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

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

(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 pth order polynomial function, $\|\cdot\|_p$ should be used; for other loss functions, we could define corresponding norms.

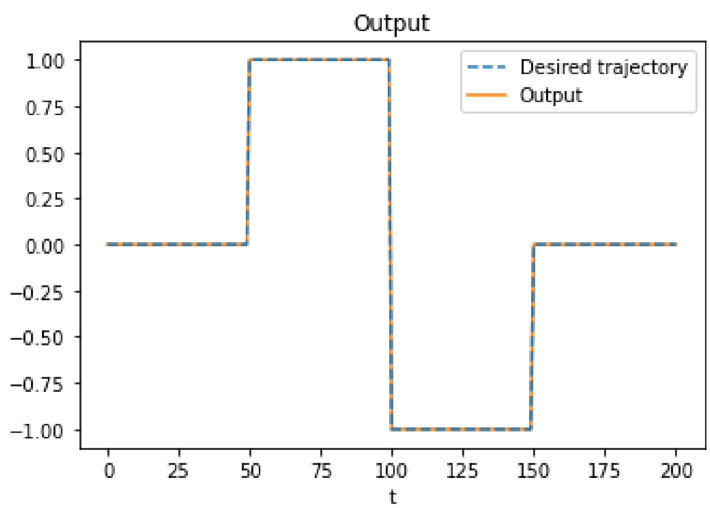
In [2]:
```
y_des_1b = np.load('y_des_1b.npz')
y1 = y_des_1b['y_des'].flatten()
```

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

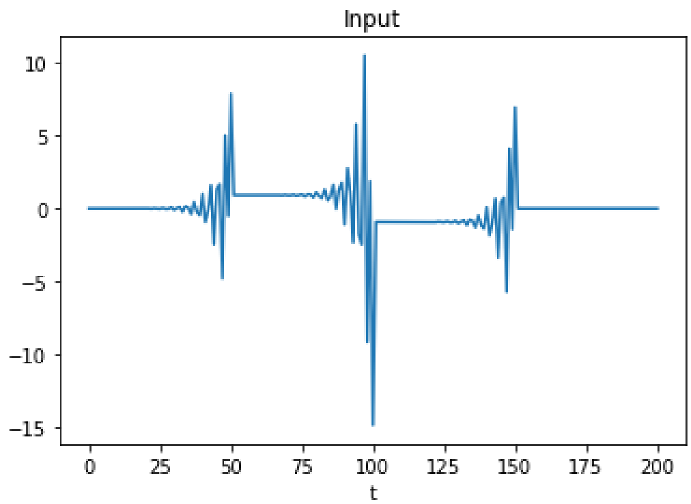
In [18]:
```
x = cp.Variable(N)
constraints = []
prob = cp.Problem(cp.Minimize(cp.norm(A@x-y1)), constraints)
prob.solve()
```

