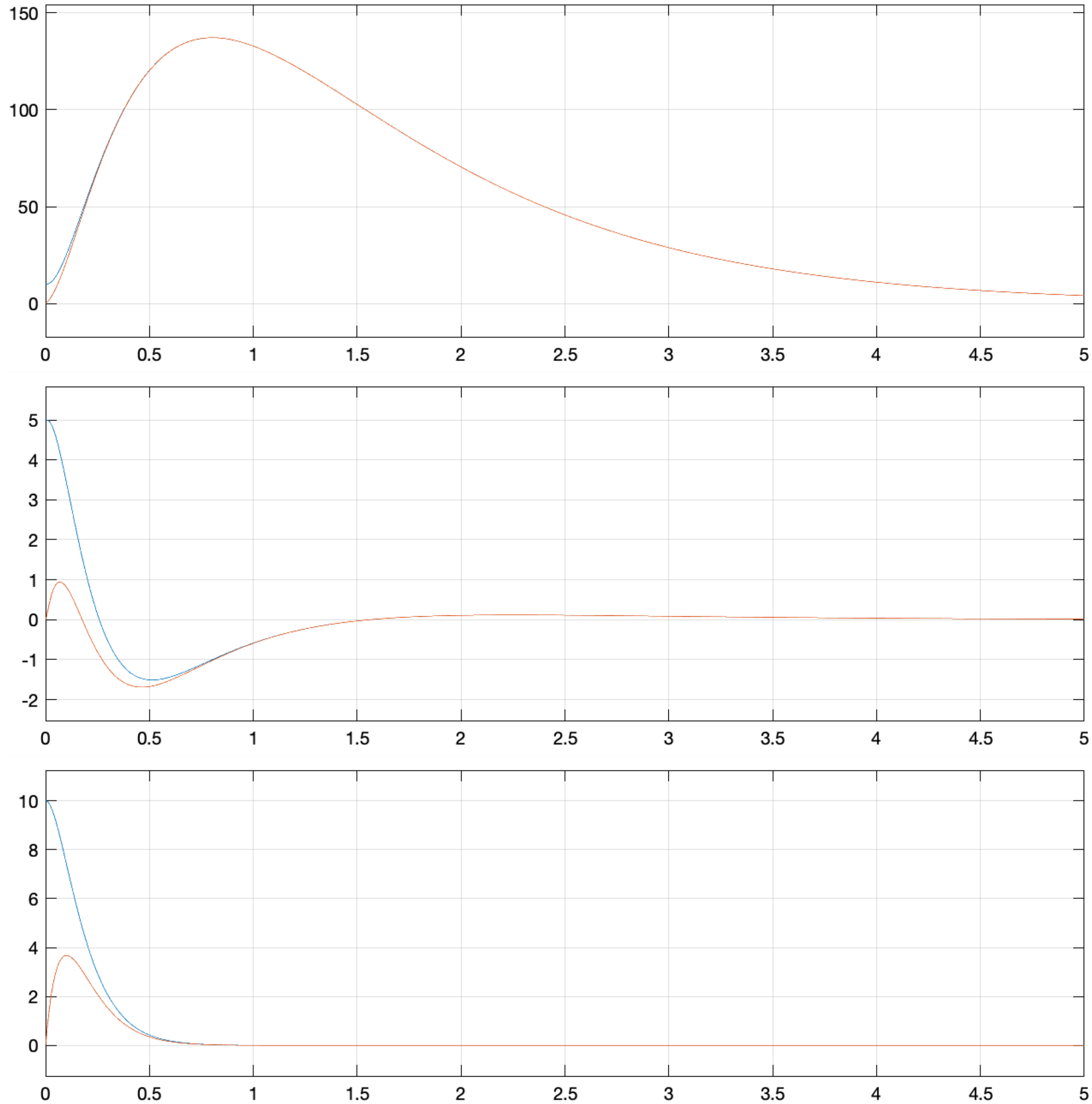
1. Using Pole Placement, try possible pole locations

```
K = place(A, B, [-1e1, -1e1, -1, -2, - 1e3, -1e3])
```



