Problem 1a (Off-line gain tuning)

The best possible tracking performance might be achieved using $K \approx 8.0$, even though the squared error is minimized using $K \approx 12.0$. This is due to the fact that the system clearly oscillates with $K \approx 12.0$ which, on a real system, most likely will result in a worse tracking performance.

![Tracking performance with K=8.0](image)

Figure 1: Tracking performance using $K = 8.0$.

Problem 1b (Approaches to adaptive control)

While model reference adaptive controllers and self tuning regulators were introduced as different approaches, the only real difference between them is that model reference schemes are direct adaptive control, whereas self tuning regulators are indirect. The self tuning regulator first identifies the plant parameter through some fixed transformation. The model reference adaptive schemes update the controller parameters directly (no explicit estimate or identification of the plant parameters is made). The model reference adaptive schemes can be seen as a special case of the self tuning regulators, with an identity transformation between updated parameters and controller parameters.
Problem 1c (Inverse dynamics control law)

The inverse dynamic control command is computed from the assumed model according to

\[ \dot{x} = \hat{f}(x) + \hat{g}(x) u + K(x_d - x) \]  

(1)

\[ \dot{x} - \hat{f}(x) - K(x_d - x) = \hat{g}(x) u \]

(2)

where \( \hat{f}(x) \) and \( \hat{g}(x) \) use the estimated parameters \( \hat{w}_1 \) and \( \hat{w}_2 \). Since in the provided simulink skeleton, the proportional controller is taken care of outside of the adaptiveControl function, the inverse dynamic control command is computed according to

\[ u = \frac{\dot{x} - \hat{f}(x) - K(x_d - x)}{\hat{g}(x)} \]  

(3)
Problem 1d (i.) (Self-tuning regulator)

The recursive least squares (RLS) update equations are given by:

\[
P^{n+1} = \frac{1}{\lambda} \left( P^n - \frac{P^n x x^T P^n}{\lambda + x^T P^n x} \right) \tag{4}
\]

\[
w^{n+1} = w^n + P^{n+1} x (t - w^n x)^T, \tag{5}
\]

where \( x = [1, x_1, \ldots, x_n]^T \) is the input data, \( w = [w_0, w_1, \ldots, w_n]^T \) the parameter vector, \( P \) the inverse covariance matrix, \( \lambda \) the forgetting factor, and \( t \) the target. The RLS update equations compute the parameter vector \( w \) which minimizes the cost function

\[
J = \frac{1}{2} (t - X w)^T (t - X w), \tag{6}
\]

where

\[
t = \begin{bmatrix} t_1 \\ t_2 \\ \vdots \\ t_n \end{bmatrix}, \quad X = \begin{bmatrix} x_1^T \\ x_2^T \\ \vdots \\ x_n^T \end{bmatrix} \tag{7}
\]

after \( n \) data points \((x, t)\) have been added. If \( \lambda = 1 \) the result will be exactly the same as for normal least squares update. However, \( \lambda < 1 \) allows to forget data which might be useful in case the target changes over time. The inverse covariance matrix is usually initialized according to

\[
P_0 = \frac{1}{\gamma} I, \tag{8}
\]

where \( \gamma << 1 \), i.e. the parameters \( w_i \) are only autocorrelated. The parameter vector \( w \) is usually initialized with zero, however, in some cases it might be more useful to initialize them with a small value, e.g. \( 10^{-6} \).
Problem 1d (ii.) (Self-tuning regulator)

Here we assume that the dynamics of the system
\[ \dot{x} = f(x) + g(x) u \] (9)
can be modeled as a linear system, where
\[ f(x) = w_1 x \] (10)
\[ g(x) = w_2 . \] (11)

The self-tuning regulator (indirectly) estimates the parameters \( w_1 \) and \( w_2 \) on the fly which fit the (previously seen) data \( x = [x, u]^T \) according to the assumed model. Here, we make use of the RLS equations and set the previously perceived velocity of \( x \) as the target \( t \).

The resulting tracking performance as well as the implementation in Matlab assuming the linear system is provided below.

The tracking performance clearly outperforms the non-adaptive scheme from problem 1a.
function command=adaptiveControl(u)

% global variables needed to learn a controller and not to forget previously
% computed parameters
global P; global w;

% variables from previous time step (with some delay)
xd_prev = u(1); x_prev = u(2); u_prev = u(3);

% current desired state
x_des = u(4); xd_des = u(5);

% current time in simulation
t = u(6);

% here comes an initialization only triggered at the first time step
% of the simulation
I = eye(2); lambda = 1.0; ridge = 10^-6;

if t==0,
    P = I*(1/ridge); w = 0.01 * diag(I);
end

% here comes your implementation of the adaptive control law and
% the parameter updates
x = [x_prev; u_prev]; y = xd_prev;

