AERSP597 Midterm

Ani Perumalla

April 1, 2021

1 Q. #2

[1]:
```
# Import all the functions used in part 1
from era_okid_tools import *

# Logistics
warnings.simplefilter("ignore", UserWarning)
sympy.init_printing()
figs_dir = (Path.cwd() / "figs")
figs_dir.mkdir(parents = True, exist_ok = True)
prob = 2
```

[2]:
```
# Set seed for consistent results
rng = np.random.default_rng(seed = 100)

# Simulation dimensions
noises = (0.001, 0.01, 0.1, 0.5) # Standard deviations of noises
cases = 3 # Number of cases
n = 2 # Number of states
r = 1 # Number of inputs
m = 2 # Number of measurements
t_max = 50 # Total simulation time
dt = 0.1 # Simulation timestep duration
nt = int(t_max/dt) # Number of simulation timesteps

# Simulation time
train_cutoff = int(20/dt) + 1
t_sim = np.linspace(0, t_max, nt + 1)
t_train = t_sim[:train_cutoff]
t_test = t_sim
nt_train = train_cutoff
nt_test = nt

# Problem parameters
theta_0 = 0.5 # Angular velocity
k = 10 # Spring stiffness
mass = 1 # Point mass
```

```
# State space model
A_c = np.array([[0, 1], [theta_0**2 - k/mass, 0]])
B_c = np.array([[0], [1]])
C = np.eye(2)
D = np.array([[0], [1]])
A, B = c2d(A_c, B_c, dt)
eig_A = spla.eig(A_c)[0] # Eigenvalues of true system
etch(f"\lambda", eig_A)
etch(f"\omega_{{n}}", np.abs(eig_A))
etch(f"\zeta", -np.cos(np.angle(eig_A)))

# True simulation values
X_0_sim = np.zeros([n, 1]) # Zero initial condition
U_sim = np.zeros([cases, r, nt]) # True input vectors
U_sim[0] = rng.normal(0, 0.1, [r, nt]) # True input for case 1
U_sim[1] = spsg.square(2*np.pi*5*t_sim[:-1]) # True input for case 2
U_sim[2] = np.cos(2*np.pi*2*t_sim[:-1]) # True input for case 3
X_sim = np.zeros([cases, n, nt + 1]) # True state vectors
Z_sim = np.zeros([cases, m, nt]) # True observation vectors
W_sim = np.zeros([len(noises), cases, m, nt]) # Measurement noise vectors

# Separation into train and test data
U_train = U_sim[0, :r, :train_cutoff] # Train input vector
U_test = U_sim # Test input vectors
X_train = np.zeros([len(noises), n, nt_train]) # Train state vector
X_test = np.zeros([len(noises), cases, n, nt_test + 1]) # Test state vectors
Z_train = np.zeros([len(noises), m, nt_train]) # Train observation vector
Z_test = np.zeros([len(noises), cases, m, nt_test]) # Test observation vectors
V_train = np.zeros([len(noises), r + m, nt_train]) # Train observation input␣
 ↪vectors
V_test = np.zeros([len(noises), cases, r + m, nt_test]) # Test observation␣
 ↪input vectors
```

$$\lambda = \begin{bmatrix} 3.1225i \\ -3.1225i \end{bmatrix}$$

$$\omega_n = \begin{bmatrix} 3.1225 \\ 3.1225 \end{bmatrix}$$

$$\zeta = \begin{bmatrix} 0.0 \\ 0.0 \end{bmatrix}$$

[3]:
```
# OKID logistics
order = 50 # Order of OKID algorithm, number of Markov parameters to identify␣
 ↪after the zeroeth
alpha, beta = 15, 20 # Number of block rows and columns in Hankel matrices
n_era = 2 # Number of proposed states
```

```
X_0_okid = np.zeros([n_era, 1]) # Zero initial condition
```