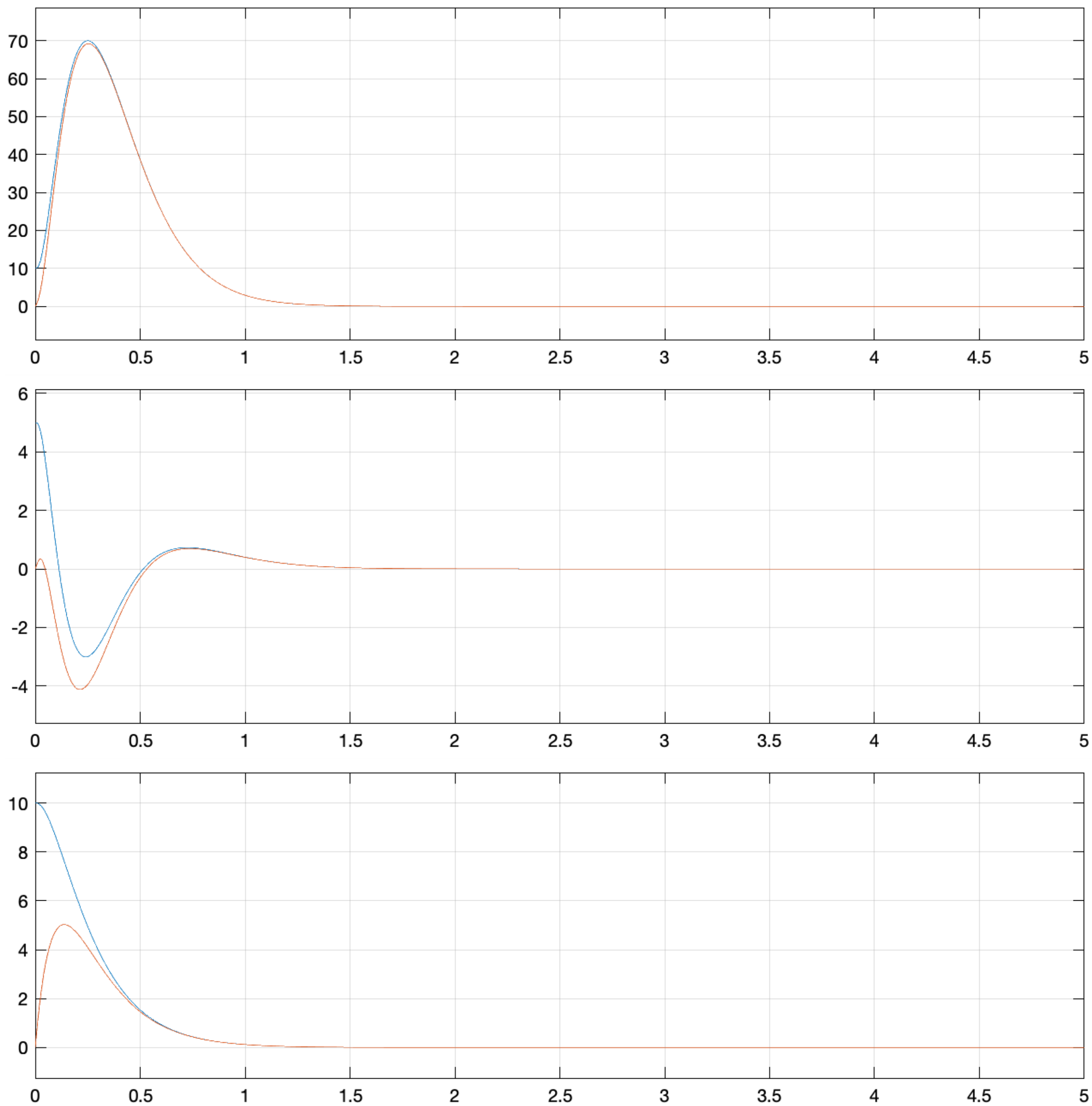
Thus, the system Simulink model and the observer-based controller works

2. Using algebraic Riccati equation solution

```
alpha_con = 1/1e-2;
[P, Lambda, K] = care(A, B*sqrt(alpha_con), C.'*C);
K = K*sqrt(alpha_con)
```

Here, *alpha_con* represents $\frac{1}{\rho}$ in the Algebraic Riccati Equation:

