EML 6351 Simulation Project 2

- README

All my codes are inside the **src** folder.

- For assignment 1, the code for implementing a standard gradient based adaptive update law is inside the ***traditional.m*** file.
- For assignment 2, the code for implementing a concurrent learning based adaptive update law is inside the ***CLmethod.m*** file.
- For assignment 3, I designed 2 method to implement the intergral concurrent learning based adaptive update law. The codes are inside the ***ICLmethod.m*** and ***ICLmethod2.m*** file. These two files are independent and both can be implemented in MATLAB directly.

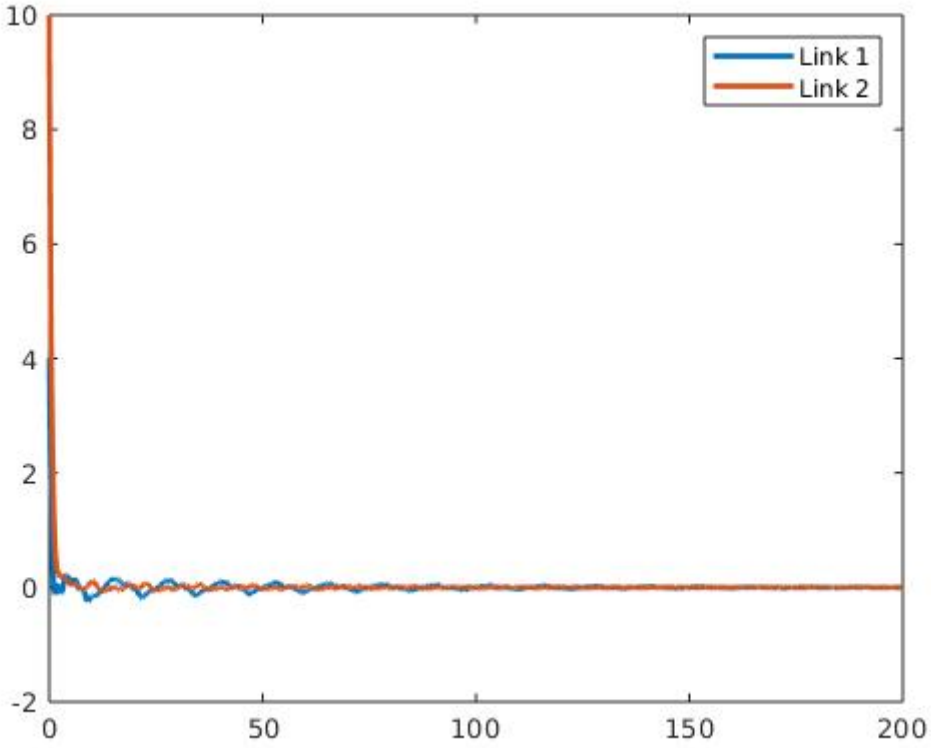
- Discussion
- **(a) Simulation Section**
  - **Standard Gradient Based Adaptive Update Law**

    1. Control gains.

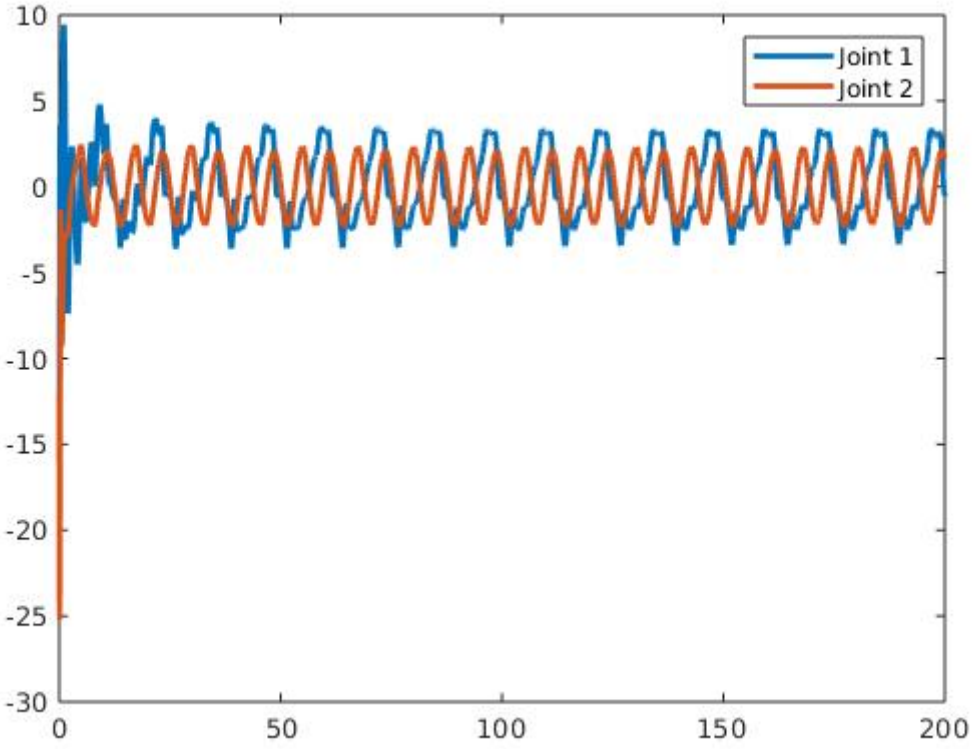
    k = 5; α = 2;

    $$\gamma = \begin{bmatrix} 5 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