[4]:
```
# OKID System Markov parameters
Y_okid = np.zeros([len(noises), order + 1, m, r])
# OKID Observer Markov Gain parameters
Y_og_okid = np.zeros([len(noises), order, m, m])
# OKID state vector, drawn from state space model derived from OKID/ERA
X_okid_train = np.zeros([len(noises), n_era, nt_train + 1])
X_okid_test = np.zeros([len(noises), cases, n_era, nt_test + 1])
X_okid_train_obs = np.zeros([len(noises), n_era, nt_train + 1])
X_okid_test_obs = np.zeros([len(noises), cases, n_era, nt_test + 1])
# OKID observations, drawn from state space model derived from OKID/ERA
Z_okid_train = np.zeros([len(noises), n_era, nt_train])
Z_okid_test = np.zeros([len(noises), cases, n_era, nt_test])
Z_okid_train_obs = np.zeros([len(noises), n_era, nt_train])
Z_okid_test_obs = np.zeros([len(noises), cases, n_era, nt_test])
# Singular values of the Hankel matrix constructed through OKID Markov␣
↪parameters
S_okid = np.zeros([len(noises), min(alpha*m, beta*r)])
eig_A_okid = np.zeros([len(noises), n_era], dtype = complex)

# OKID/ERA state space model
A_okid = np.zeros([len(noises), n_era, n_era])
B_okid = np.zeros([len(noises), n_era, r])
C_okid = np.zeros([len(noises), m, n_era])
D_okid = np.zeros([len(noises), m, r])
G_okid = np.zeros([len(noises), m, m])
# OKID/ERA state space model augmented with observer
A_okid_obs = np.zeros([len(noises), n_era, n_era])
B_okid_obs = np.zeros([len(noises), n_era, r + m])
C_okid_obs = np.zeros([len(noises), m, n_era])
D_okid_obs = np.zeros([len(noises), m, r + m])
```

