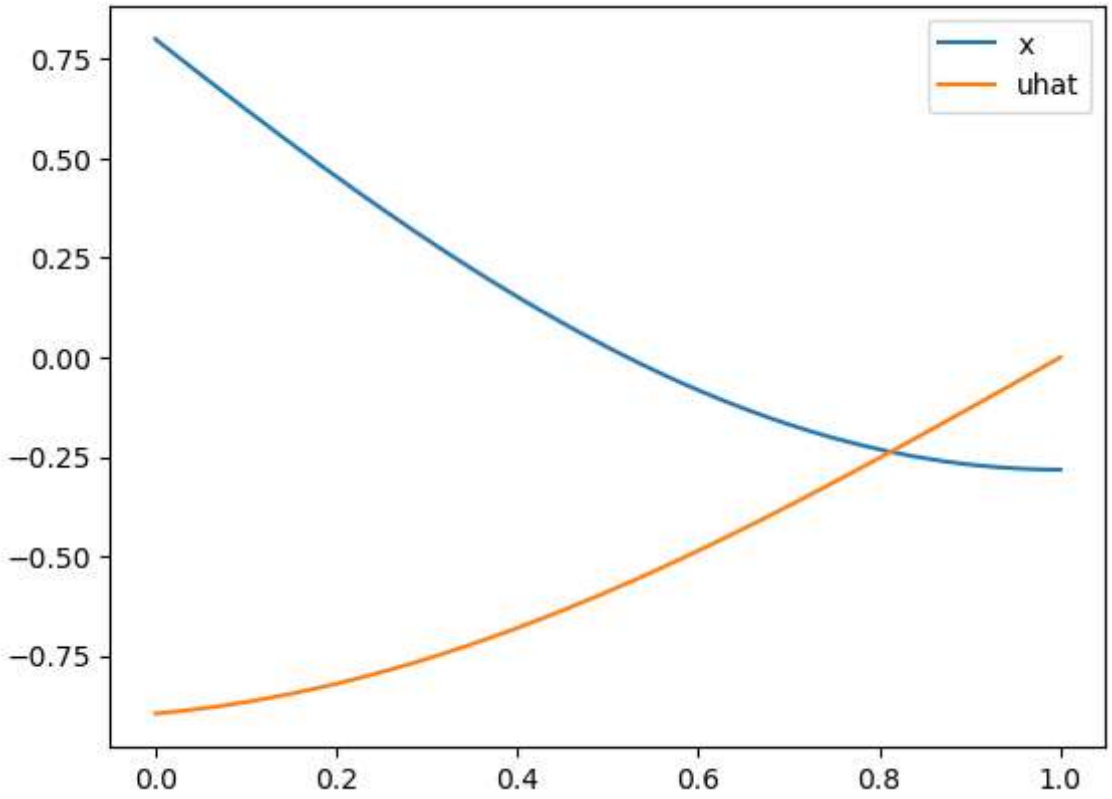
In [4]:
```
import numpy as np
from scipy.integrate import solve_bvp
from matplotlib import pyplot as plt
from scipy.integrate import solve_ivp
```

In [110…
```
E = 20
f = lambda t: 2000 #*np.sin(t)

def uhat(p1,p2):
     return (p2-p1)/2

def ode(t, X):
    ans = np.empty((4,t.size))
    x = X[0]
    y = X[1]
    p1 = X[2]
    p2 = X[3]
    return np.vstack((-a*np.exp(-((2*x/a)-2)**2) + uhat(p1,p2) + E,
                          f(t) - uhat(p1,p2),
                          p1*((8/a)*x-4)*np.exp(-((2*x/a)-2)**2)+2*x,
                          2*y))

def bc(ya,yb):
    return np.array([ya[0]-20, ya[1]-100, yb[2], yb[3]])

t = np.linspace(0,10)
y = np.zeros((4,t.size))
sol = solve_bvp(ode, bc, t, y)

t_plot = np.linspace(0,10,1000)
y_plot = sol.sol(t_plot)
plt.plot(t_plot, y_plot[0], label='x')
plt.plot(t_plot, y_plot[1], label='y')

# plt.legend()
# plt.show()

u1 = []
for i in range(len(y_plot[2])):
    u1.append(uhat(y_plot[2][i],y_plot[3][i]))


plt.plot(t_plot, u1, label='u1')
plt.legend()
plt.show()
```




In [90]:

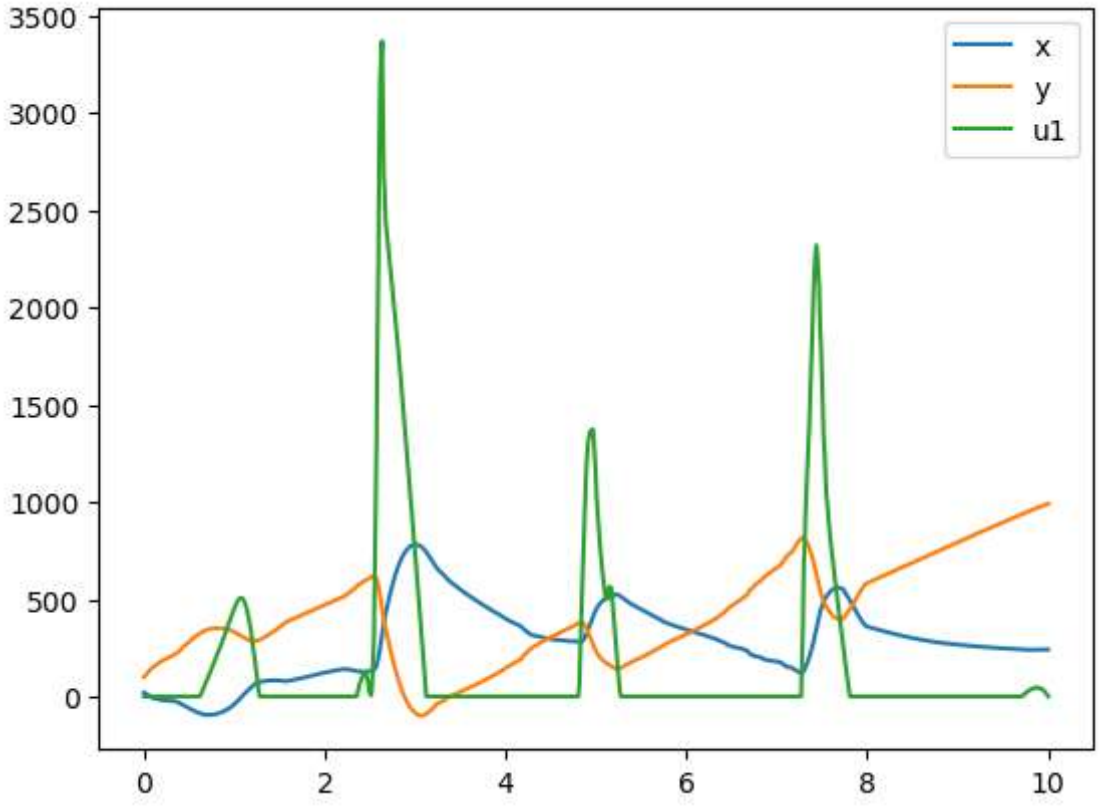
```
u = lambda p: p/2
xstart = .8
def ode(t,x):
    return np.vstack((x[1], 2-2*x[0]))

def bc(ya, yb):
    return np.array([ya[0]-xstart, yb[1]])

t = np.linspace(0,1)
y = np.zeros((2,t.size))
sol = solve_bvp(ode, bc, t, y)

t_plot = np.linspace(0,1,1000)
y_plot = sol.sol(t_plot)
plt.plot(t_plot, y_plot[0], label='x')

# plt.legend()
# plt.show()

uhat = u(y_plot[1])
plt.plot(t_plot, uhat, label='uhat')
plt.legend()
plt.show()
```




In [133…

```
E = 20
f = lambda t: 200 #*np.sin(t)

def uhat(p1,p2):
     return np.maximum((p2-p1)/2,0)


def ode(t, X):
    ans = np.empty((4,t.size))
    x = X[0]
    y = X[1]
    p1 = X[2]
    p2 = X[3]
    return np.vstack((-np.abs(x-100) + 100 + uhat(p1,p2) + E,
                          f(t) - uhat(p1,p2),
                          p1*((8/a)*x-4)*np.exp(-((2*x/a)-2)**2)+15*x,
                          2*y))
def residual(x, y):
    res = ode(x, y)
#     print('before',res)
    res[0] = np.maximum(res[0], 0) # Non-negativity constraint
    res[1] = np.maximum(res[1], 0)
#     print('after',res)
    return res


def bc(ya,yb):
    return np.array([ya[0]-20, ya[1]-100, yb[2], yb[3]])

t = np.linspace(0,10)
y = np.zeros((4,t.size))
sol = solve_bvp(ode, bc, t, y)
```