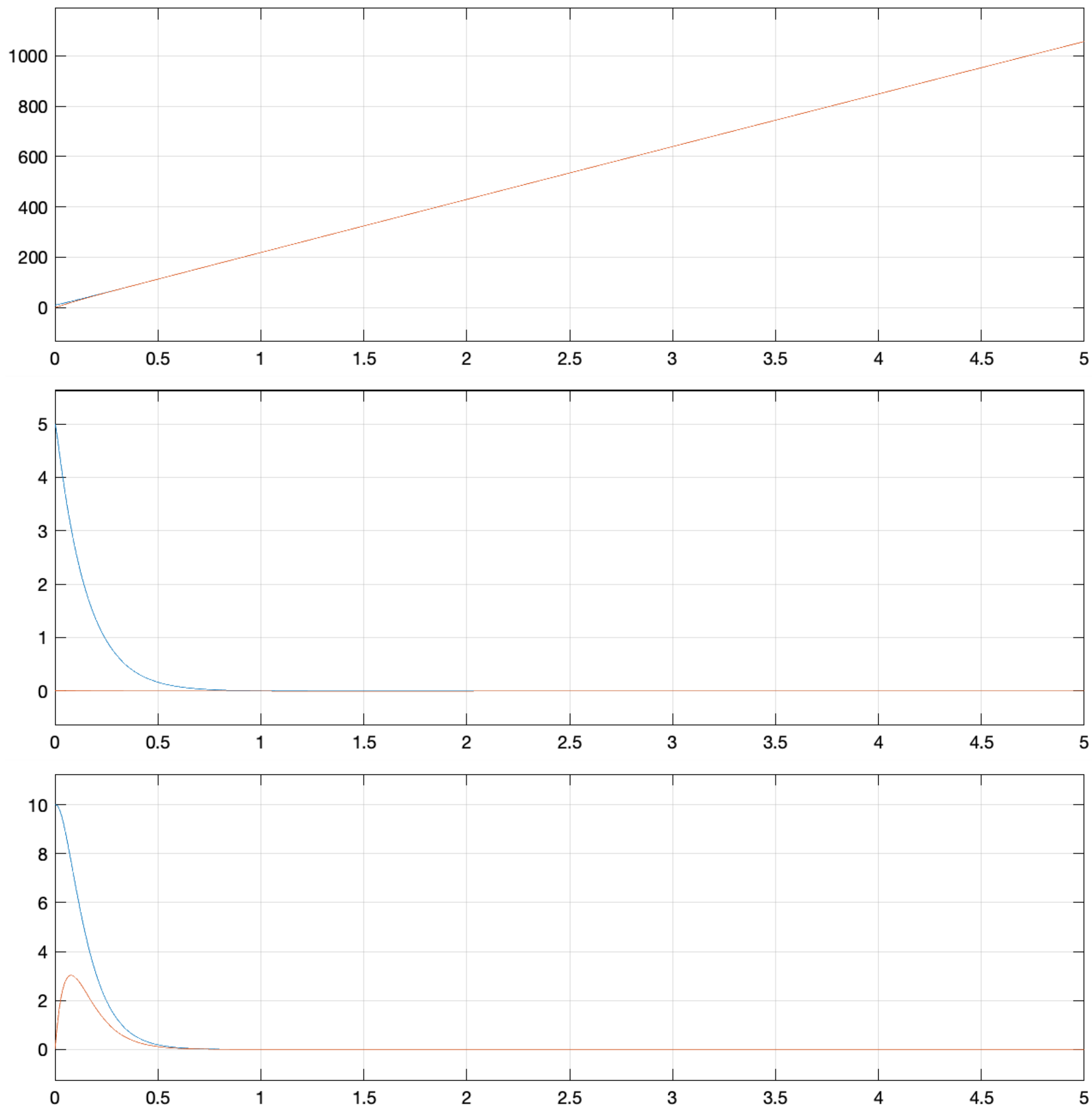
$$A^T P + PA - \frac{1}{\rho} PBB^T P + C^T C = 0$$



Three angles converge quicker, but not good enough, especially for angle ψ (the middle one)

3. Try LQR

```
Q_lqr = [
    1e-6   0   0   0   0   0;
    0   0   0   0   0   0;
    0   0   1e9   0   0   0;
    0   0   0   0   0   0;
    0   0   0   0   1   0;
    0   0   0   0   0   0];
R_lqr = [
    1e-3   0;
    0   1e-3];
N_lqr = zeros(6, 2)
[K, S_con, eval_con] = lqr(A, B, Q_lqr, R_lqr, N_lqr)
```



This is the best result I can get using LQR to minimize the settling time for tilting angle ψ (the middle one), even though the settling time for ψ is less than 1 second (which is what we desired), obviously wheel angle θ won't converge (I think this is because those two angles are coupled in the nonlinear model, and LQR method is a method optimizing all the angles simultaneously, when we focus on one angle, the other will behave less satisfyingly like above).

4. Back to pole placement

```
% K = place(A, B, 1e2*[-3.4577 + 0.0000i, -0.0857 + 0.0000i, ...
%     -0.0757 + 0.0092i, -0.0757 - 0.0092i, ...
%     -1.8270 + 0.0000i, -0.0519 + 0.0000i])
% K = place(A, B, 1e2*[-3.5, -0.09, ...
%     -0.08, -0.08, ...
%     -2, -5e-2])
% K = place(A, B, 1e2*[-5, -1e-2, ...
%     -1.5, -1.5, ...
%     -2, -1e-2])
% K = place(A, B, 1e2*[-5, -2e-2, ...
%     -0.8, -0.8, ...
%     -1, -2e-2])
K = place(A, B, 1e2*[-5, ...
    -1, -0.8, -0.8, ...
    -2e-2, -2e-2])
```

Use the closed-loop eigenvalues generated from the Algebraic Riccati Equation above, see the outcome and tendency from changing the value of each eigenvalue slightly one by one, then find the best eigenvalue pair that suits the project.
List below show the “function” we defined for each eigenvalue.

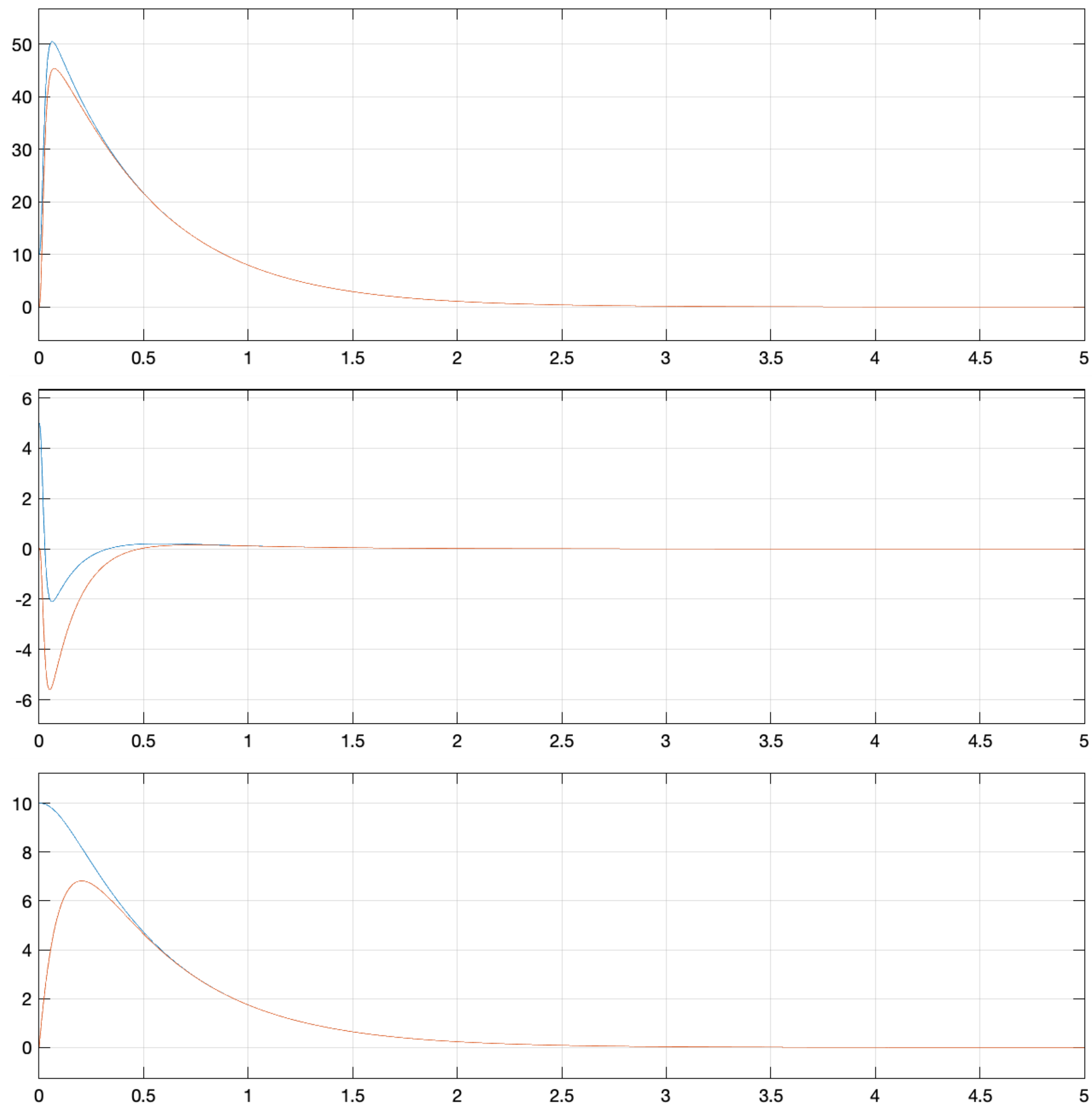
```
K:
first one:
        larger: useless, but may cause discrete model unstable
        smaller: \phi overshot

second one:
        larger: useless, but may cause discrete model unstable
        smaller: \phi overshot

third one:
        value: less than 2, currently 0.1 is better
        larger: \theta \psi, \phi overshot larger, but converges quicker;
        smaller: \theta smoother but converges slower, \psi no second overshot


fourth and fifth one:
        value: near the third one
        smaller: \theta converges slower but converges slower, \psi no second overshot,
\phi overshoot and converges slower


sixth one:
        value: -1e-3 to 1e-2
        larger: unstable;
        smaller: \psi converges quicker with smaller overshot, but \theta converges more
slower and may never reach equilibrium
```



And this is the final result we have for the continuous model with observer-based controller.