Out[18]: `9.095882644690729e-08`

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

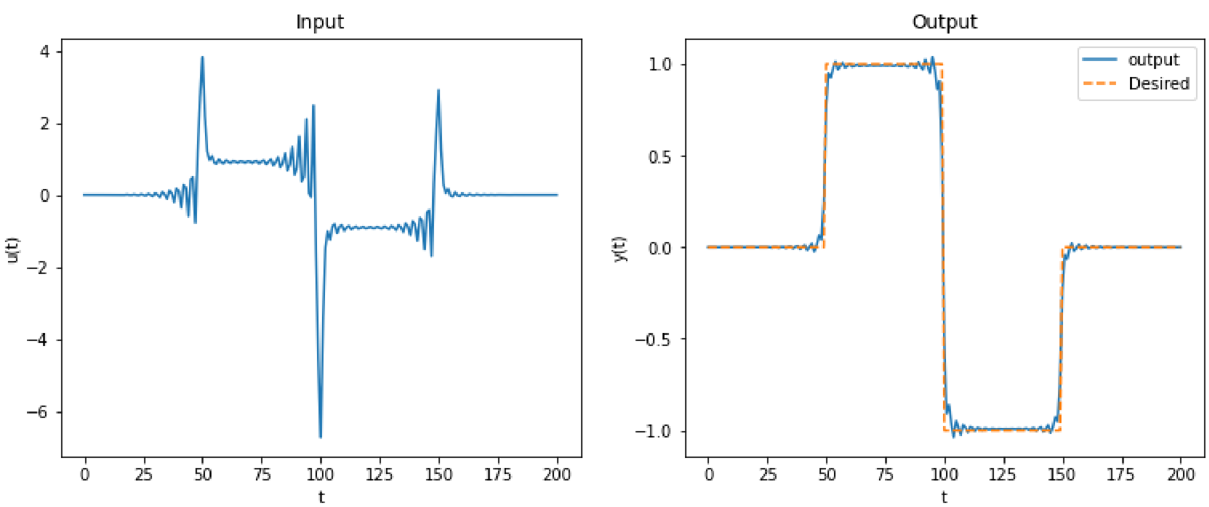


Our input looks like below.

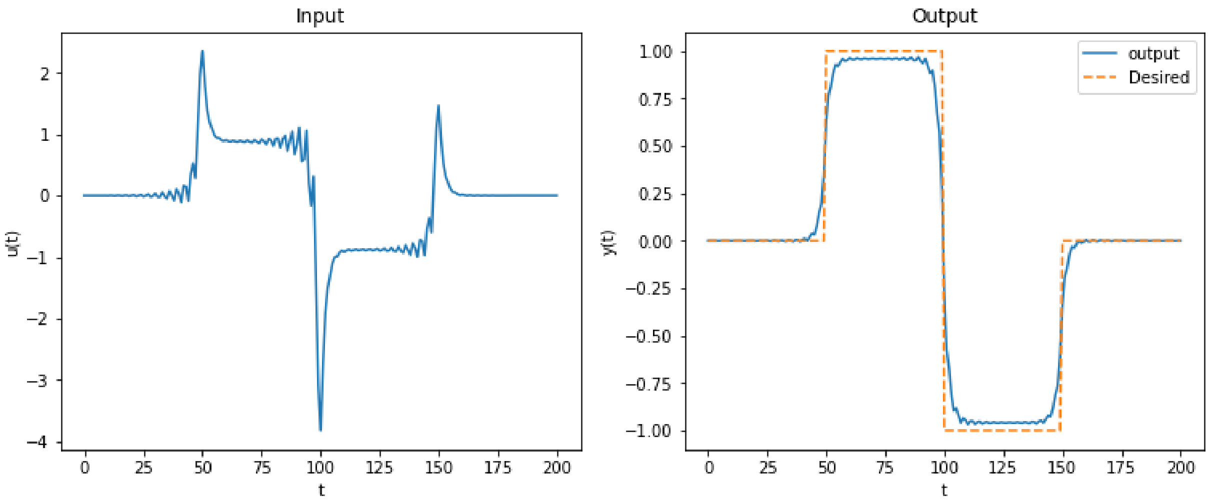


(c)

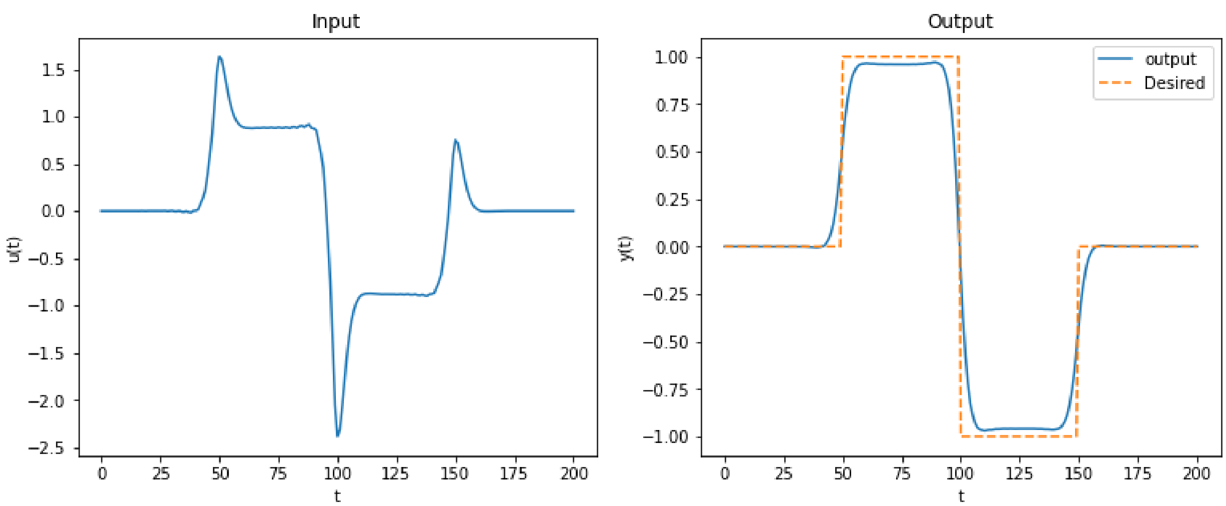
$\delta = 0$, $\eta = 0.01$



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



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



(d)

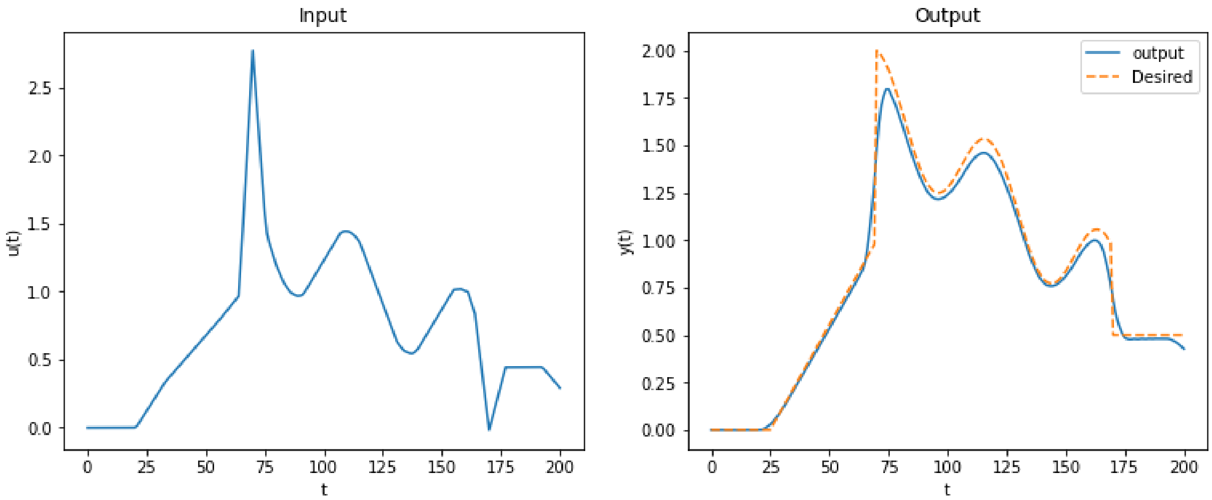
To encourage piecewise-linear input, we could change the smoothness loss part into second-order variation.

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.

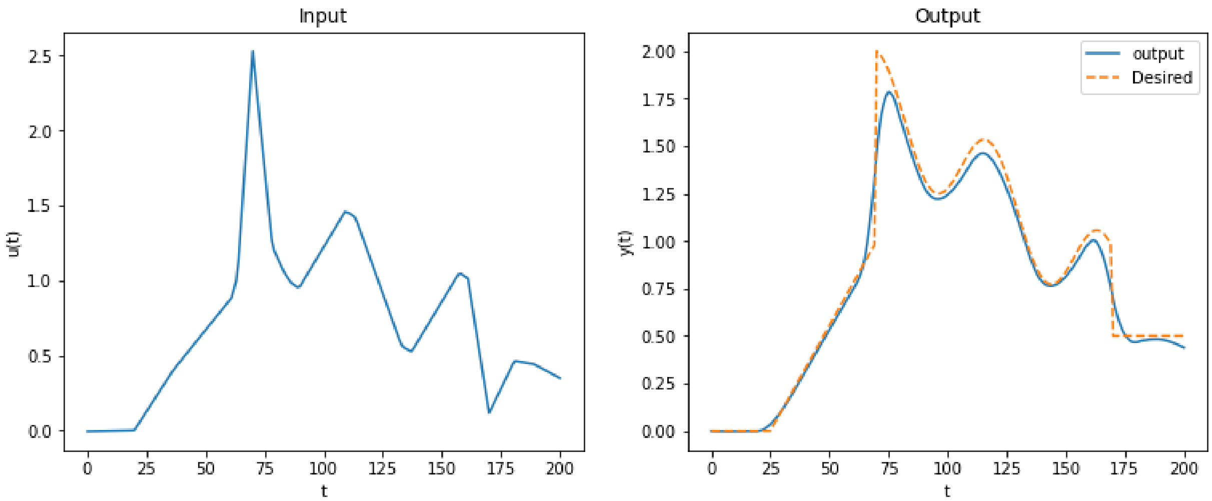
What's more, instead of encouraging small variation, we want 0 second-order variation, which means absolute value function would perform better. Therefore, we should modify J_{der} as $J_{der}\frac{1}{N-1}\sum_{t=1}^{N-1}|2u(t) - u(t+1) - u(t-1)|$.

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

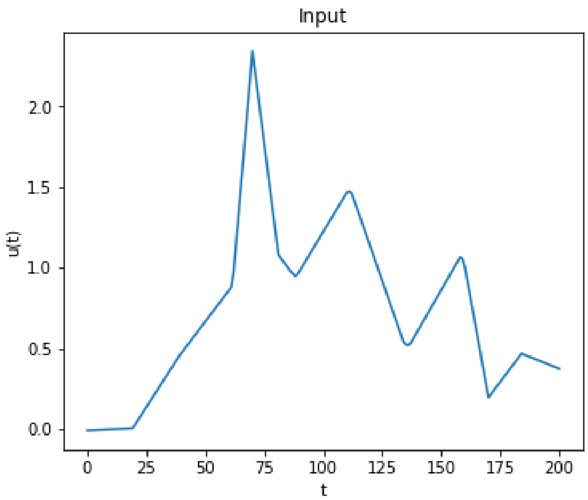
$\delta = 0.1,\ \eta = 0.05$

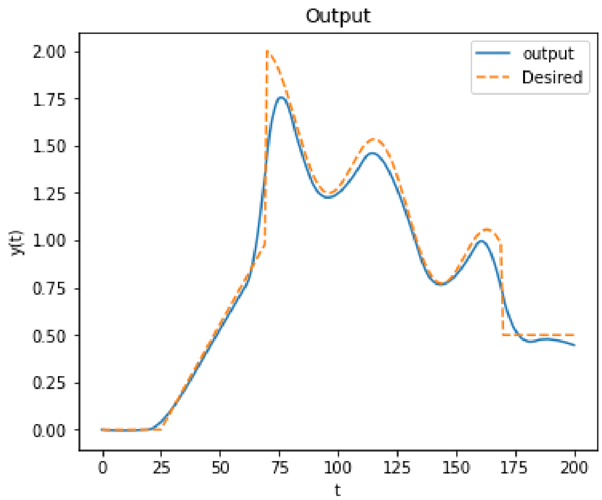


$\delta = 0.3,\ \eta = 0.05$



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





Question 2

(a)