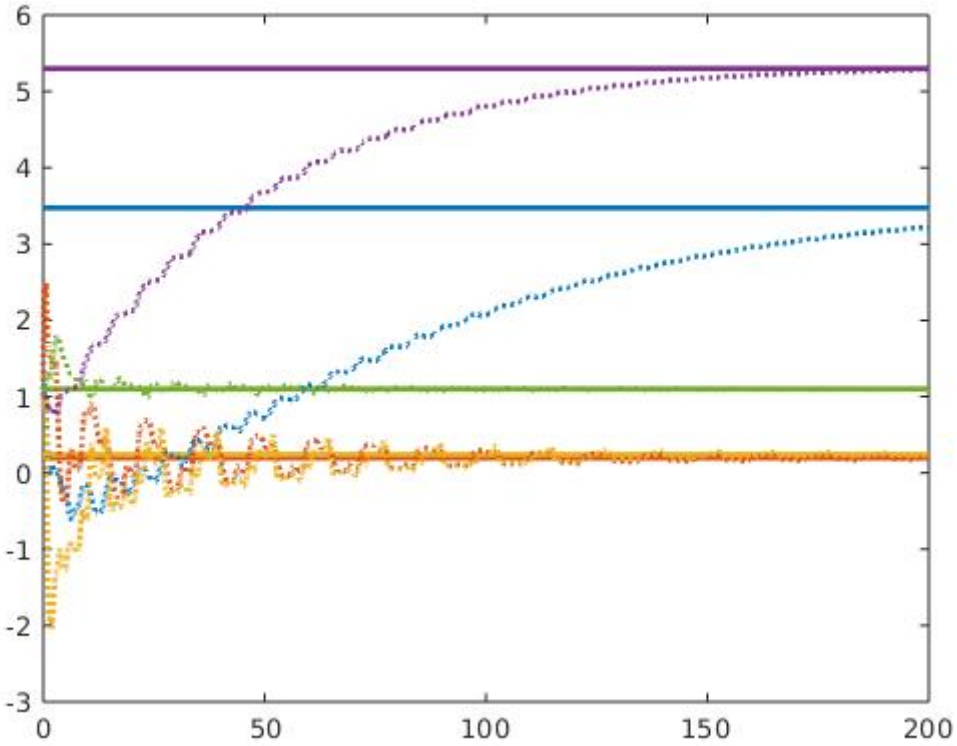
    2. Tracking error plot for each link.



3. Control input plot for each link.



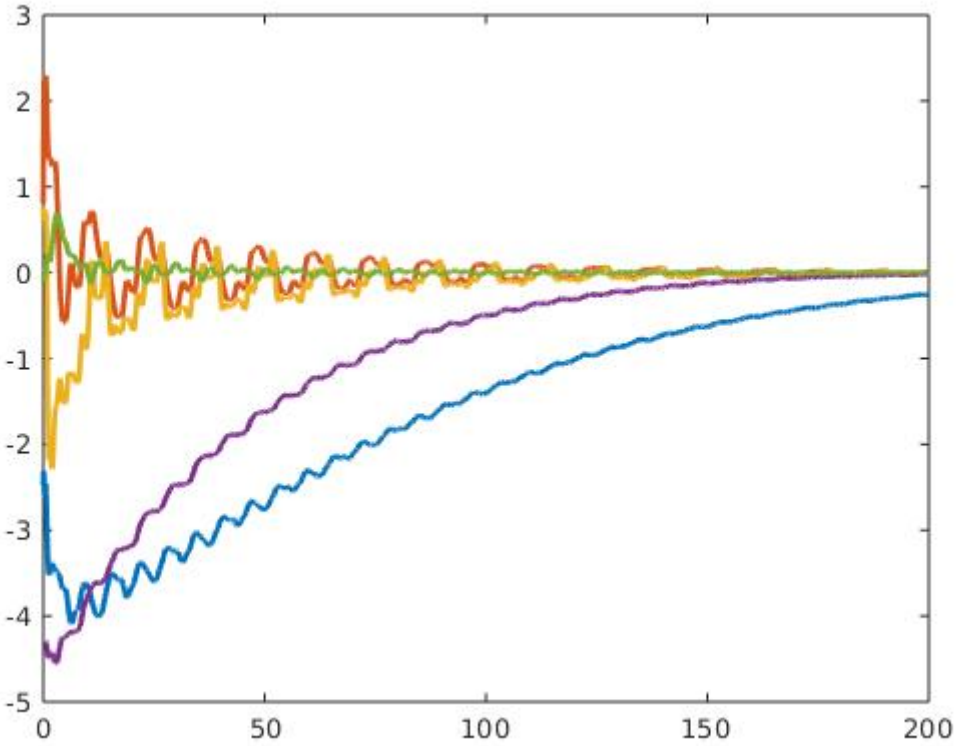
4. Plot of the adaptive estimates.



5. Plot of the minimum eigenvalue of the Y^TY summation.

In this method, we don't compute eigenvalue for update law.

6. Plot of the parameter estimate errors.



- **concurrent learning based adaptive update law**

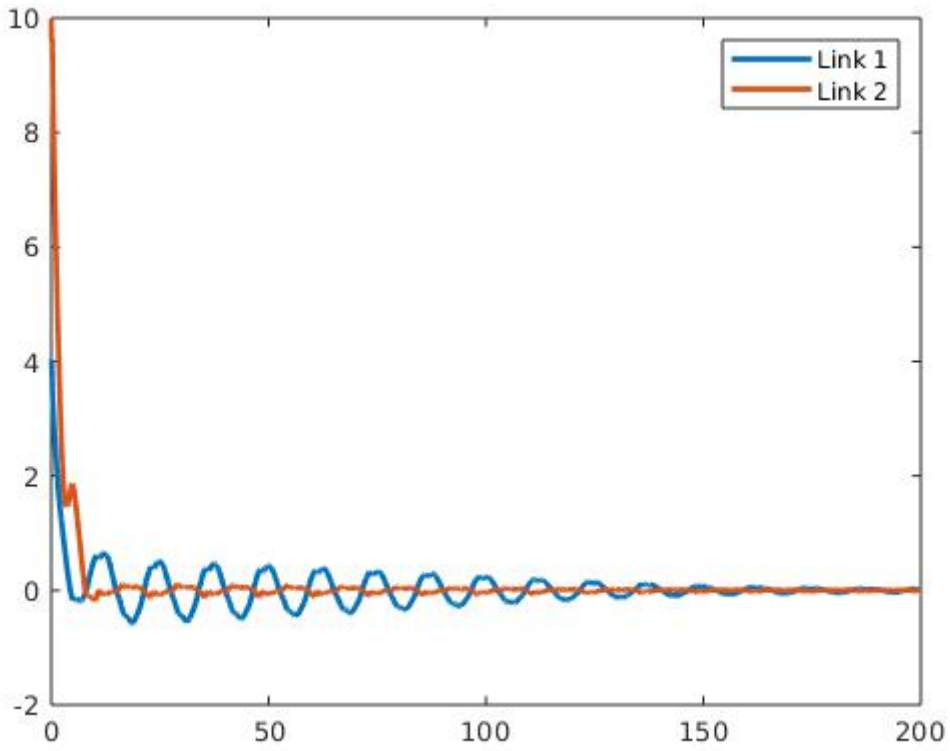
1. Control gains.

k = 1; kcl = 10e-5; α = 2;

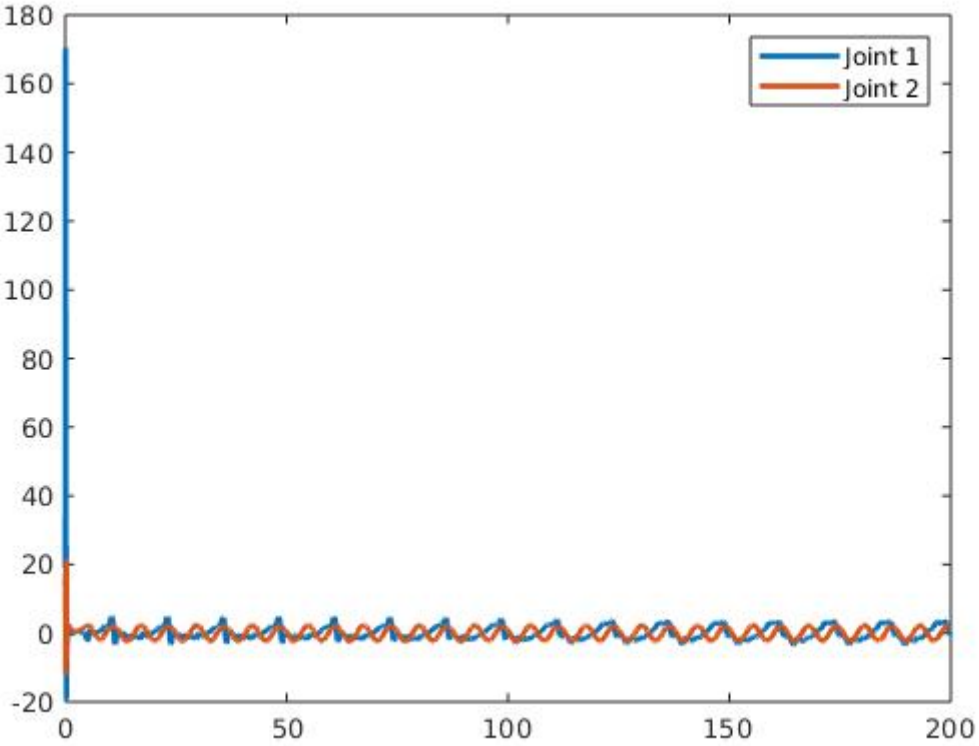
$$\gamma = \begin{bmatrix} 5 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

And I set λ =1, as it will affect the simulation's performance.

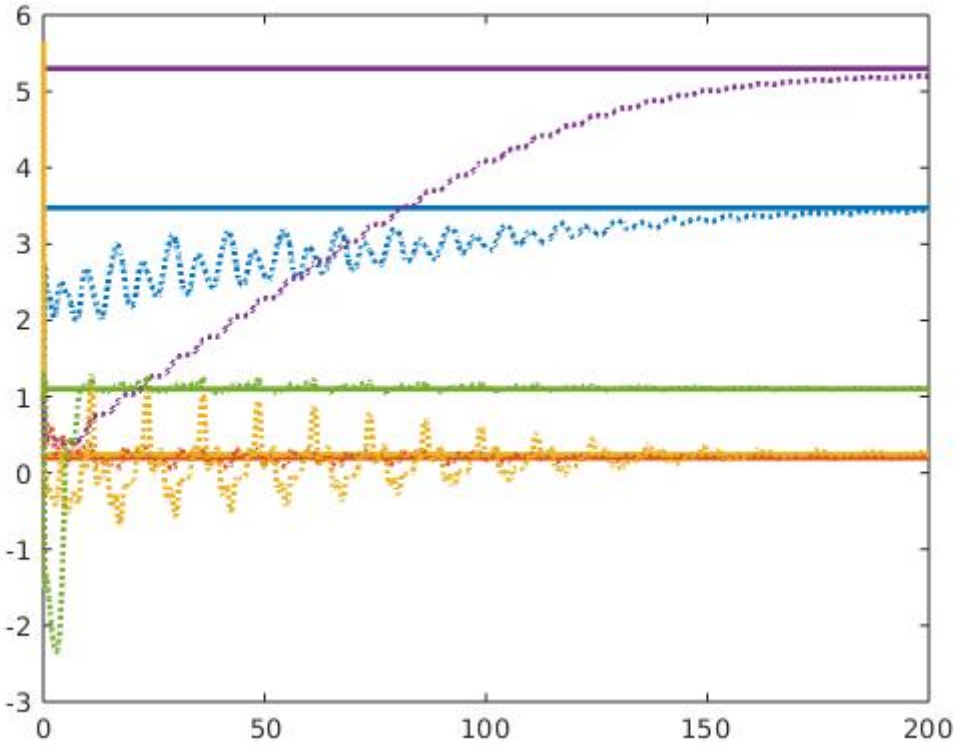
2. Tracking error plot for each link.



