

# EEOR 4650: Homework 5

Ruofan Meng, Columbia UID: rm3726

December 7, 2020

Estimated time spent on homework: 300

## Question 1

(a)

If we use quadratic function to measure the tracking error, to make  $y(t) - y_{des}(t)$  small is to minimize  $J_{track} = \frac{1}{N+1} \sum_{t=0}^N (y(t) - y_{des}(t))^2$ . With  $y = (y(0), \dots, y(N))^T \in \mathbf{R}^{N+1}$  and  $y_{des} = (y_{des}(0), \dots, y_{des}(N))^T \in \mathbf{R}^{N+1}$ , it is equivalent to minimize  $\|y - y_{des}\|_2 = \sqrt{\sum_{t=0}^N (y(t) - y_{des}(t))^2}$ , because  $J_{track} = \frac{1}{N+1} \|y - y_{des}\|_2^2$ .

Clearly, take  $b = y_{des} \in \mathbf{R}^{N+1}$  and  $A \in \mathbf{R}^{(N+1) \times (N+1)}$

$$A = \begin{bmatrix} h(0) & 0 & 0 & \cdots & 0 \\ h(1) & h(0) & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ h(N) & h(N-1) & h(N-2) & \cdots & h(0) \end{bmatrix}$$

we have  $y = Ax$  with  $x = (u(0), \dots, u(N))^T \in \mathbf{R}^{N+1}$ . This means  $\|y - y_{des}\|_2 = \|Ax - b\|_2$ . Therefore, the tracking problem can be written as

$$\min_x \|Ax - b\|_2$$

(b)

The choice of norm should depend on the loss function with respect to the tracking error. If we use quadratic function,  $\|\cdot\|_2$  should be used; if we use  $p$ th order polynomial function,  $\|\cdot\|_p$  should be used; for other loss functions, we could define corresponding norms.

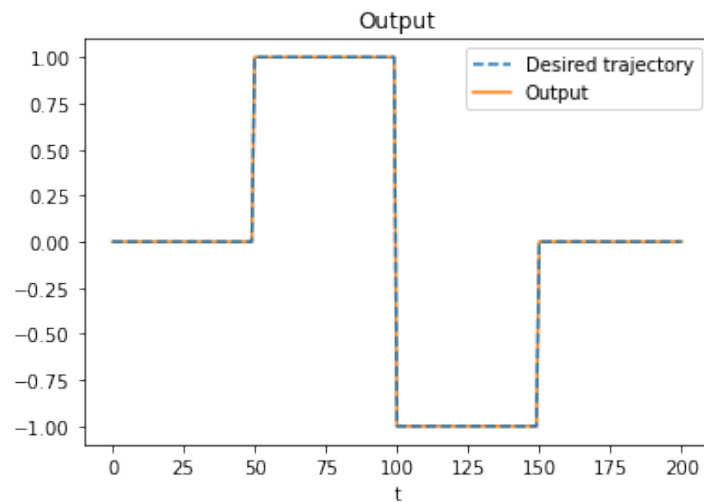
```
In [2]: 1 y_des_1b = np.load('y_des_1b.npz')
        2 y1 = y_des_1b['y_des'].flatten()
```

```
In [10]: 1 N=len(y1)
        2 def h(x):
        3     return 0.9**x*(1-0.4*np.cos(2*x))/9
        4 A=np.zeros((N,N))
        5 for i in range(N):
        6     for j in range(i+1):
        7         A[i][j]=h(i-j)
```

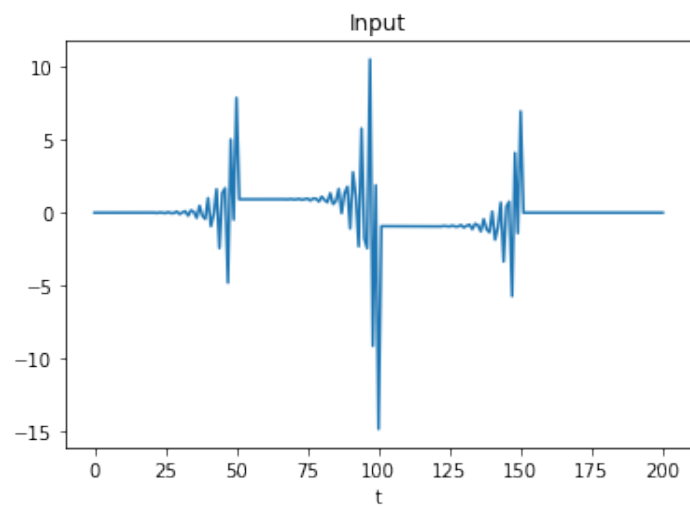
```
In [18]: 1 x = cp.Variable(N)
        2 constraints = []
        3 prob = cp.Problem(cp.Minimize(cp.norm(A@x-y1)), constraints)
        4 prob.solve()
```

Out[18]: 9.095882644690729e-08

The optimal value of our target function is  $9.096 \times 10^{-8}$ , which implies a well tracking. Our output and the desired trajectory are shown as followed.

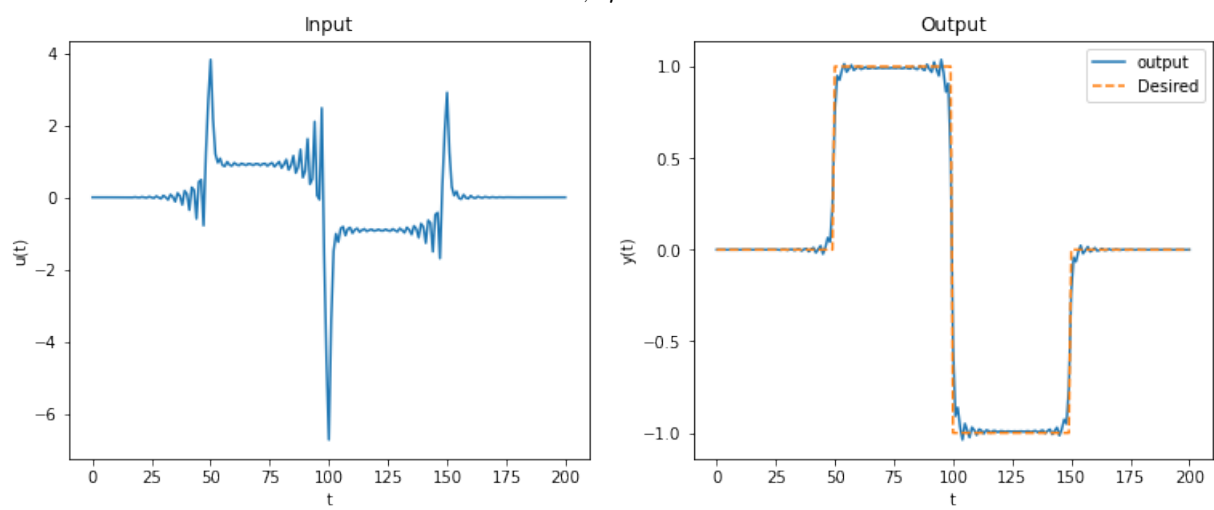


Our input looks like below.

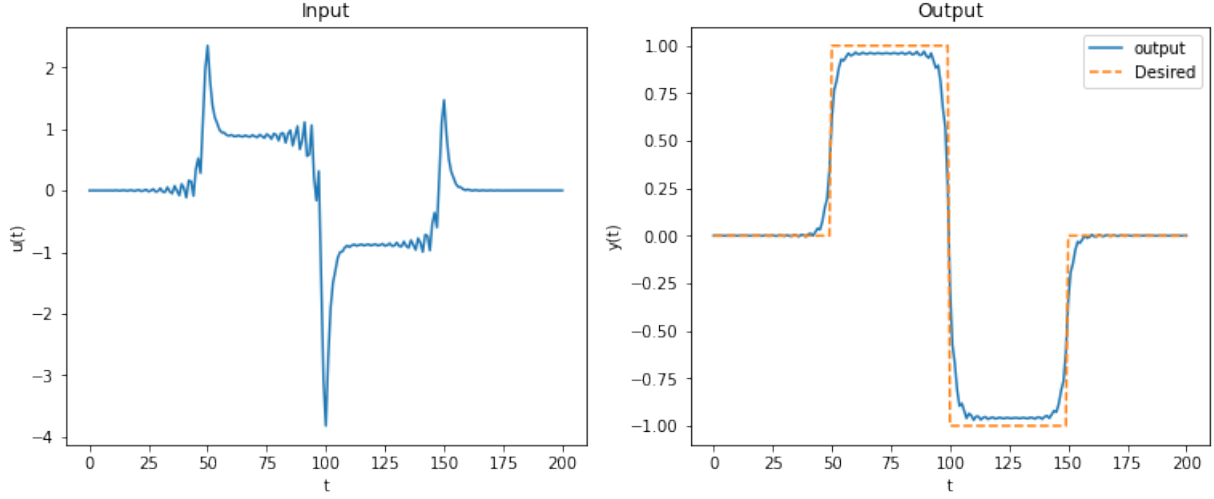


(c)

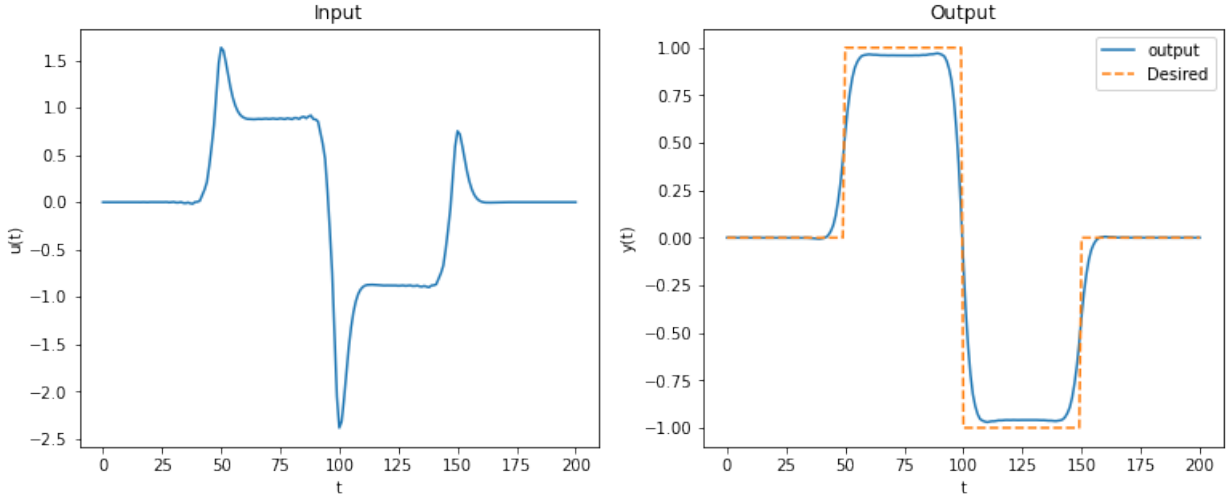
$$\delta = 0, \eta = 0.01$$



$$\delta = 0, \eta = 0.05$$



$$\delta = 0.3, \eta = 0.05$$



(d)

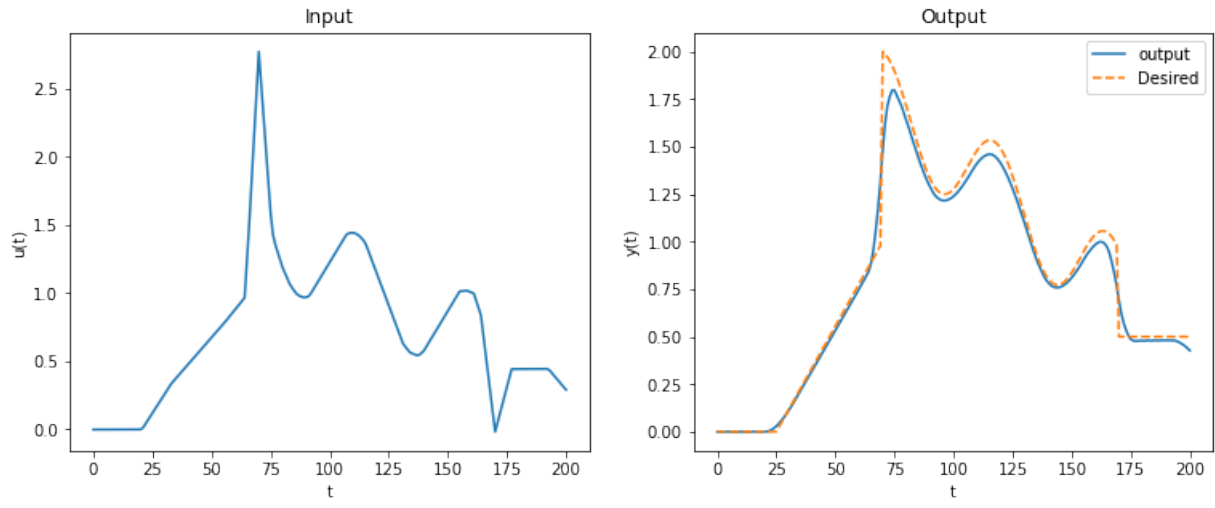
To encourage piecewise-linear input, we could change the smoothness loss part into second-order variation.

The original smoothness loss is quadratic function of the first-order variation  $J_{der} = \frac{1}{N} \sum_{t=0}^{N-1} (u(t+1) - u(t))^2$ , it works by encouraging small first-order variation because the square of a small value is even smaller. For piecewise-linear function, we want it to have 0 second-order derivative almost everywhere. Though with second-order variation, at some points where the function changes its slope, it would have some large second-order variation. Actually, those values will turn small when divided by the number of all points.

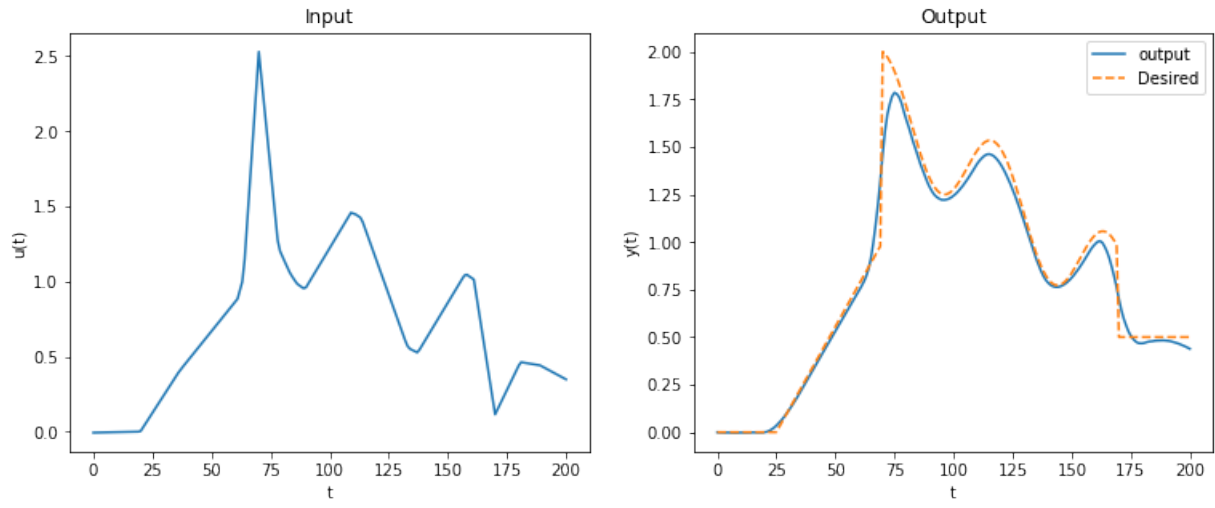
What's more, instead of encouraging small variation, we want 0 second-order variation, which means absolute value function would perform better. Therefore, we should modify  $J_{der}$  as  $J_{der} \frac{1}{N-1} \sum_{t=1}^{N-1} |2u(t) - u(t+1) - u(t-1)|$ .

My final parameters are  $\delta = 0.6$  and  $\eta = 0.05$ .

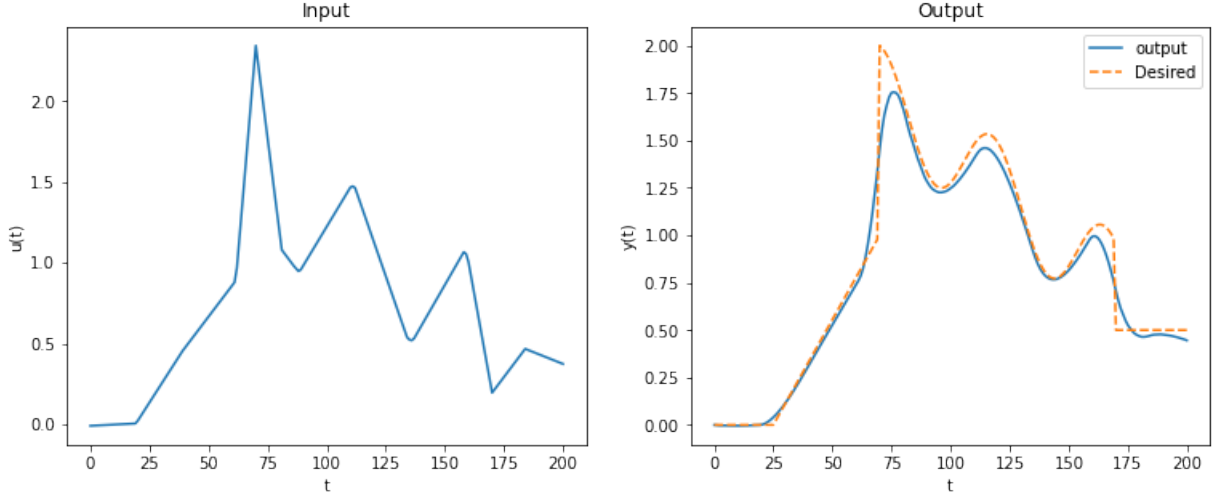
$$\delta = 0.1, \eta = 0.05$$



$$\delta = 0.3, \eta = 0.05$$



$$\delta = 0.6, \eta = 0.05$$



## Question 2

(a)

Denote  $f(a, b) = \max_i \left| \frac{p(t_i)}{q(t_i)} - y_i \right|$ ,  $D = \{(a, b) \mid \forall t \in [\alpha, \beta], q(t) > 0\}$  and  $S_\alpha = \{(a, b) \in D \mid f(a, b) \leq \alpha\}$ .

First notice that  $D$  is convex, for any two points  $(x, y)$  and  $(m, n)$  in  $D$ , we have  $\forall t \in [\alpha, \beta]$ ,  $1 + y_1 t + \dots + y_n t^n > 0$  and  $1 + n_1 t + \dots + n_n t^n > 0$  and thus for  $0 \leq \theta \leq 1$ ,  $1 + (\theta y_1 + (1 - \theta) n_1) t + \dots + (\theta y_n + (1 - \theta) n_n) t^n > 0$ , which means  $(\theta x + (1 - \theta) m, \theta y + (1 - \theta) n) \in D$ . So  $S_\alpha$  is convex if  $\{(a, b) \mid f(a, b) \leq \alpha\}$  is convex. With  $A_{\alpha, i} = \{(a, b) \mid q(t_i)(y_i - \alpha) \leq p(t_i) \leq q(t_i)(y_i + \alpha)\}$ , we have

$$\begin{aligned}
 \{(a, b) \mid f(a, b) \leq \alpha\} &= \{(a, b) \mid \max_i \left| \frac{p(t_i)}{q(t_i)} - y_i \right| \leq \alpha\} \\
 &= \{(a, b) \mid \forall i, \left| \frac{p(t_i)}{q(t_i)} - y_i \right| \leq \alpha\} \\
 &= \bigcap_{i=1}^k \{(a, b) \mid \left| \frac{p(t_i)}{q(t_i)} - y_i \right| \leq \alpha\} \\
 &= \bigcap_{i=1}^k \{(a, b) \mid q(t_i)(y_i - \alpha) \leq p(t_i) \leq q(t_i)(y_i + \alpha)\} \\
 &= \bigcap_{i=1}^k A_{\alpha, i}
 \end{aligned}$$

With  $p_1(t) = x_0 + \dots + x_m t^m$ ,  $p_2(t) = m_0 + \dots + m_m t^m$ ,  $q_1(t) = 1 + y_1 t + \dots + y_n t^n$ ,  $q_2(t) = 1 + n_1 t + \dots + n_n t^n$ ,  $(\theta p_1 + (1 - \theta) p_2)(t) = (\theta x_0 + (1 - \theta) m_0) + \dots + (\theta x_m + (1 - \theta) m_m) t^m$  and  $(\theta q_1 + (1 - \theta) q_2)(t) = (\theta + (1 - \theta)) + \dots + (\theta y_n + (1 - \theta) n_n) t^n$ , clearly we could have

$$(\theta q_1 + (1 - \theta) q_2)(t_i)(y_i - \alpha) \leq (\theta p_1 + (1 - \theta) p_2)(t_i) \leq (\theta q_1 + (1 - \theta) q_2)(t_i)(y_i + \alpha)$$

which means  $(\theta x + (1-\theta)m, \theta y + (1-\theta)n) \in A_{\alpha,i}$  and  $A_{\alpha,i}$  is convex. Thus  $\{(a, b) \mid f(a, b) \leq \alpha\}$  is convex, the sublevel set  $S_\alpha$  is convex, and the objective functions is a quasiconvex function. The problem is quasiconvex programming.

(b)

Python code

```
In [2]: 1 rational_fit_data = np.load('rational_fit_data.npz')
2 y = rational_fit_data['y_des'].flatten()
3 t = rational_fit_data['t'].flatten()
```

```
In [3]: 1 def phi(a,b,c):
2     def p(x):
3         return a[0]+a[1]*x**1+a[2]*x**2+a[3]*x**3+a[4]*x**4\
4             +a[5]*x**5+a[6]*x**6+a[7]*x**7+a[8]*x**8+a[9]*x**9
5     def q(x):
6         return 1+b[0]*x**1+b[1]*x**2+b[2]*x**3+b[3]*x**4+b[4]*x**5\
7             +b[5]*x**6+b[6]*x**7+b[7]*x**8
8     return cp.max(cp.abs(p(t)-cp.multiply(y,q(t))))-c*q(t))
```

```
In [4]: 1 l=0
2 u=1
3 epsilon=0.0001
4 convergence=[]
5 while (u-l)>epsilon:
6     convergence.append(u-l)
7     x=(l+u)/2
8     a = cp.Variable(10)
9     b = cp.Variable(8)
10    constraints = [phi(a,b,x)<=0]
11    prob = cp.Problem(cp.Minimize(1), constraints)
12    prob.solve()
13    if prob.status=='infeasible':
14        l=x
15    else:
16        u=x
```

a,b value

```
In [10]: 1 a.value
```

```
Out[10]: array([1.99122852, 5.2222467 , 4.65128738, 2.73670298, 0.49617771,
0.94320976, 1.64683605, 1.42963531, 0.63846494, 0.10660798])
```

```
In [11]: 1 b.value
```

```
Out[11]: array([2.66101167, 4.6054937 , 2.50193462, 0.77319206, 0.68437213,
0.66483055, 0.30912078, 0.05180544])
```

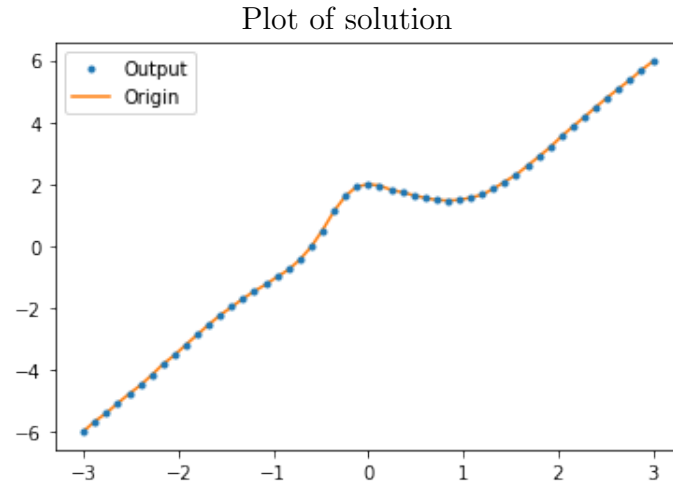
The function  $p(x)$  is

$$p(x) = 1.99 + 5.22x + 4.65x^2 + 2.74x^3 + 0.5x^4 + 0.94x^5 + 1.65x^6 + 1.43x^7 + 0.64x^8 + 0.11x^9$$

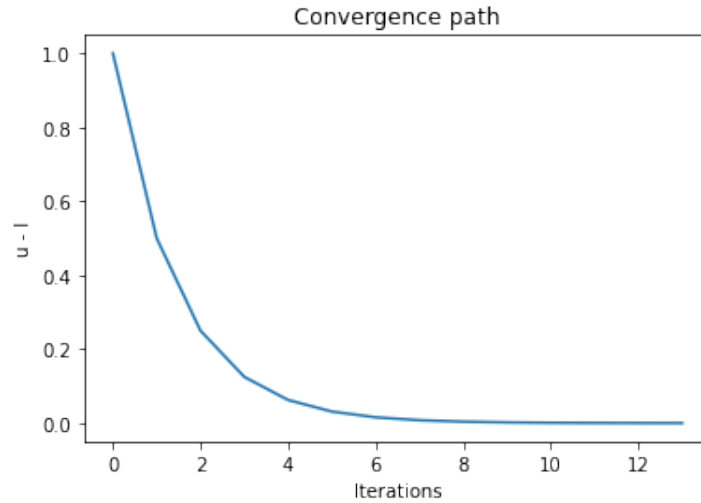
The functions  $q(x)$  is

$$q(x) = 1 + 2.66x + 4.61x^2 + 2.5x^3 + 0.77x^4 + 0.68x^5 + 0.66x^6 + 0.31x^7 + 0.05x^8$$

The final error is 0.017758482837463507.



To show the convergence, I plot the difference between the upper bound and the lower bound throughout the iterations.



We can easily see that  $u - l$  decreases by  $1/2$  every iteration.

(c)



Similar to (a), we could prove that programming

$$\begin{aligned} \min_{a, b} \quad & \max_i \left| \frac{p(t_i)}{q(t_i)} - y_i \right| \\ \text{s.t.} \quad & \|a\|_1 \leq M_1, \\ & \|b\|_1 \leq M_2 \end{aligned}$$

is a quasiconvex problem. With  $\phi_\alpha(a, b) = \max_i |p(t_i) - y_i q(t_i)| - \alpha q(t_i)$ . Then we could do the bisection for  $\alpha$ ,  $M_1$  and  $M_2$  to gain a sparse solution.