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

First notice that D is convex, for any two points (x,y) and (m,n) in D, we have $\forall t \in [\alpha,\beta]$, $1+y_1t+\cdots+y_nt^n > 0$ and $1+n_1t+\cdots+n_nt^n > 0$ and thus for $0 \le \theta \le 1$, $1+(\theta y_1+(1-\theta)n_1)t+\cdots+(\theta y_n+(1-\theta)n_n)t^n > 0$, which means $(\theta x+(1-\theta)m, \theta y+(1-\theta)n) \in D$. So S_α is convex if $\{(a,b)\,|\,f(a,b) \le \alpha\}$ is convex. With $A_{\alpha,i} = \{(a,b)\,|\,q(t_i)(y_i-\alpha) \le p(t_i) \le q(t_i)(y_i+\alpha)\}$, we have

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

With $p_1(t) = x_0+\cdots+x_mt^m$, $p_2(t) = m_0+\cdots+m_mt^m$, $q_1(t) = 1+y_1t+\cdots+y_nt^n$, $q_2(t) = 1+n_1t+\cdots+n_nt^n$, $(\theta p_1+(1-\theta)p_2)(t) = (\theta x_0+(1-\theta)m_0)+\cdots+(\theta x_m+(1-\theta)m_m)t^m$ and $(\theta q_1+(1-\theta)q_2)(t) = (\theta+(1-\theta))+\cdots+(\theta y_n+(1-\theta)n_n)t^n$, clearly we could have

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

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_α is convex, and the objective functions is a quasiconvex function. The problem is quasiconvex programming.
(b)

Python code

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

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

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

a,b value

In [10]:
```
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]:
```
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

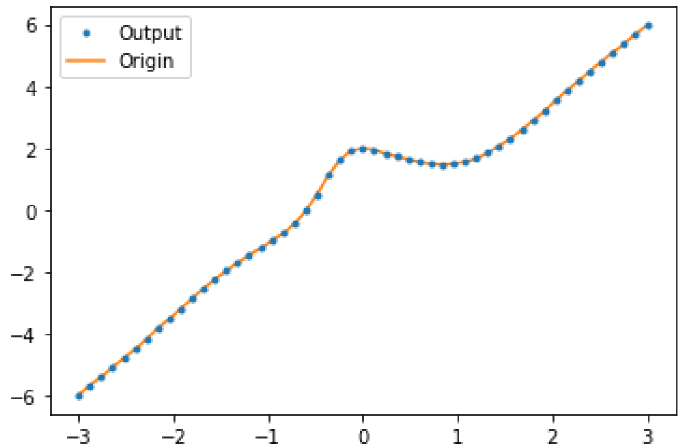
$$p(x) = 1.99 + 5.22x + 4.65x^2 + 2.74x^3 + 0.5x^4 + 0.94x^5 + 1.65x^6 + 1.43x^7 + 0.64x^8 + 0.11x^9$$

The functions $q(x)$ is

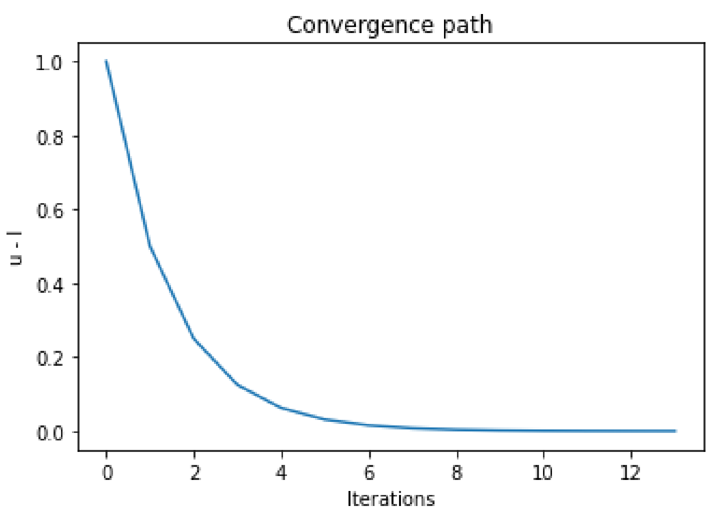
$$q(x) = 1 + 2.66x + 4.61x^2 + 2.5x^3 + 0.77x^4 + 0.68x^5 + 0.66x^6 + 0.31x^7 + 0.05x^8$$

The final error is 0.017758482837463507.