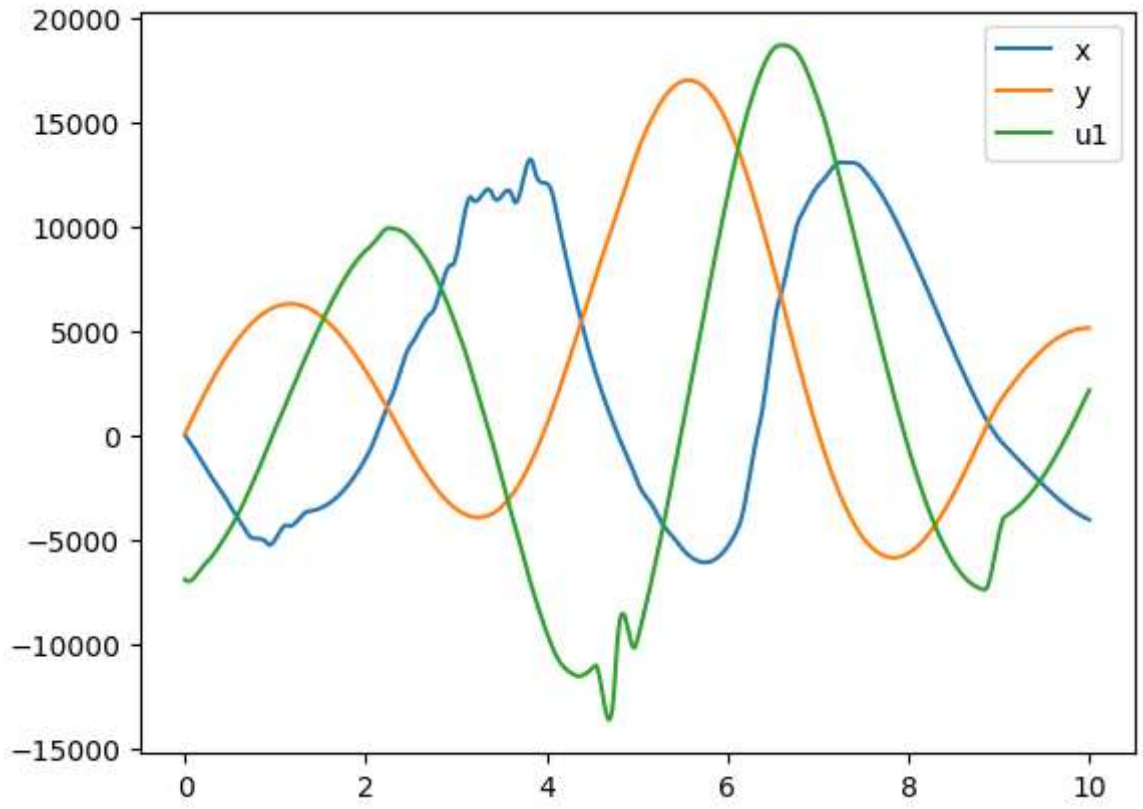
```
t_plot = np.linspace(0,10,1000)
y_plot = sol.sol(t_plot)
plt.plot(t_plot, y_plot[0], label='x')
plt.plot(t_plot, y_plot[1], label='y')

# plt.legend()
# plt.show()

u1 = []
for i in range(len(y_plot[2])):
    u1.append(uhat(y_plot[2][i],y_plot[3][i]))


plt.plot(t_plot, u1, label='u1')
plt.legend()
plt.show()
```




In [128…

```
E = 20
f = lambda t: 2000 #*np.sin(t)

def uhat(p1,p2):
     return (p2-p1)/2

def ode(t, X):
    ans = np.empty((4,t.size))
    x = X[0]
    y = X[1]
    p1 = X[2]
    p2 = X[3]
    return np.vstack((-np.abs(x-100) + 100 + uhat(p1,p2) + E,
                          f(t) - uhat(p1,p2),
                          p1*((8/a)*x-4)*np.exp(-((2*x/a)-2)**2)+2*x,
                          2*y))
```

```
def bc(ya,yb):
    return np.array([ya[0]-20, ya[1]-100, yb[2], yb[3]])

t = np.linspace(0,10)
y = np.zeros((4,t.size))
sol = solve_bvp(ode, bc, t, y)

t_plot = np.linspace(0,10,1000)
y_plot = sol.sol(t_plot)
plt.plot(t_plot, y_plot[0], label='x')
plt.plot(t_plot, y_plot[1], label='y')

# plt.legend()
# plt.show()

u1 = []
for i in range(len(y_plot[2])):
    u1.append(uhat(y_plot[2][i],y_plot[3][i]))


plt.plot(t_plot, u1, label='u1')
plt.legend()
plt.show()
```




In [ ]: