CMPT 419 – Assignment 2

1 Sequential Quadratic Programming

A. Use SQP like in our slides, but in 2 dimensions.

$$\text{minimize } f(x)$$
$$\text{subject to } g(x) \le 0$$
$$h(x) = 0$$

$$\nabla f(x) = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_n}\right)$$

- Objective: $\underset{d_x}{\text{minimize}} \ (r^k)^{\mathsf{T}} d_x + \frac{1}{2} d_x^{\mathsf{T}} B_k d_x$ where $d_x := x - x^k$, r^k, B_k
  - Depend on x^k
  - to be chosen
- Constraints: subject to $\nabla g(x^k)^{\mathsf{T}} d_x + g(x^k) \le 0$, $\nabla h(x^k)^{\mathsf{T}} d_x + h(x^k) = 0$

$$\mathrm{H}f(x) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$

Where,

$r_k = \nabla f(x_k)$
$B_k = \mathrm{H}f(x_k)$
$d_x := x - x_k$

So,

$x = x\_1$
$y = x\_2$

$r_k = \nabla f(x_k, y_k)$
$B_k = \mathrm{H}f(x_k, y_k)$
$d_x := x - x_k$

$$Minimize\ d_x \left(\left(r^k\right)^T d_x + \frac{1}{2}(d_x)^T B_k d_x\right)$$

$$Minimize\ d_x \left(\left(\nabla \mathrm{f}(x_k, y_k)\right)^T d_x + \frac{1}{2}\left(\mathrm{Hf}(x_k, y_k)\right)^T B_k d_x\right)$$

Constraints in standard form

$$-x - 1 \le 0$$
$$x - 1 \le 0$$
$$-y - 1 \le 0$$
$$y - 1 \le 0$$

Where,

$$x = x_k - d_{k,x}$$

$$y = y_k - d_{k,y}$$

$$f(x, y) = \sin(PI * x) * \sin(2 * PI * y)$$

Grad f(x,y) is,

d/dx(sin(π x) sin(2 π y)) = π cos(π x) sin(2 π y)

d/dy(sin(π x) sin(2 π y)) = 2 π sin(π x) cos(2 π y)

$$\frac{\partial}{\partial x}(\sin(\pi x)\sin(2\pi y)) = \pi\cos(\pi x)\sin(2\pi y)$$

$$\frac{\partial}{\partial y}(\sin(\pi x)\sin(2\pi y)) = 2\pi\sin(\pi x)\cos(2\pi y)$$

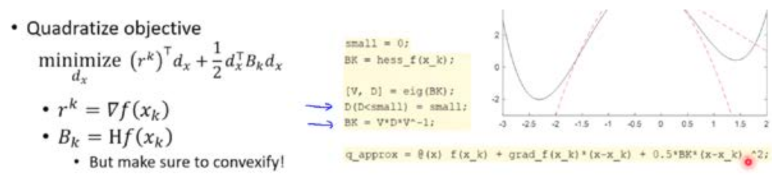
Hf(x,y) is,

(-π^2 sin(π x) sin(2 π y) | 2 π^2 cos(π x) cos(2 π y)

2 π^2 cos(π x) cos(2 π y) | -4 π^2 sin(π x) sin(2 π y))

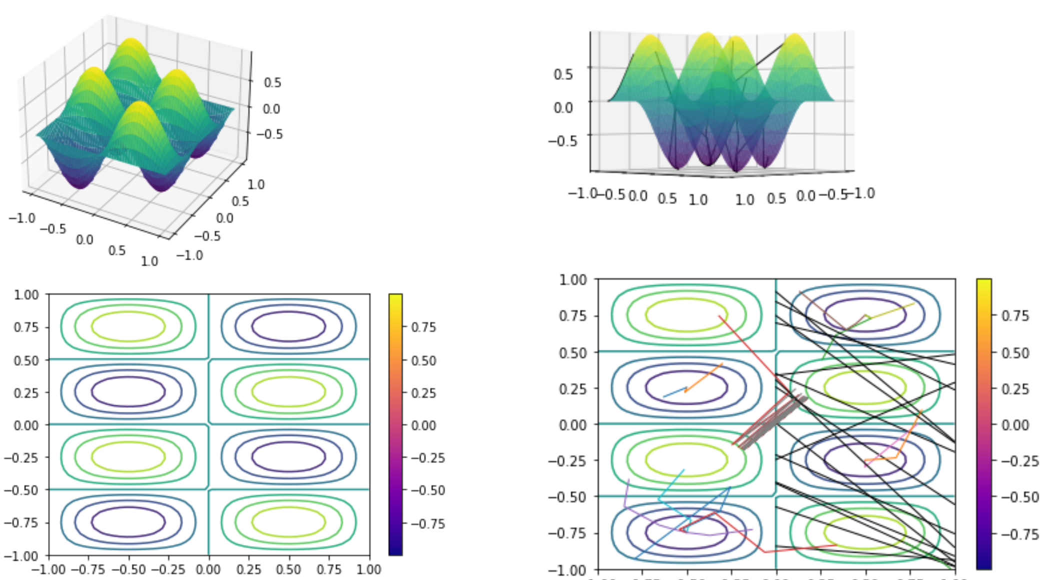
$$\begin{pmatrix} -\pi^2\sin(\pi x)\sin(2\pi y) & 2\pi^2\cos(\pi x)\cos(2\pi y) \\ 2\pi^2\cos(\pi x)\cos(2\pi y) & -4\pi^2\sin(\pi x)\sin(2\pi y) \end{pmatrix}$$

Note that will need to discard the negative eigs of Hf(x,y), and put in in Quad matrix form, as show on the slides.

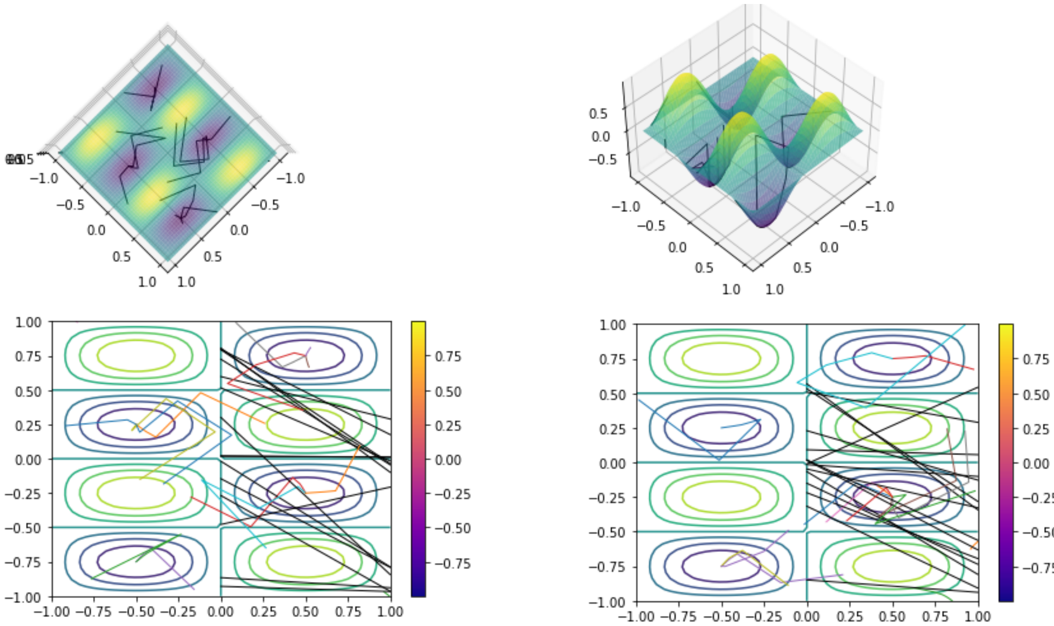


B. See C

C.



301314376



−1.0 −0.5 0.0 0.5 1.0
0.5 0.0 −0.5
1.00 0.75 0.50 0.25 0.00 −0.25 −0.50 −0.75 −1.00
−1.00 −0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00
0.75 0.50 0.25 0.00 −0.25 −0.50 −0.75

```
from random import random, randint
import numpy as np
import cvxpy as cp
# import matplotlib as mpl
import matplotlib.pyplot as plt
from matplotlib import cm
from mpl_toolkits import mplot3d

import warnings
warnings.filterwarnings("ignore") # I live on the edge :)

# howto install https://www.cvxpy.org/install/
# ref  http://yetanothermathprogrammingconsultant.blogspot.com/2019/11/cvxpy-matrix-style-modeling-limits.html

def cvx_problem(x1, x2):
    f = np.sin(np.pi *x1)*np.sin(2*np.pi*x2)

    gf = np.array([np.pi * np.cos(np.pi*x1)*np.sin(2*np.pi*x2),
                   2*np.pi*np.sin(np.pi*x1)*np.cos(2*np.pi*x2)])

    hf = np.array([[((-np.pi**2)*np.sin(np.pi*x1)*np.sin(2*np.pi*x2)),
(2*(np.pi**2)*np.cos(np.pi*x1)*np.cos(2*np.pi*x2))],
                   [(2*(np.pi**2)*np.cos(np.pi*x1)*np.cos(2*np.pi*x2)), (-
4*(np.pi**2)*np.sin(np.pi*x1)*np.sin(2*np.pi*x2))]])

    d, v = np.linalg.eig(hf) #convexify; zero-out neg eigens
    d[d<0] = 0
    d = np.matrix([[d[0], 0], [0, d[1]]])
    BK = v * d * np.linalg.inv(v)

    #q_approx = f + gf + 0.5*BK*()
#     print(f'v {v}')
#     print(f'vI {vI}')
#     print(f'BK {BK}')

    x = cp.Variable(2)
    bounds =       [-x1 - x[0] - 1 <= 0,
                     x1 + x[0] - 1 <= 0,
                    -x2 - x[1] - 1 <= 0,
                     x2 + x[1] - 1 <= 0]
    prob = cp.Problem(cp.Minimize((1/2)*cp.quad_form(x, BK) + gf.T @ x), bounds)
    p1 = prob.solve()
    return x.value


x = np.linspace(-1, 1)
y = np.linspace(-1, 1)
X, Y = np.meshgrid(x, y)
Z = np.sin(np.pi * X) * np.sin(2*np.pi*Y)

fig = plt.figure()
plt.xlim(-1,1)
plt.ylim(-1,1)
ax1 = fig.add_subplot(111)
ax1 = plt.axes(projection='3d')
ax1.plot_surface(X, Y, Z, rstride=1, cstride=1,
                cmap='viridis', edgecolor='none')
ax1.contour( X, Y, Z)
plt.show()
####

fig2 = plt.figure()
plt.xlim(-1,1)
plt.ylim(-1,1)
ax2 = fig2.add_subplot(111)
ax2 = plt.axes() # projection='2d'

ax2.contour( X, Y, Z)

m = cm.ScalarMappable(cmap=cm.plasma)
m.set_array(Z)
cbar = plt.colorbar(m)
plt.show()

###############
```

```
simulations = 15
epochs = 300   # max steps
small =0.00005 # convergence
step_size = 0.30
bool_switch = True
number_of_iterations = []

# mainview
fig = plt.figure()
ax1 = fig.add_subplot(111)
ax1 = plt.axes(projection='3d')
ax1.plot_surface(X, Y, Z, rstride=1, cstride=1,
                cmap='viridis', edgecolor='none', alpha=0.60, linewidth=0, antialiased=True)

#overview
fig2 = plt.figure()
plt.xlim(-1,1)
plt.ylim(-1,1)
ax2 = fig2.add_subplot(111)
ax2 = plt.axes() #
ax2.contour(X, Y, Z)
m = cm.ScalarMappable(cmap=cm.plasma)
m.set_array(Z)
cbar = plt.colorbar(m)


for j in range(0, simulations):
    # [0 to 1) -> [0 to 2) -> [-1 to 1)
    x1  = random()*2-1
    x2  = random()*2-1
    x1s = [x1]
    x2s = [x2]
    print(f'Initialization points: {x1:.2f} and {x2:.2f}')

    for i in range(0, epochs):
        point = cvx_problem(x1, x2)
        x1 += step_size*point[0]
        x2 += step_size*point[1]
        x1s.append(x1)
        x2s.append(x2)
#         print("the computed optimal value is " + str(point[0]) + " " + str(point[1]))
#         print(f'points after step: {x1s[-1]:.2f} and {x1s[-2]:.2f}')

        if i > 3 and np.abs(x1s[-1] - x1s[-2]) < small: # might not be the best way, but it works
            print(f'Convergence achieved, Terminal points: {x1s[-1]:.2f} and {x1s[-2]:.2f}\n')
            number_of_iterations.append(i)
            break
        if i == 100: print(f'This is the 100th iteration..')

    # supporting calculations from plot
    x1s, x2s = np.array(x1s), np.array(x2s)
    Z1 = np.sin(np.pi * x1s) * np.sin(2*np.pi*x2s)

    # mainview
    ax1.plot3D(x1s, x2s, Z1,'k', linewidth=1)

    # overview
    ax2.plot(x1s, x2s, Z1,'k', linewidth=1)


# ax1.view_init(elev=90, azim=45) # azim=45
ax1.view_init(elev=45, azim=45) # azim=45

plt.show()

print(f'With a step size of {step_size}, it took an average of {sum(number_of_iterations)/len(number_of_iterations):.2f}
iterations to achieve convergence. Small = {small}')
```

2 Differential Flatness

a)

$$\dot{x} = v\cos(\theta)$$
$$\dot{y} = v\sin(\theta)$$
$$\dot{\theta} = \omega$$
$$\dot{v} = a$$

Obtain heading and longitudinal,

$$\frac{\dot{y}}{\dot{x}} = \frac{\sin(\theta)}{\cos(\theta)} = \tan(\theta)$$
$$\theta = \arctan\left(\frac{\dot{y}}{\dot{x}}\right)$$
$$v = \sqrt{(\dot{x})^2 + (\dot{y})^2}$$

Obtain the turn rate,

$$\dot{\theta} = \omega$$
$$\theta = \arctan\left(\frac{\dot{y}}{\dot{x}}\right)$$
$$so, \dot{\theta} = \omega = \frac{d\theta}{dt}\arctan\left(\frac{\dot{y}}{\dot{x}}\right)$$

Obtain the longitudinal acceleration,

$$\dot{v} = a = \frac{dv}{dt}\sqrt{(v\cos(\theta))^2 + (v\sin(\theta))^2}$$

Show that the system is differentially flat by letting z = (x; y), and deriving the functions β and γ,

$$z = (x, y) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}\begin{bmatrix} x \\ y \\ \theta \end{bmatrix}$$

We saw that in lecture, but we also could have z equal the **Identity matrix of 2*2, times** transpose of **[x,y]**

β takes in $\left[x, y, \arctan\left(\frac{\dot{y}}{\dot{x}}\right), \sqrt{(\dot{x})^2 + (\dot{y})^2}\,\right]$, and gives us our state, $[\mathrm{x}, \mathrm{y}, \theta, v]$

γ takes in $[\dot{\theta}, \dot{v}\,]$, *and gives us our control* $[\omega, a]$

which is also,

γ takes in $\left[\frac{d\theta}{dt}\arctan\left(\frac{\dot{y}}{\dot{x}}\right), \frac{dv}{dt}\sqrt{(v\cos(\theta))^2 + (v\sin(\theta))^2}\right]$, *and gives us our control* $[\omega, a]$

The LHS can be formulated as a function of x, y, and the derivatives.

This forms the, $\begin{aligned}(x, y, \theta, v) &= \beta(z, \dot{z}, \ldots, z^{(q)}) \\ (\omega, a) &= \gamma(z, \dot{z}, \ldots, z^{(q)})\end{aligned}$, which we wanted

b)

We start by expanding the following:

Using the basis functions $\psi_0(t) = 1, \psi_1(t) = t, \psi_2(t) = t^2, \psi_3(t) = t^3$, we parameterize the flat outputs as follows:

$$x(t) = \sum_{i=0}^{3} b_{0i}\psi_i(t) \tag{11}$$

$$y(t) = \sum_{i=0}^{3} b_{1i}\psi_i(t) \tag{12}$$

$$x(t) = b_{0,0} + b_{0,1}t + b_{0,2}t^2 + b_{0,3}t^3$$

$$y(t) = b_{1,0} + b_{1,1}t + b_{1,2}t^2 + b_{1,3}t^3$$

get d/dt

$$\dot{x}(t) = b_{0,1} + 2b_{0,2}t^1 + 3b_{0,3}t^2$$
$$\dot{y}(t) = b_{1,1} + 2b_{1,2}t^1 + 3b_{1,3}t^2$$

$$\begin{bmatrix} x(0) \\ y(0) \\ \theta(0) \\ v(0) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}, \quad \begin{bmatrix} x(T) \\ y(T) \\ \theta(T) \\ v(T) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \frac{\pi}{2} \\ 1 \end{bmatrix}$$

Recall that, $\theta = \arctan\left(\frac{\dot{y}}{\dot{x}}\right)$, then ignore that as we were given the above image....

$$\dot{x}(0) = v(0)\cos(\theta(0)) = 1\cos(0) = 1$$
$$\dot{y}(0) = v(0)\sin(\theta(0)) = 1\sin(0) = 0$$
$$\dot{x}(T) = v(T)\cos(\theta(T)) = 1\cos\left(\frac{PI}{2}\right) = 0$$
$$\dot{y}(T) = v(T)\sin(\theta(T)) = 1\sin\left(\frac{PI}{2}\right) = 1$$
$$x(0), x(T) = 0$$
$$y(0), y(T) = 0$$

$$x \; and \; \dot{x} \; form, \quad \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & T & T^2 & t^3 \\ 0 & 1 & 2T & 3T^2 \end{bmatrix} \begin{bmatrix} b_{0,0} \\ b_{0,1} \\ b_{0,2} \\ b_{0,3} \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

$$y \; and \; \dot{y} \; form, \quad \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & T & T^2 & t^3 \\ 0 & 1 & 2T & 3T^2 \end{bmatrix} \begin{bmatrix} b_{1,0} \\ b_{1,1} \\ b_{1,2} \\ b_{1,3} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

c)

Looks like we'll need w(0), w(T), a(0) and a(T), so recall that

$$\beta \text{ takes in } \left[x, y, \arctan\left(\frac{\dot{y}}{\dot{x}}\right), \sqrt{(\dot{x})^2 + (\dot{y})^2}\,\right], \text{and gives us our state, } [\mathrm{x}, \mathrm{y}, \theta, v]$$

$$\gamma \text{ takes in } \left[\dot{\theta}, \dot{v}\,\right], and \; gives \; us \; our \; control \; [\omega, a]$$

Let's elaborate on the control policy,

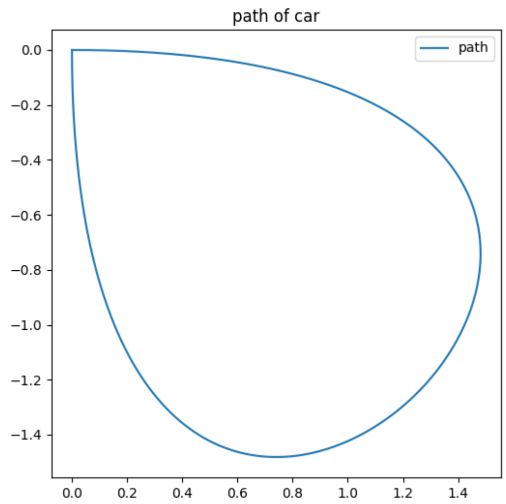
$$a(\mathrm{t}) = \dot{v} = \frac{\mathrm{d}}{\mathrm{dt}}\mathrm{v}^2 \; = \frac{\mathrm{d}}{\mathrm{dt}}\,(\dot{x})^2 + (\dot{y})^2$$
$$\mathrm{a(t)} = \; 2\mathrm{v}\dot{v} = 2\dot{x}\ddot{x} + \; 2\dot{y}\ddot{y}$$
$$a(\mathrm{t}) = \frac{\dot{x}\ddot{x} + \; \dot{y}\ddot{y}}{\mathrm{v}}$$

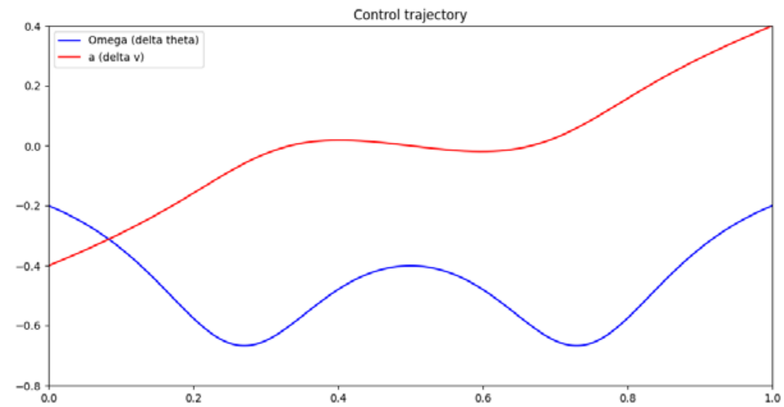
$$\omega(t) = \dot{\theta} = \frac{d}{dt}\arctan\left(\frac{\dot{y}}{\dot{x}}\right)$$
$$\omega(\mathrm{t}) = \frac{\ddot{y}\dot{x} - \; \ddot{x}\dot{y}}{\dot{x}^2 + \; \dot{y}^2}$$
$$\omega(\mathrm{t}) = \frac{\ddot{y}\dot{x} - \; \ddot{x}\dot{y}}{\mathrm{v}^2}$$

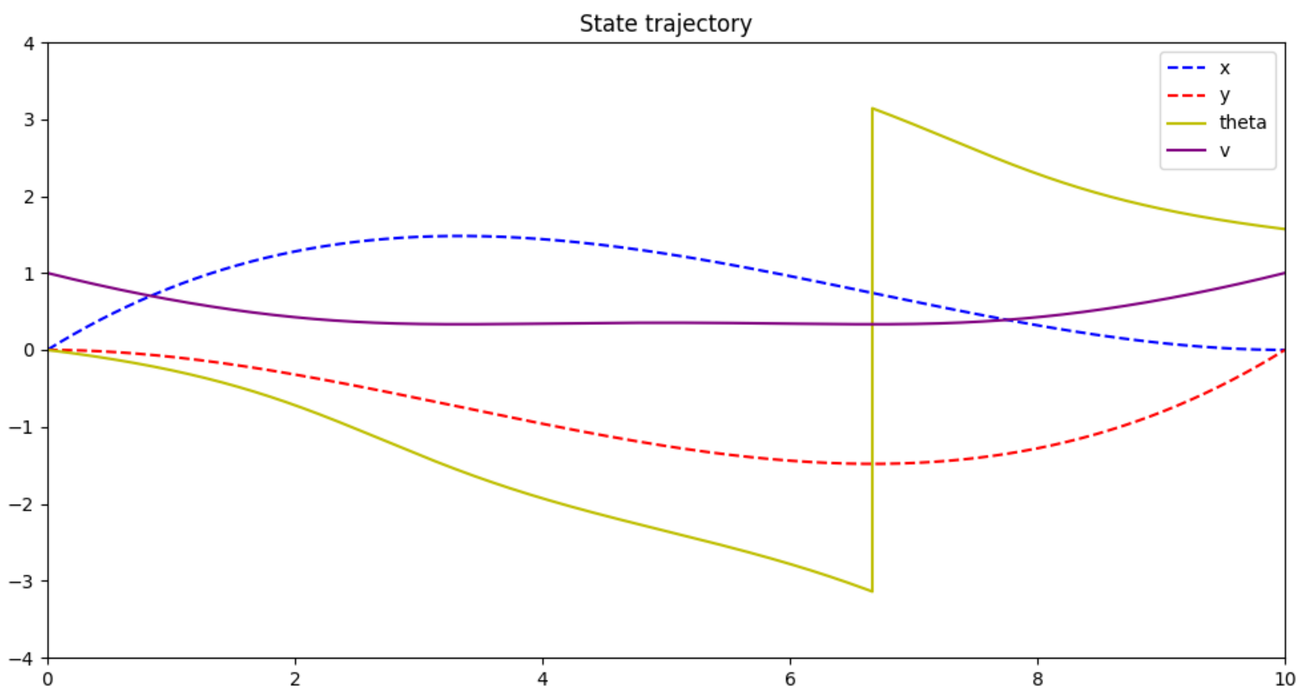
See code for rest, no point in doing unneeded work/detail

See Q2.py

301314376


path of car
path


Control trajectory
Omega (delta theta)
a (delta v)


State trajectory
x
y
theta
v

3 Multiple Shooting with Casadi

a.

$$M = 0.5kg$$
$$m = 0.2kg$$
$$l = 0.3m$$
$$I = 0.006kg * m^2$$
$$b = 0.1\text{N/m/s}$$
$$\mathrm{g} = 9.81\frac{\mathrm{M}}{\mathrm{s}^2}$$
$$\dot{x} = \mathrm{v}$$
$$\dot{\theta} = \omega$$

Want,

$$\min_{U(.)} \int_{t=0}^{5} F^2(t)dt$$

Subject to,

$$S(t) = [\,x, y, \theta, \omega]$$
$$S(t = 0) = [0, 0, 0, 0]$$
$$s(t = 5) = [0, 0, PI, 0]$$
$$ensuring\ |x(t)| \leq 1$$
$$ensuring\ |F(t)| \leq 0.2$$
$$the\ later\ is, |u(t)| \leq 0.2$$

$$(M+m)\ddot{x} + b\dot{x} + ml\ddot{\theta}\cos\theta - ml\dot{\theta}^2\sin\theta = F$$
$$(I+ml^2)\ddot{\theta} + mgl\sin\theta = -ml\ddot{x}\cos\theta$$

$$\begin{bmatrix} M+m & ml\cos\theta \\ ml\cos\theta & I+ml^2 \end{bmatrix} * \begin{bmatrix} \dot{v}(t) \\ \dot{\omega}(t) \end{bmatrix} = \begin{bmatrix} -bv + ml\omega^2\sin\theta + F \\ -mgl\sin\theta \end{bmatrix}$$

$$\dot{x}(t) = v(t)$$
$$\dot{\theta}(t) = \ \omega(t)$$

b.

Want a NLP with N=50, forward Euler & Left first order integration.

Where i is the discretized representation of a time point. h is the stepsize, T/N

$$\min_{U(.)} h \sum_{i=0}^{N-1} F_i^2$$

Subject to,

$$S(t) = [\, x, y, \theta, \omega]$$
$$S(t=0) = [0, 0, 0, 0]$$
$$s(t=N) = [0, 0, PI, 0]$$
$$ensuring\ |x_i| \leq 1$$
$$ensuring\ |F_i| \leq 0.2$$
$$the\ later\ is, |u_i| \leq 0.2$$

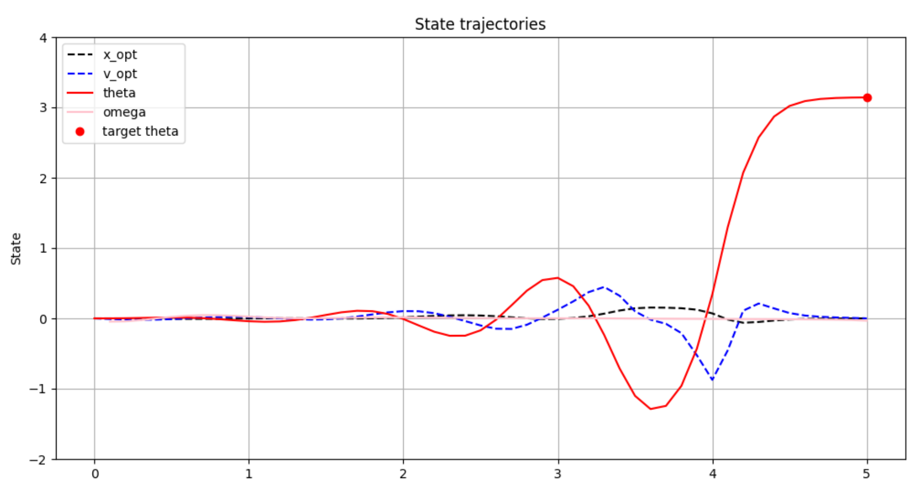
$$(M+m)\ddot{x} + b\dot{x} + ml\ddot{\theta}\cos\theta - ml\dot{\theta}^2\sin\theta = F$$
$$(I+ml^2)\ddot{\theta} + mgl\sin\theta = -ml\ddot{x}\cos\theta$$

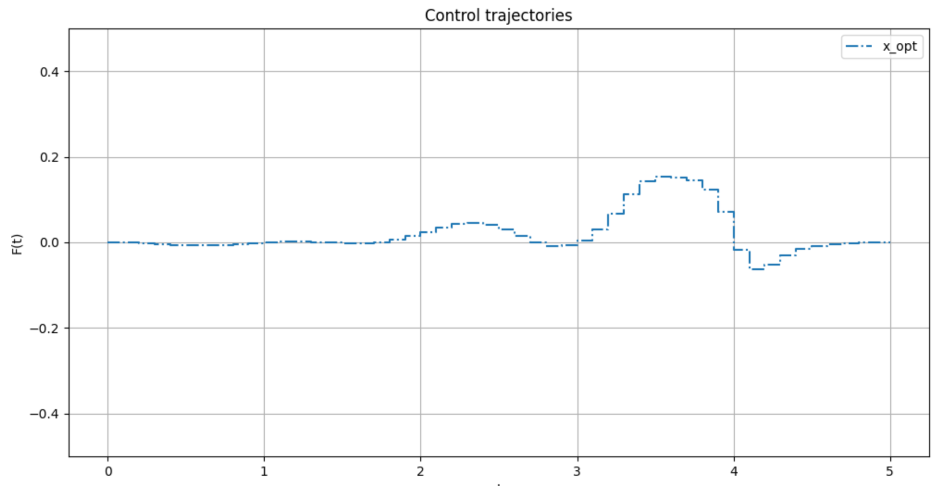
$$\begin{bmatrix} M+m & ml\cos\theta \\ ml\cos\theta & I+ml^2 \end{bmatrix} * \begin{bmatrix} \dot{v}_i \\ \dot{\omega}_i \end{bmatrix} = \begin{bmatrix} -bv + ml\omega^2\sin\theta + F \\ -mgl\sin\theta \end{bmatrix}$$

$$\dot{x}_i = v_i$$
$$\dot{\theta}_i = \omega_i$$

Every i is from [0..5], except the i in u_i, which is [0..5). We are not supposed to have a control in the last moment

c.





My code is based on the official docs

4 Robotic Safety via Reachability Analysis

$$\dot{x} = v * cos(\theta) + d_x$$
$$\dot{y} = v * \sin(\theta) + d_y$$
$$\dot{\theta} = \omega$$
$$\dot{v} = a$$

d_x and d_y are wind disturbance

under the following controls

$$|\omega| \leq 0.5\frac{rad}{s}$$
$$|a| \leq 10\frac{m}{s^2}$$

a) where r is 1

$$T = \left\{ (x, y, \theta, v) : \sqrt{x^2 + y^2} \leq r \right\} \subseteq \mathbb{R}^4$$

b) find a suitable cost function l(x,y,theta,v) >= 0, which is in T

$$l(T, x(T)) = l(x_r, y_r, \theta_r, v_r) = \sqrt{x^2 + y^2} - r$$

Speed doesn't not matter, except in the derivative of the cost function

c)

$$z = \{x, y, \theta, v\}$$
$$u = \{w, a\}$$

v(t,z)

$$\min \left\{ \frac{\partial V}{\partial t}(t, z) + \max_{u \in \mathcal{U}} \min_{d \in \mathcal{D}} \frac{\partial V}{\partial z}(t, z)^\top f(z, u, d), l(z) - V(t, z) \right\} = 0.$$

Find

$$u^*(t, x, y, \theta, v) = \arg\max_{u \in \mathcal{U}} \min_{d \in \mathcal{D}} \frac{\partial V}{\partial z}(t, z)^\top f(z, u, d),$$
$$d^*(t, x, y, \theta, v) = \arg\min_{d \in \mathcal{D}} \frac{\partial V}{\partial z}(t, z)^\top f(z, u^*, d),$$

$$f(x, y, \theta, v, u, d) = \left[v * cos(\theta) + d_x, v * \sin(\theta) + d_y, u\right]^T$$

Note that vector is transposed. Not to the power of T

$$\frac{dV}{dz}V(t, x, y, \theta, v) = \left[\frac{\partial V}{\partial x}, \frac{\partial V}{\partial y}, \frac{\partial V}{\partial \theta}, \frac{\partial V}{\partial v}\right]^T$$

$$u^*(t,x,y,\theta,v) = \arg\max_{u\in\mathcal{U}}\min_{d\in\mathcal{D}} \frac{\partial V}{\partial x}(v * \cos(\theta) + d_x) + \frac{\partial V}{\partial y}(v * sin(\theta) + d_y) + \frac{\partial V}{\partial \theta}(u)$$

The above is a single expression <u>including</u> the part in the image

Note that,

$$\frac{\partial V}{\partial x}(v * \cos(\theta) + d_x) + \frac{\partial V}{\partial y}(v * sin(\theta) + d_y) + \frac{\partial V}{\partial \theta}(u + d_\theta)$$
$$= \frac{\partial V}{\partial x}(v * \cos(\theta)) + \frac{\partial V}{\partial x}(d_x) + \frac{\partial V}{\partial y}(d_y) + \frac{\partial V}{\partial y}(v * sin(\theta)) + \frac{\partial V}{\partial \theta}(u)$$

From that,

$$u^*(t,x,y,\theta,v) = \arg\max_{u\ in\ U} \frac{\partial V}{\partial \theta}(u) = \text{max\_possible}(u) * \frac{\partial V}{\partial \theta} = 0.5 * \frac{\partial V}{\partial \theta}$$
$$\frac{\partial V}{\partial \theta}\ is\ 1\ if\ positive, else\ it's - 1. this\ makes\ its\ dicrete$$

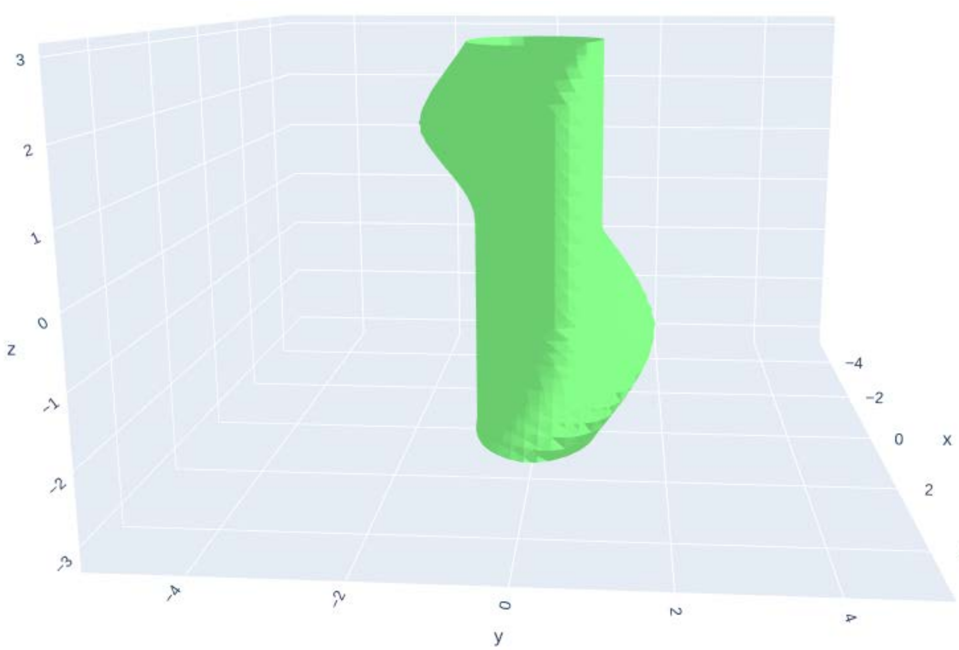
And also,

Since -25 is the smallest number such the |d_x| <= 25m/s

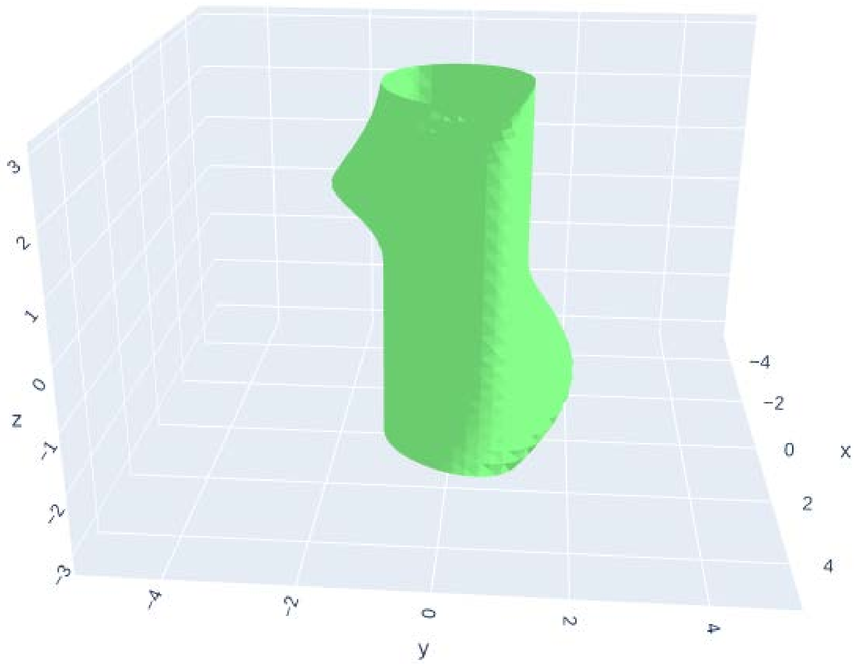
$$d_x = -25\frac{\partial V}{\partial x}$$
$$d_y = -25\frac{\partial V}{\partial y}$$
$$a = 10\ m/s^2$$

d)

Protruding flop is very depended on the lookback_length. The flop goes to y = +/- 1.9

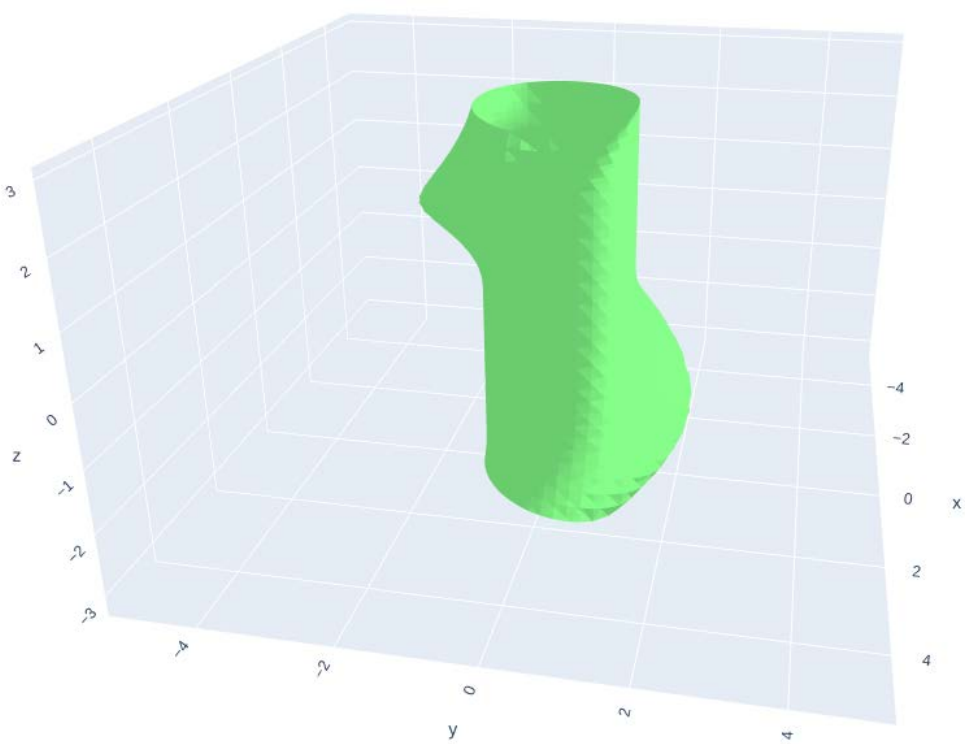


e)
The shape is more oval, but the flop still goes to y = +/- 1.9



f)

The flop is larger and more pronounced, it goes to about y = +/- 2.1



```
import numpy as np
# Utility functions to initialize the problem
from Grid.GridProcessing import Grid
from Shapes.ShapesFunctions import *
# Specify the  file that includes dynamic systems
from dynamics.DubinsCar4D import *
from dynamics.DubinsCapture import *
from dynamics.DubinsCar4D2 import *
# Plot options
from plot_options import *
# Solver core
from solver import HJSolver

import math

""" USER INTERFACES
- Define grid

- Generate initial values for grid using shape functions

- Time length for computations

- Initialize plotting option

- Call HJSolver function
"""

# Second Scenario
g = Grid(np.array([-5.0, -5.0,-5.0, -math.pi]), np.array([5.0, 5.0, 5.0, math.pi]), 4,
         np.array([50, 50, 30, 35]), [3])

# Define my object

dMin = [0, 0]
dMax = [0, 0]

uMin = [-0.01, -0.05]
uMax = [0.01, 0.05]

## part e - change control
uMin = [0.08, 0.08]
uMax = [0.3, 0.3]

## Part f - set disturbance
dMin = [-0.025,-0.025]
dMax = [0.025, 0.025]

my_car = DubinsCar4D(uMin=uMin, uMax=uMax, dMin=dMin, dMax=dMax,
         uMode="max", dMode="min")

# Use the grid to initialize initial value function
# Initial_value_f = CylinderShape(g, [3,4], np.zeros(4), 1)
init_val_f  = CylinderShape(g, [2, 3], np.zeros(4), 1)

#print("what is target set shape:" , Initial_value_f.shape[0])
## a = Lower_Half_Space(g,   0, 0.01)  # 10km/s
## b = Upper_Half_Space(g ,  0, 0.08)  #
## c = Lower_Half_Space(g ,  1, 0.01)  # 10km/s
## d = Upper_Half_Space(g ,  1, 0.08)  #

## part e - change control
a = Lower_Half_Space(g,   2, 0.08)
b = Upper_Half_Space(g ,  2, 0.3)

# print(init_val_f)
# print(init_val_f[:, :, 0, :])
# print(init_val_f[:, :, 1, :])

init_val_f = np.subtract(init_val_f, a)
init_val_f = np.subtract(init_val_f, b)
# init_val_f = np.subtract(init_val_f, c)
# init_val_f = np.subtract(init_val_f, d)


print("shape: ",init_val_f.shape)


# Look-back length and time step
lookback_length = 6.0 # was 3.0
t_step = 0.05 # 0.075 #0.1

small_number = 1e-5
tau = np.arange(start=0, stop=lookback_length + small_number, step=t_step)

po = PlotOptions("3d_plot", [0,1,3], [15])
HJSolver(my_car, g, init_val_f, tau, "minVWithV0", po)
```