Plot of solution



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



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

Similar to (a), we could prove that programming

$$\begin{array}{ll} \min\limits_{a,b} & \max\limits_i |\frac{p(t_i)}{q(t_i)} - y_i| \\ \text{s.t.} & \|a\|_1 \leq M_1, \\ & \|b\|_1 \leq M_2 \end{array}$$

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 α, M_1 and M_2 to gain a sparse solution. For convenience, I take $M_1 = M_2$. So we could minimize the M_1 and M_2 for a given error, which is equivalent to add a 1-norm regularizer to the objective function. So here I used bisection with respect to M for a series of error.

Python code

```
def fc(alpha,M):
    a = cp.Variable(10)
    b = cp.Variable(8)
    constraints = [phi(a,b,alpha)<=0] + [cp.norm(a,1)<=M] + [cp.norm(b,1)<=M]
    prob = cp.Problem(cp.Minimize(1), constraints)
    prob.solve()
    temp=0,0
    if prob.status!='infeasible':
        temp = sum(np.abs(a.value)>0.01),sum(np.abs(b.value)>0.01)
    return prob.status!='infeasible',temp,a.value,b.value
```

```
def bisec(alpha,epsilon):
    lM=0
    uM=20
    while uM-lM>epsilon:
        x=(uM+lM)/2
        temp=fc(alpha,x)
        if temp[0]:
            uM=x
        else:
            lM=x
    temp=fc(alpha,uM)
    return uM,temp[1][0],temp[1][1],temp[2],temp[3]
```

```
for i in np.linspace(0.018,0.03,13):
    temp=bisec(i,0.1)
    print('Error:',np.around(i,3),'M:',temp[0],'L0 norm of a,b:',temp[1],temp[2])
```

Output for bisection

```
Error: 0.018 M: 17.109375 L0 norm of a,b: 10 8
Error: 0.019 M: 9.0625 L0 norm of a,b: 9 8
Error: 0.02 M: 6.953125 L0 norm of a,b: 8 8
Error: 0.021 M: 6.5625 L0 norm of a,b: 5 6
Error: 0.022 M: 6.484375 L0 norm of a,b: 3 6
Error: 0.023 M: 6.484375 L0 norm of a,b: 5 6
Error: 0.024 M: 6.40625 L0 norm of a,b: 5 6
Error: 0.025 M: 6.328125 L0 norm of a,b: 4 5
Error: 0.026 M: 6.25 L0 norm of a,b: 3 6
Error: 0.027 M: 6.25 L0 norm of a,b: 4 7
Error: 0.028 M: 6.171875 L0 norm of a,b: 4 7
Error: 0.029 M: 6.171875 L0 norm of a,b: 4 7
Error: 0.03 M: 6.09375 L0 norm of a,b: 4 7
```

It is hard to gain a real 0 entry for either a or b, so here I count the L_0 norm with discarding all entries whose absolute value are less than 0.01. We could gain a rather sparse solution for error 0.022. With higher tolerance, the solution become more sparse.

a,b value

```
In [154]:  1 temp[3]

Out[154]: array([ 1.97605672e+00,  3.12770846e+00, -3.87534857e-04,  9.36669448e-04,
                  1.65674469e-04,  1.36790300e+00,  2.33412438e-04,  8.34545888e-04,
                 -2.42492129e-04,  7.18210840e-03])

In [155]:  1 temp[4]

Out[155]: array([ 1.57044485e+00,  2.11483632e+00, -6.84224834e-01,  1.35898821e-01,
                  1.12761862e-01,  6.12841326e-02, -6.24531942e-03,  8.26926922e-04])
```

The error is 0.021942468198457465.
The function $p(x)$ is

$$p(x) = 1.98 + 3.13x + 1.37x^5$$

The function $q(x)$ is

$$q(x) = 1 + 1.57x + 2.11x^2 - 0.68x^3 + 0.14x^4 + 0.11x^5 + 0.06x^6$$