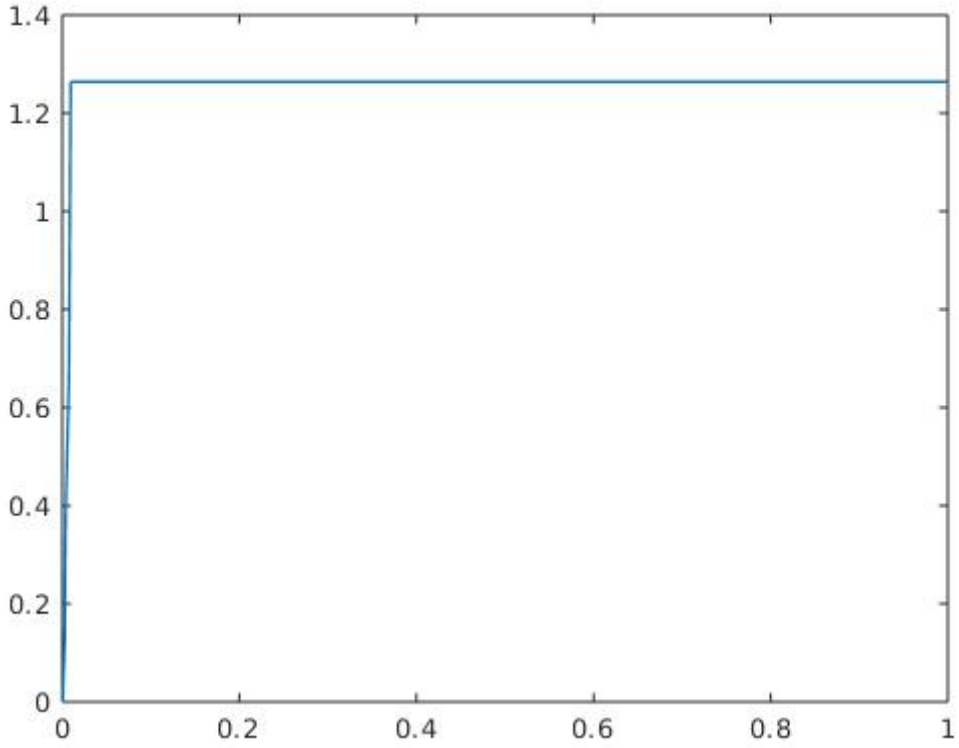
3. Control input plot for each link.



4. Plot of the adaptive estimates.

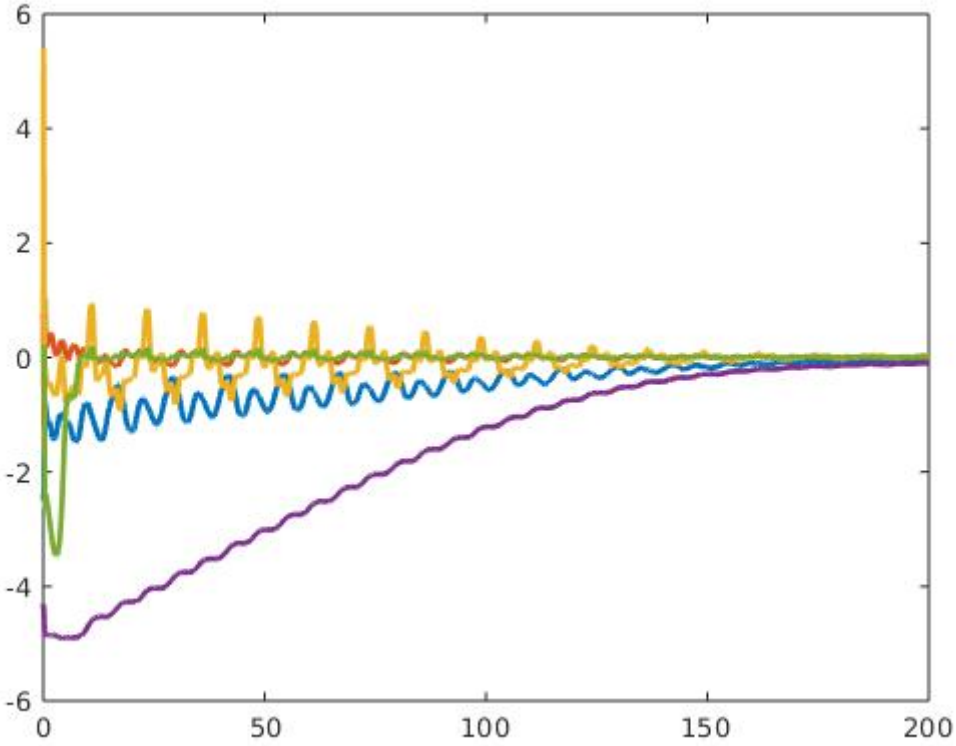


5. Plot of the minimum eigenvalue of the Y^TY summation.



Once our ODE solver get the eigenvalue of the Y^TY summation bigger than λ , we stop updating our Y1i history list thus it will remain till the simulation stops.

6. Plot of the parameter estimate errors.



- **Integral concurrent learning based adaptive update law**

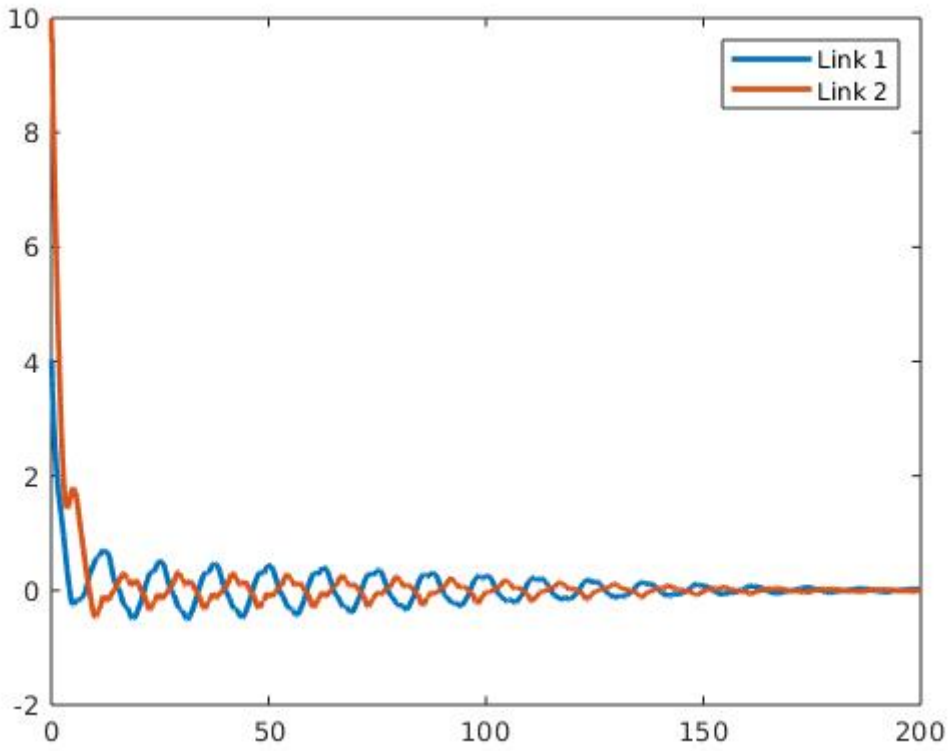
1. Control gains.

k = 1; kcl = 10e-5; α = 2;

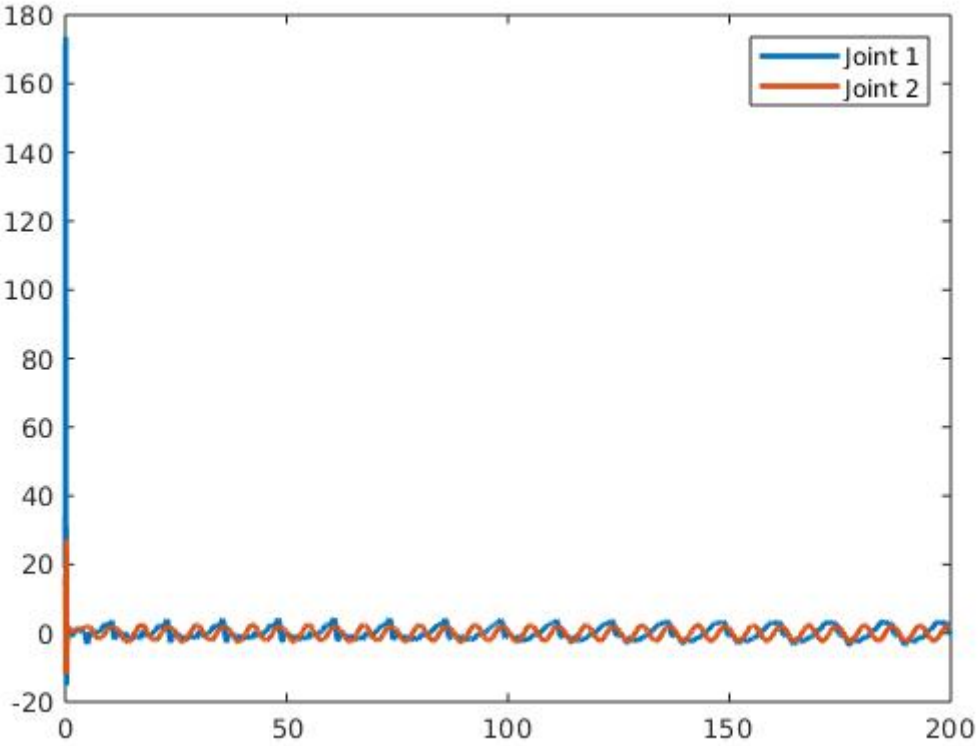
$$\gamma = \begin{bmatrix} 5 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

And I set λ =1, as it will affect the simulation's performance.

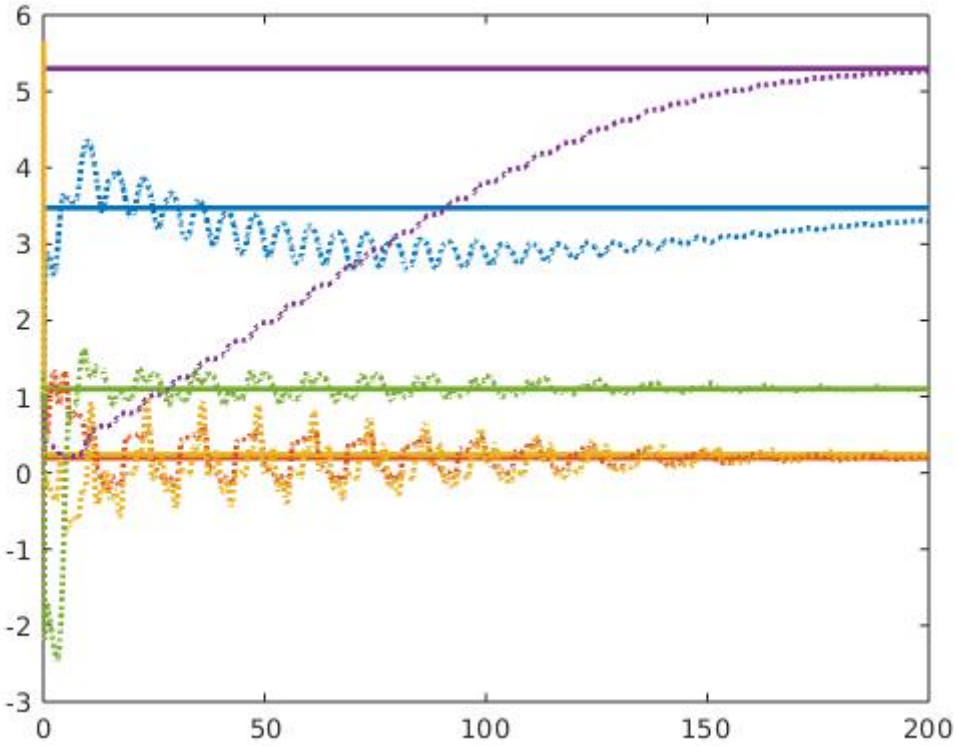
2. Tracking error plot for each link.