[5]:
```
# Simulation
for i, j in it.product(range(cases), range(len(noises))):
    W_sim[j, i] = rng.normal(0, noises[j], size = Z_sim[i].shape)
    X_sim[i], Z_sim[i] = sim_ss(A, B, C, D, X_0 = X_0_sim, U = U_sim[i], nt =␣
↪nt)
    if i == 0:
        # Split between train and test data for case 1
        X_train[j], Z_train[j] = \
            X_sim[i, :, :train_cutoff], (Z_sim[i, :, :train_cutoff] + W_sim[j,␣
↪i, :, :train_cutoff])
        # Identify System Markov parameters and Observer Gain Markov parameters
        Y_okid[j], Y_og_okid[j] = \
            okid(Z_train[j], U_train,
                 l_0 = order, alpha = alpha, beta = beta, n = n_era)
```

```
        # Identify state space model using System Markov parameters for ERA
        A_okid[j], B_okid[j], C_okid[j], D_okid[j], S_okid[j] = \
            era(Y_okid[j], alpha = alpha, beta = beta, n = n_era)
        # Construct observability matrix
        O_p_okid = np.array([C_okid[j] @ np.linalg.matrix_power(A_okid[j], i)
                             for i in range(order)])
        # Find observer gain matrix
        G_okid[j] = spla.pinv2(O_p_okid.reshape([order*m, n_era])) @␣
↪Y_og_okid[j].reshape([order*m, m])
        # Augment state space model with observer
        A_okid_obs[j] = A_okid[j] + G_okid[j] @ C_okid[j]
        B_okid_obs[j] = np.concatenate([B_okid[j] + G_okid[j] @ D_okid[j],␣
↪-G_okid[j]], 1)
        C_okid_obs[j] = C_okid[j]
        D_okid_obs[j] = np.concatenate([D_okid[j], np.zeros([m, m])], 1)
        V_train[j] = np.concatenate([U_train, Z_train[j]], 0)
        # Simulate OKID realization with "raw" state and OKID realization with␣
↪estimated state
        X_okid_train[j], Z_okid_train[j] = \
            sim_ss(A_okid[j], B_okid[j], C_okid[j], D_okid[j],
                   X_0 = X_0_okid, U = U_train, nt = nt_train)
        X_okid_train_obs[j], Z_okid_train_obs[j] = \
            sim_ss(A_okid_obs[j], B_okid_obs[j], C_okid_obs[j], D_okid_obs[j],
                   X_0 = X_0_okid, U = V_train[j], nt = nt_train)
        # Display outputs
        etch(f"A_{{OKID}}(\eta = {noises[j]})", A_okid[j])
        etch(f"B_{{OKID}}(\eta = {noises[j]})", B_okid[j])
        etch(f"C_{{OKID}}(\eta = {noises[j]})", C_okid[j])
        etch(f"D_{{OKID}}(\eta = {noises[j]})", D_okid[j])
        etch(f"G_{{OKID}}(\eta = {noises[j]})", G_okid[j])
        # Calculate and display eigenvalues
        eig_A_okid[j] = spla.eig(d2c(A_okid[j], B_okid[j], dt)[0])[0] #␣
↪Eigenvalues of identified system
        etch(f"\hat{{\lambda}}(\eta = {noises[j]})", eig_A_okid[j])
        etch(f"\hat{{\omega}}_{{n}}(\eta = {noises[j]})", np.abs(eig_A_okid[j]))
        etch(f"\hat{{\zeta}}(\eta = {noises[j]})", -np.cos(np.
↪angle(eig_A_okid[j])))
    X_test[j, i], Z_test[j, i] = \
        X_sim[i], (Z_sim[i] + W_sim[j, i])
    X_okid_test[j, i], Z_okid_test[j, i] = \
        sim_ss(A_okid[j], B_okid[j], C_okid[j], D_okid[j],
               X_0 = X_0_okid, U = U_test[i], nt = nt_test)
    V_test[j, i] = np.concatenate([U_test[i], Z_test[j, i]], 0)
    X_okid_test_obs[j, i], Z_okid_test_obs[j, i] = \
        sim_ss(A_okid_obs[j], B_okid_obs[j], C_okid_obs[j], D_okid_obs[j],
               X_0 = X_0_okid, U = V_test[j, i], nt = nt_test)
```

```
Rank of H(0): 20
Rank of H(1): 20
Rank of H(0): 20
Rank of H(1): 20
Rank of H(0): 20
Rank of H(1): 20
Rank of H(0): 20
Rank of H(1): 20
```

$$A_{OKID}(\eta = 0.001) = \begin{bmatrix} 0.95328 & -0.3363 \\ 0.2805 & 0.94974 \end{bmatrix}$$

$$B_{OKID}(\eta = 0.001) = \begin{bmatrix} -0.22304 \\ -0.20422 \end{bmatrix}$$

$$C_{OKID}(\eta = 0.001) = \begin{bmatrix} 0.05973 & -0.09051 \\ -0.25697 & -0.20499 \end{bmatrix}$$

$$D_{OKID}(\eta = 0.001) = \begin{bmatrix} 0.00112 \\ 1.00066 \end{bmatrix}$$

$$G_{OKID}(\eta = 0.001) = \begin{bmatrix} -0.09684 & 0.19216 \\ 0.02143 & 0.07533 \end{bmatrix}$$

$$\hat{\lambda}(\eta = 0.001) = \begin{bmatrix} -0.00152 + 3.12221i \\ -0.00152 - 3.12221i \end{bmatrix}$$

$$\hat{\omega}_n(\eta = 0.001) = \begin{bmatrix} 3.12221 \\ 3.12221 \end{bmatrix}$$

$$\hat{\zeta}(\eta = 0.001) = \begin{bmatrix} 0.00049 \\ 0.00049 \end{bmatrix}$$

$$A_{OKID}(\eta = 0.01) = \begin{bmatrix} 0.95388 & 0.33775 \\ -0.28623 & 0.95335 \end{bmatrix}$$

$$B_{OKID}(\eta = 0.01) = \begin{bmatrix} -0.22103 \\ 0.19442 \end{bmatrix}$$

$$C_{OKID}(\eta = 0.01) = \begin{bmatrix} 0.06856 & 0.09306 \\ -0.24908 & 0.19842 \end{bmatrix}$$

$$D_{OKID}(\eta = 0.01) = \begin{bmatrix} 0.00258 \\ 1.02114 \end{bmatrix}$$

$$G_{OKID}(\eta = 0.01) = \begin{bmatrix} -0.04897 & 0.09519 \\ -0.23187 & -0.09554 \end{bmatrix}$$

$$\hat{\lambda}(\eta = 0.01) = \begin{bmatrix} 0.03018 + 3.15178i \\ 0.03018 - 3.15178i \end{bmatrix}$$

$$\hat{\omega}_n(\eta = 0.01) = \begin{bmatrix} 3.15192 \\ 3.15192 \end{bmatrix}$$

$$\hat{\zeta}(\eta = 0.01) = \begin{bmatrix} -0.00958 \\ -0.00958 \end{bmatrix}$$

$$A_{OKID}(\eta = 0.1) = \begin{bmatrix} -0.04213 & -0.89319 \\ 0.93711 & -0.16848 \end{bmatrix}$$

$$B_{OKID}(\eta = 0.1) = \begin{bmatrix} 0.40392 \\ -0.26569 \end{bmatrix}$$

$$C_{OKID}(\eta = 0.1) = \begin{bmatrix} -0.16834 & -0.37907 \\ -0.40714 & -0.09103 \end{bmatrix}$$

$$D_{OKID}(\eta = 0.1) = \begin{bmatrix} -0.04839 \\ 1.10276 \end{bmatrix}$$

$$G_{OKID}(\eta = 0.1) = \begin{bmatrix} 0.15765 & -0.17379 \\ -0.12753 & 0.05737 \end{bmatrix}$$

$$\hat{\lambda}(\eta = 0.1) = \begin{bmatrix} -0.84731 + 16.85665i \\ -0.84731 - 16.85665i \end{bmatrix}$$

$$\hat{\omega}_n(\eta = 0.1) = \begin{bmatrix} 16.87793 \\ 16.87793 \end{bmatrix}$$

$$\hat{\zeta}(\eta = 0.1) = \begin{bmatrix} 0.0502 \\ 0.0502 \end{bmatrix}$$

$$A_{OKID}(\eta = 0.5) = \begin{bmatrix} 0.17814 & -0.96852 \\ 0.92465 & 0.10682 \end{bmatrix}$$

$$B_{OKID}(\eta = 0.5) = \begin{bmatrix} -0.72471 \\ 0.67518 \end{bmatrix}$$

$$C_{OKID}(\eta = 0.5) = \begin{bmatrix} -0.81206 & -0.22644 \\ -0.5849 & -0.22807 \end{bmatrix}$$

$$D_{OKID}(\eta = 0.5) = \begin{bmatrix} -0.43204 \\ 1.25503 \end{bmatrix}$$

$$G_{OKID}(\eta = 0.5) = \begin{bmatrix} 0.01048 & -0.06473 \\ 0.01817 & 0.04844 \end{bmatrix}$$

$$\hat{\lambda}(\eta = 0.5) = \begin{bmatrix} -0.44651 + 14.21255i \\ -0.44651 - 14.21255i \end{bmatrix}$$

$$\hat{\omega}_n(\eta = 0.5) = \begin{bmatrix} 14.21956 \\ 14.21956 \end{bmatrix}$$

$$\hat{\zeta}(\eta = 0.5) = \begin{bmatrix} 0.0314 \\ 0.0314 \end{bmatrix}$$

- $\eta = 0.001$: The eigenvalues are very closely identified. Some damping is identified but on the whole the modes are predicted closely.

- $\eta = 0.01$: Although the damping is close to 0, and the natural frequency is close to its true value, the identified eigenvalues contain a positive real part, causing the identified system to be unstable, and the overall identification is quite poor.
- $\eta = 0.1$: The identified eigenvalues are stable but are quite far off from the true values. As a result, the identification is again poor.
- $\eta = 0.5$: What is measured is dominated by noise; as a result, the realization is essentially meaningless. The identified eigenvalues are nowhere close to the true eigenvalues.

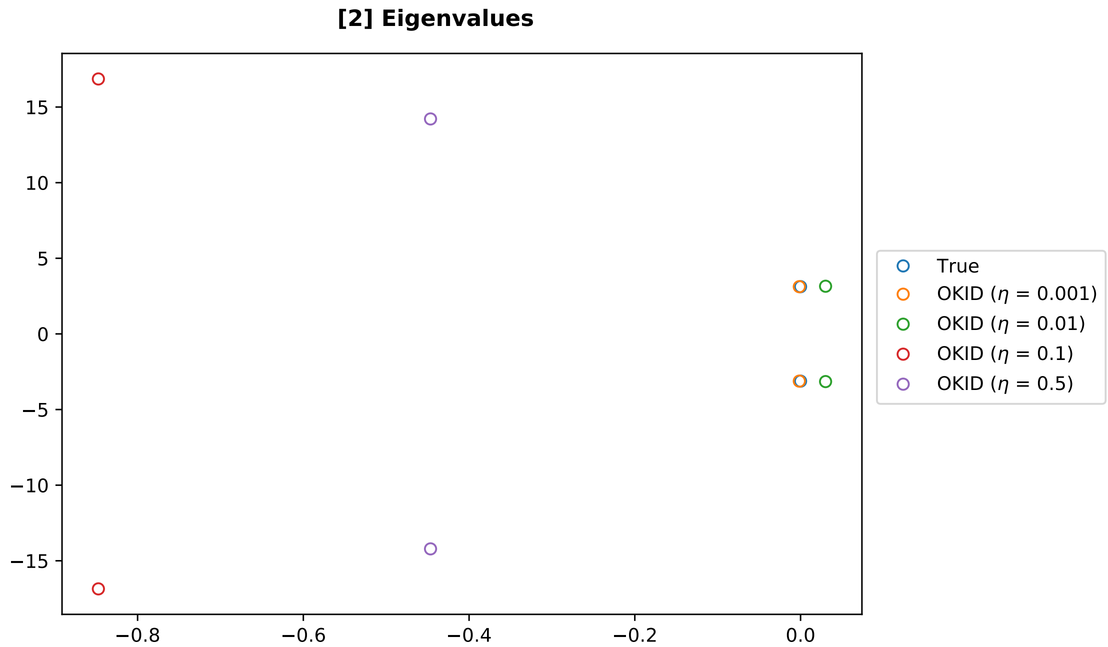
[6]:
```
RMS_train = np.zeros([len(noises), m])
RMS_test = np.zeros([len(noises), cases, m])
for j in range(len(noises)):
    RMS_train[j] = np.sqrt(np.mean((Z_okid_train[j] - Z_train[j])**2, axis = 1))
    print(f"RMS Error of sim. for system found via OKID for train data, noise␣
↪std.dev. = {noises[j]}: {RMS_train[j]}")
    for i in range(cases):
        RMS_test[j, i] = np.sqrt(np.mean((Z_okid_test[j, i] - Z_test[j, i])**2,␣
↪axis = 1))
        print(f"RMS Error of sim. for system found via OKID for test data,␣
↪noise std.dev. = {noises[j]}, case {i}: {RMS_test[j, i]}")
```