P = (1/lambda) * (P - ( (P * x * x' * P) / (lambda + x' * P * x)) );
w = w + P * x * (y - w' * x)';

fx = w(1)*x_des;
gx = w(2);

% this should be replaced by something sensible.
if(gx<0.1)
    command = u_prev;
else
    command = (xd_des - fx) / gx;
end
Problem 1d (iii.) (Self-tuning regulator)

Here we assume that the dynamics of the system in (9) can be modeled as a non-linear system of the form

\[ f(x) = w_1 x + w_2 x^2 \]  
\[ g(x) = w_2 x \]  
\[ (12) \]
\[ (13) \]

Thus, the estimated parameters become

\[ w = \begin{bmatrix} \hat{w}_1 \\ \hat{w}_2 \\ \hat{w}_3 \\ \hat{w}_4 \end{bmatrix}, \text{ and } x = \begin{bmatrix} x \\ x^2 \\ 1 \\ x \end{bmatrix}. \]
\[ (14) \]

The resulting tracking performance as well as the implementation in Matlab assuming the nonlinear system is provided below.

![Tracking performance using the non-linear model](image1)

Figure 3: Tracking performance of the adaptive controller assuming the non-linear system.

An increased tracking performance can be achieved by setting the forgetting factor \( \lambda = 0.97 \) as it allows to “forget” bad initial conditions.

![Tracking performance using the non-linear model with lambda=0.97](image2)

Figure 4: Increased tracking performance of the adaptive controller assuming the non-linear system with using a forgetting factor \( \lambda = 0.97 \).
function command=adaptiveControl_nonlinear(u)

% global variables needed to learn a controller and not to forget previously
% computed parameters
global P; global w;

% variables from previous time step (with some delay)
x_d_prev = u(1); x_prev = u(2); u_prev = u(3);

% current desired state
x_d = u(4); x_d_des = u(5);

% current time in simulation
t = u(6);

% here comes an initialization only triggered at the first time step
% of the simulation
I = eye(4);
lambda = 1;
ridge = 10^-6;

if t==0,
    P = I*(1/ridge);
    w = 0.01 * diag(I);
end

% here comes your implementation of the adaptive control law and
% the parameter updates
x = [x_prev; x_prev*x_prev; u_prev; x_prev*u_prev];
y = x_d_prev;

P = (1/lambda) * (P - { (P * x * x' * P) / (lambda + x' * P * x) } );
w = w + P * x * (y - w' * x)';

fx = w(1)*x_d_des + w(2)*x_d_des^2;
gx = w(3) + w(4)*x_d;

% this should be replaced by something sensible.
if(gx<0.1)
    command = u_prev;
else
    command = (x_d_des - fx) / gx;
end
Problem 1e (Model-reference adaptive control)

Consider the following one dimensional nonlinear control system:

\[ \dot{x} = f(x) + g(x) u, \]

where

\[ f(x) = w_1 x \quad \text{and} \quad g(x) = w_2 x. \]

The parameters \( w_1 \) and \( w_2 \) are constant but unknown. The input \( u \) and state \( x \) are available from (delayed) measurements. The objective is to generate an adaptive law for on-line estimation of \( w_1 \) and \( w_2 \) using the observed signals \( u(t) \) and \( x(t) \), such that the resulting inverse dynamics control command enables the system to track the desired trajectory \( x_{\text{des}}(t) \) robustly. The estimate of \( w_1 \) and \( w_2 \) are referred to as \( \hat{w}_1 \) and \( \hat{w}_2 \), respectively. The system is modeled with an extra proportional controller according to

\[ \dot{x} = f(x) + g(x) u + Ke = w_1 x + w_2 x u + Ke, \]

where \( K \) is a positive proportional gain and

\[ e = x_{\text{des}} - x, \]

where \( x_{\text{des}} \) is the desired state and \( x \) the current state as measured by the system. Replacing the unknown parameters \( w_1 \) and \( w_2 \) with their estimates \( \hat{w}_1 \) and \( \hat{w}_2 \), we obtain \( \dot{x} \) according to

\[ \dot{x} = \hat{w}_1 x - \hat{w}_2 x u. \]

Thus, the estimated state \( \hat{x} \) is computed by integrating the the current state \( x \) with the predicted velocity \( \dot{x} \). The prediction or estimation error, which reflects the parameter uncertainty\(^1\), is formed as the difference between the estimated state \( \hat{x} \) and the true state \( x^* \) obtained by integrating Eq. (17) using the true parameters, i.e. \( e = \hat{x} - x^* \). Note the difference to Eq. (18). However, since the true parameters are unknown, we are not able to compute \( x^* \) in simulation, thus, we can only use it to derive the update rules which guarantee stability. Later, in simulation, we will replace the estimation error \( e = \hat{x} - x^* \) with the tracking error \( e = x_{\text{des}} - x \). The error dynamics which drives the adaptive law can be derived according to

\[ \dot{e} = \dot{\hat{x}} - \dot{x} = \hat{w}_1 x - \hat{w}_2 x u - (w_1 x + w_2 x u + Ke) = -Ke + (\hat{w}_1 - w_1) x + (\hat{w}_2 - w_2) x u = -Ke + \hat{w}_1 x + \hat{w}_2 x u = -Ke + \hat{w}^T \tilde{x}, \]

where

\[ \hat{w} = \begin{bmatrix} \hat{w}_1 \\ \hat{w}_2 \end{bmatrix} \quad \text{and} \quad \tilde{x} = \begin{bmatrix} x \\ xu \end{bmatrix}. \]

The error of the estimated parameters are denoted as \( \hat{w}_1 \) and \( \hat{w}_2 \), i.e., \( \hat{w}_1 = \hat{w}_1 - w_1 \) and \( \hat{w}_2 = \hat{w}_2 - w_2 \). To guarantee the stability of the system, we need to choose the update equations for \( \hat{w}_1 \) and \( \hat{w}_2 \) such that the time derivative of the Lyapunov function is negative. As candidate function we choose

\[ V = \frac{1}{2} e^2 + \frac{1}{2} \hat{w}^T \Gamma^{-1} \hat{w}, \]

\(^1\)because \( \hat{w}_1(t) \) and \( \hat{w}_2(t) \) are different from the true (constant) parameters \( w_1 \) and \( w_2 \).
where $\Gamma$ is a positive definite matrix. Thus, $V \geq 0$ and $V = 0$ only if $e = 0$ and $\dot{w} = 0$. The derivative with respect to time is

$$
\dot{V} = e \dot{e} + \dot{w}^T \Gamma^{-1} \dot{w} \\
= e \dot{e} + \dot{w}^T \Gamma^{-1} \dot{w} \\
= e (-K e + \dot{w}^T x) + \dot{w}^T \Gamma^{-1} \dot{w} \\
= -K e^2 + e \dot{w}^T x + \dot{w}^T \Gamma^{-1} \dot{w}.
$$

(23)

To guarantee stability of the system we need to ensure that $\dot{V} \leq 0$ by choosing an appropriate update equation for $\dot{w}$. Since $K e^2 \geq 0$, we choose $e \dot{w}^T x + \dot{w}^T \Gamma^{-1} \dot{w} \neq 0$, leading to

$$
0 = e \dot{w}^T x + \dot{w}^T \Gamma^{-1} \dot{w} = \dot{w}^T (x e + \Gamma^{-1} \dot{w}).
$$

(24)

Assuming that there is always an estimation error $\tilde{w} = [\tilde{w}_1, \tilde{w}_2]$, we need to choose $\dot{\tilde{w}}$ such that $x e + \Gamma^{-1} \dot{\tilde{w}} = 0$, which leads to

$$
\dot{\tilde{w}} = -\Gamma x e.
$$

(25)

The individual update rules for each parameter separately is given by

$$
\dot{\tilde{w}}_1 = -\Gamma_1 x e
\dot{\tilde{w}}_2 = -\Gamma_2 x u e,
$$

(26) (27)

where $\Gamma_1$ is the first row and $\Gamma_2$ the second row of the positive definite matrix $\Gamma$. The values of $\Gamma$ determine the learning rate of the adaptive system and need to be tuned accordingly.

The resulting tracking performance and implementation of these equations in Matlab which realize the model-reference adaptive controller in Matlab is given below.

![Tracking performance using MRAC](image)

Figure 5: Tracking performance of the model-reference adaptive controller (MRAC).
function command=adaptiveControl_mrac(u)

% global variables needed to learn a controller and not to forget
% previously computed parameters

5 global w;

% variables from previous time step (with some delay)
xd_prev = u(1); x_prev = u(2); u_prev = u(3);

10 % current desired state
x_des = u(4);

15 % current time in simulation
t = u(6);

% here comes an initialization only triggered at the first time step of the
% simulation
if t==0
    w = [0.0; 0.0];
    command = 0;
else
    dt = 0.002;
    Gamma = [4.0, 0.0; 0.0, 0.01];

    e = x_des - x_prev;

    w = w + dt * -Gamma * [e * x_prev; e * x_prev * u_prev];

    fx = w(1)*x_prev;
    gx = w(2)*x_prev;

    if(gx<0.1)
        gx=0.1;
    end

    xd_des = 0;
    command = (xd_des - fx) / gx;
end

Note, in the code above, the desired velocity is set to zero. The reason for that is because it is discontinuous
and causes a step input into the system, which makes it go unstable. The tracking performance of the
Model-reference adaptive controller can be increased by changing the gain position gain $K$. The result for
$K = 11.5$ is shown below.
Figure 6: Tracking performance of the model-reference adaptive controller using 11.5 as position gain.