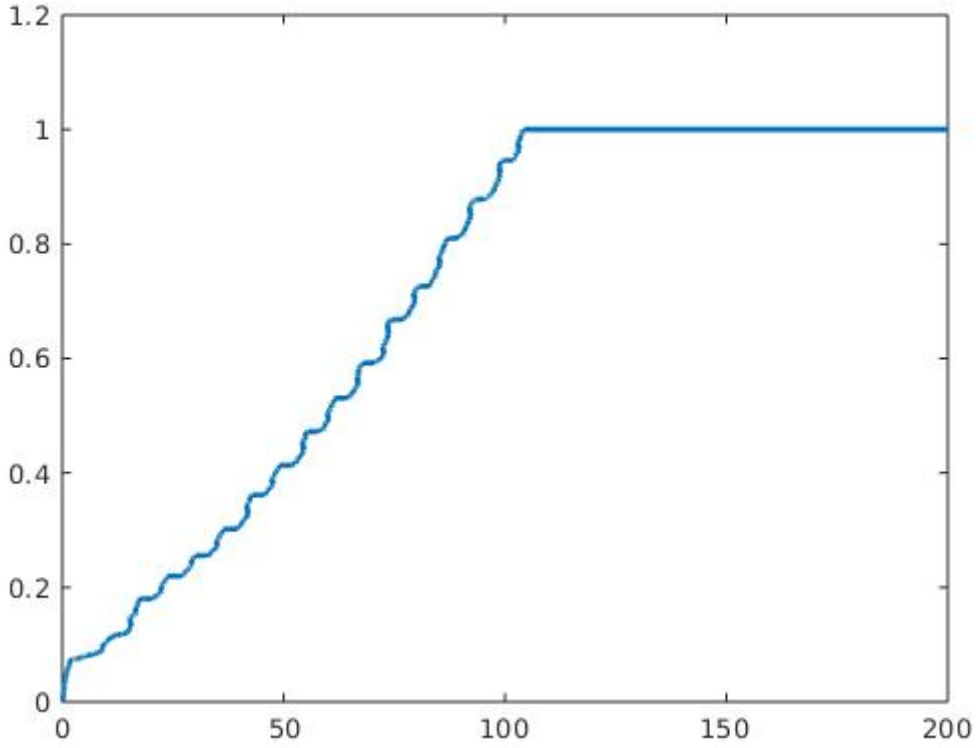
3. Control input plot for each link.



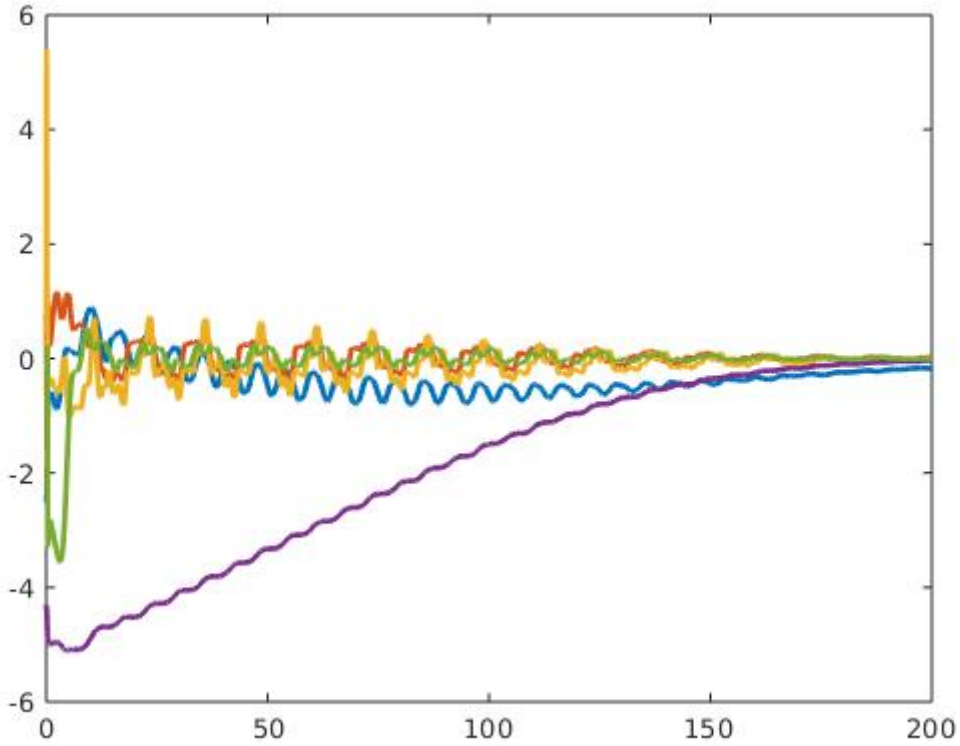
4. Plot of the adaptive estimates.



5. Plot of the minimum eigenvalue of the Y^TY summation.



6. Plot of the parameter estimate errors.

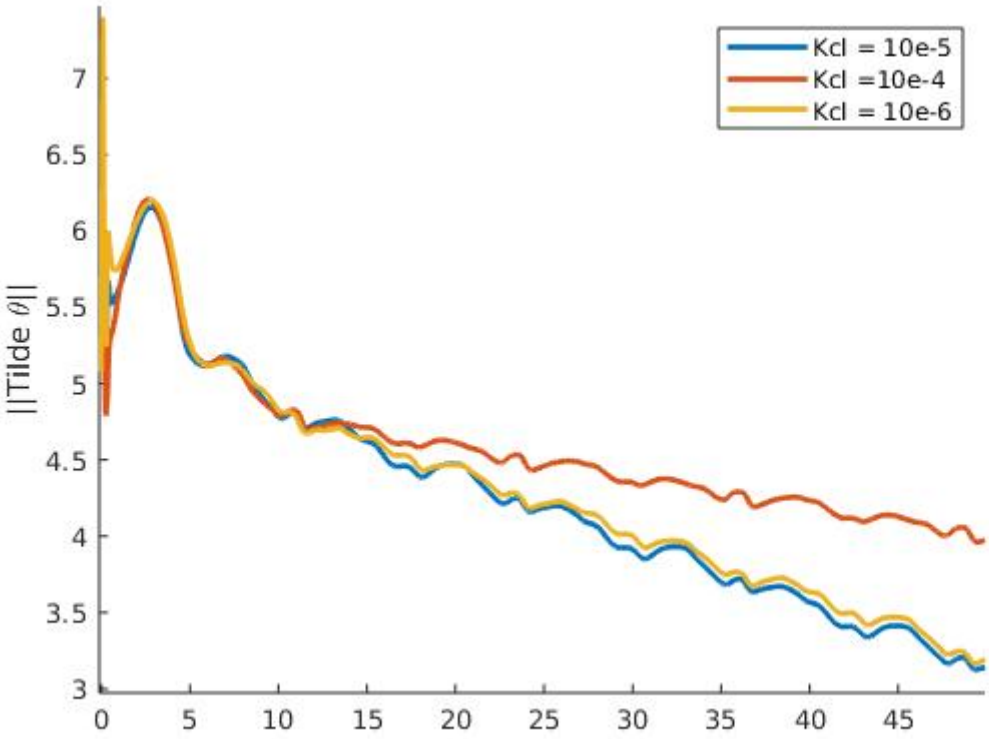


- **(b) Discussion section**
  - **Differences in tuning the control gains/adaptations**

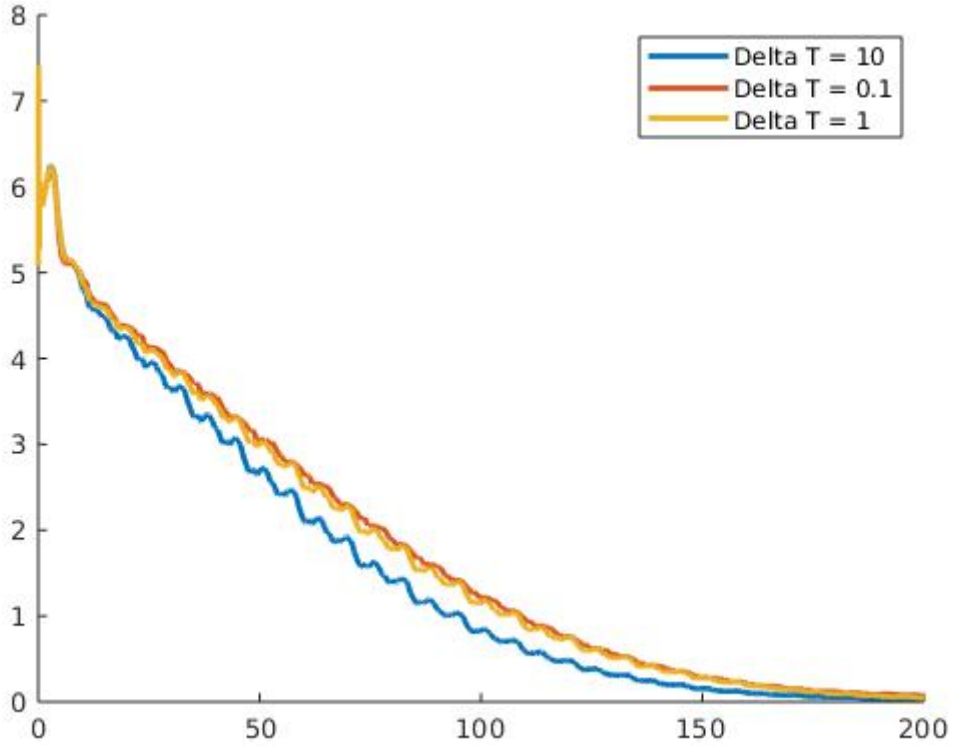
From the analysis of lyapunov function, we can know that K (Kcl,Kicl) and α can both affect the Convergence of the lyapunov function V. If $\dot{V}$ is semi negative definite, those parameters can still affect the convergence of the tracking error.

In my simulation, I found that increasing K or α can both make tracking error decrease faster, but increase the input as tradeoff, and in addition, when tracking error decrease faster, the convergence of the parameters estimate error will slow down. To get a better performance, I think we have to make K bigger than α. As, α would bring in oscillation in tracking error, and K can suppress the oscillation.

For parameter Kcl or Kicl, we have to set them properly in CL or ICL controllers, the parameters estimate error would converge when Kcl or Kicl are set properly. For my implementation, I found when I set Kcl to 10e-5, I can get a better convergence in parameters estimate error, and when Kicl is bellow 10e-7, I can get the best performance in the ICL simulation scenario. Below is a example of tunning Kcl.



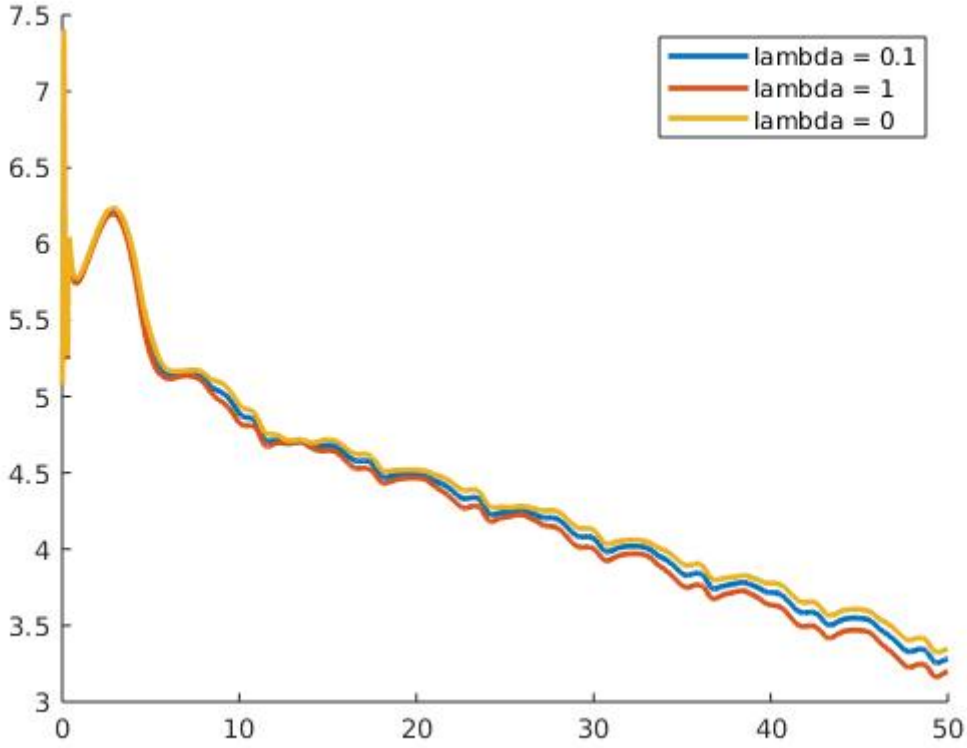
For comparing userdefined Delta T, I found it very difficult to implement in ODE45 solver, so I designed anouther method to compare them, and the results are as below:



I think increasing Delta T can generally make better performance, as it can supress the noise, but we have to tune the Kicl at the same time in case that our Y^TY summation become too large.

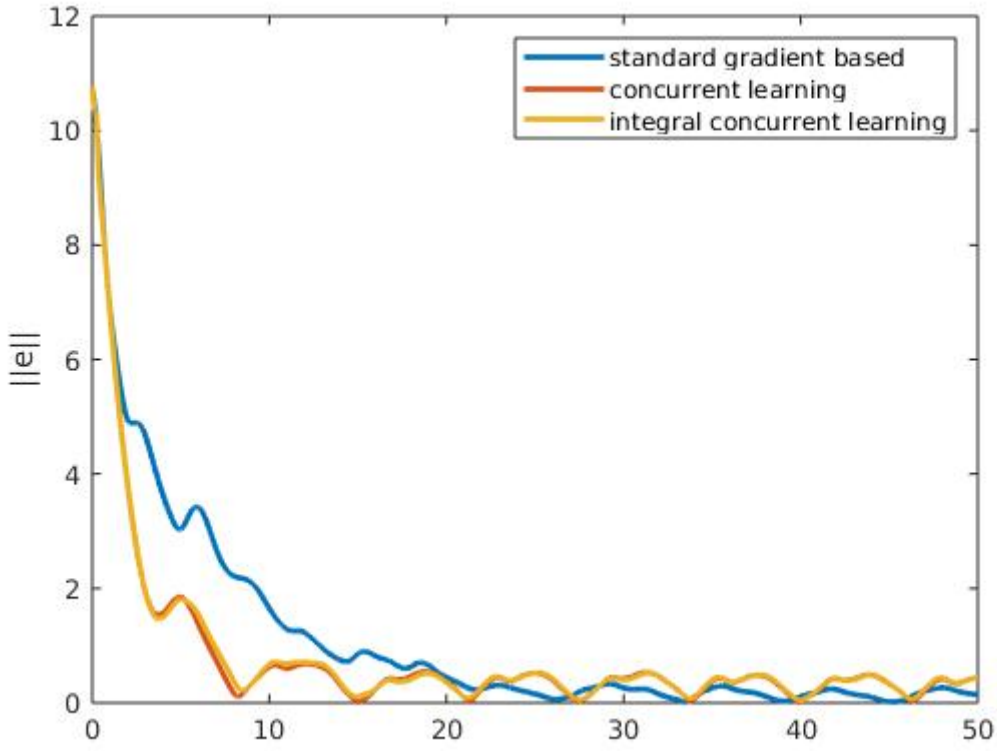
For γ, it can control the speed of the parameter estimate error convergence. In general, we can increase them to get faster parameter estimate convergence, however, in my opinion, I think we should just tune the diagonal element γ, because it's risky to add covariance in those parameter estimate.

For λ, I think with bigger λ, we can have better convergence in parameter estimate error. Below is the comparison for scenario with different λ but the same Kci:



- **Performance of the tracking error for each controller**

  I compared the $\|e\|$ generated from the 3 simulation using the 3 different controllers. The result is as below:



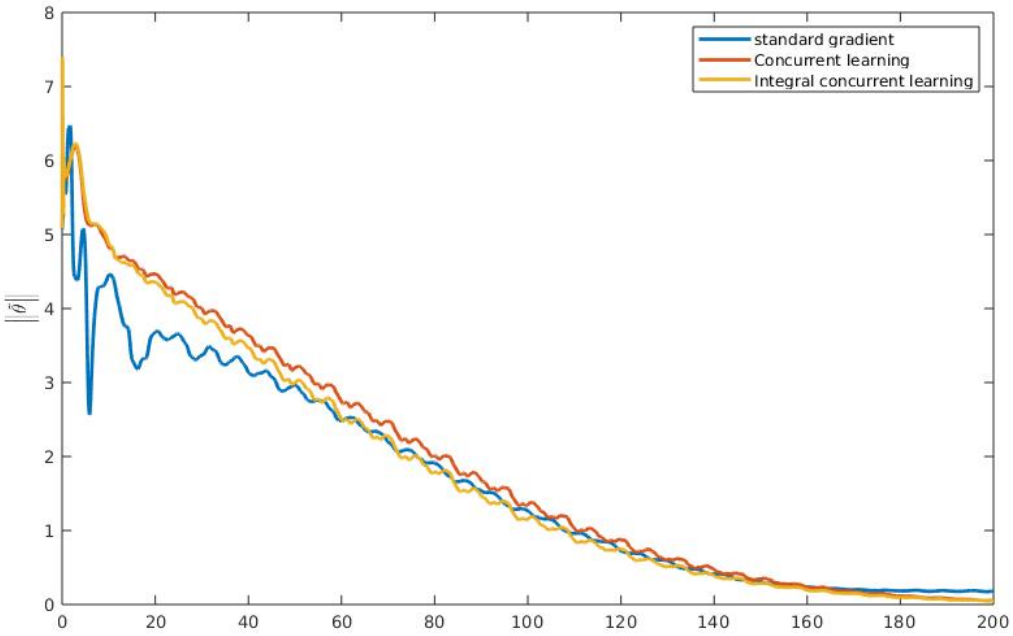
From the aespect of $\|e\|$, I can not conclude that which controller is better. concurrent learning method and integral concurrent learning has the very same performance of the tracking error. I

think that's because they have the very same structure of $\dot{\hat{\theta}}$ and the same τ .

From lyapunov analysis, we can infer that all the three $\|e\|$ would asymptotically converge to 0. So the results from the simulation are as expected.

- **Performance of the adaptation for each case**

  Comparing the $\left\|\tilde{\theta}\right\|$ form three simulation as below:



From lyapunov analysis, we only can infer that the estimate error of the standard gradient based adaptive update law is just bounded, but we can infer that the estimate errors of the other two method can exponentially converge to 0. At the beginning of the simulation, the estimate error of the standard gradient based adaptive update law outperfroms the other two methods, but at the end of the simulation, the performances of the CL and ICL method are better than the standard one. I think that's consistent with the lyapunov analysis, because it shows the error will converge to 0 in CL or ICL method.

However, I can't see much noise in the simulation of CL method, and the oscillation of estimate error behave very similar bettween ICL and CL method. I think that's because in my simulation, I only collect the first couple timesteps' data until we achieve our ideal minimum eigen value, then stop updating the datas. It makes us very hard to see the noisy signal. This is exactly what I want to improve in my simulation, the second thing I want to improve is the ODE method itself. Because I am using RK45 to simulate, that makes me very hard to collect exact timestep's data.