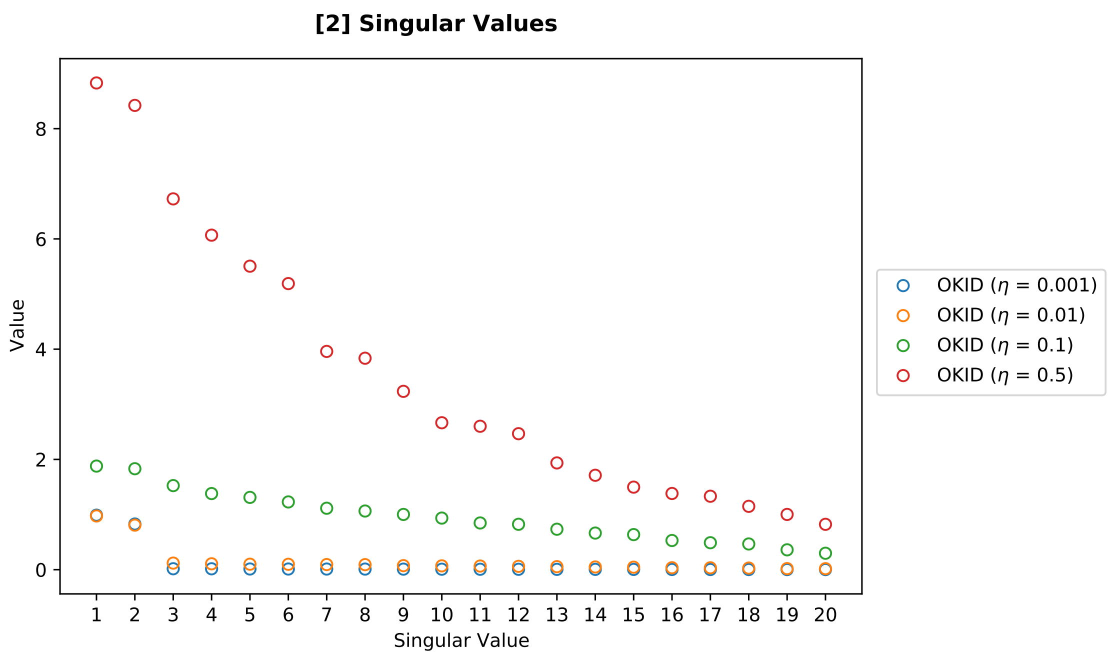
```
RMS Error of sim. for system found via OKID for train data, noise std.dev. =
0.001: [0.00109362 0.00155008]
RMS Error of sim. for system found via OKID for test data, noise std.dev. =
0.001, case 0: [0.00182514 0.00464139]
RMS Error of sim. for system found via OKID for test data, noise std.dev. =
0.001, case 1: [0.00354919 0.011518  ]
RMS Error of sim. for system found via OKID for test data, noise std.dev. =
0.001, case 2: [0.00134281 0.00185268]
RMS Error of sim. for system found via OKID for train data, noise std.dev. =
0.01: [0.02221237 0.05409954]
RMS Error of sim. for system found via OKID for test data, noise std.dev. =
0.01, case 0: [0.08718866 0.24790127]
RMS Error of sim. for system found via OKID for test data, noise std.dev. =
0.01, case 1: [0.18623582 0.56862618]
RMS Error of sim. for system found via OKID for test data, noise std.dev. =
0.01, case 2: [0.03014392 0.08282229]
RMS Error of sim. for system found via OKID for train data, noise std.dev. =
0.1: [0.12149536 0.15746327]
RMS Error of sim. for system found via OKID for test data, noise std.dev. = 0.1,
case 0: [0.11818669 0.17565933]
RMS Error of sim. for system found via OKID for test data, noise std.dev. = 0.1,
case 1: [0.51899148 1.17558481]
RMS Error of sim. for system found via OKID for test data, noise std.dev. = 0.1,
case 2: [0.19144686 0.23953701]
RMS Error of sim. for system found via OKID for train data, noise std.dev. =
0.5: [0.51015322 0.54601244]
RMS Error of sim. for system found via OKID for test data, noise std.dev. = 0.5,
case 0: [0.50900493 0.53325144]
```

```
RMS Error of sim. for system found via OKID for test data, noise std.dev. = 0.5,
case 1: [1.42150535 1.51855733]
RMS Error of sim. for system found via OKID for test data, noise std.dev. = 0.5,
case 2: [1.9059425  1.60856124]
```

The RMS error for the test grows as η increases; the RMS is acceptable for $\eta = 0.001$ and for the training case when $\eta = 0.01$, but all test and training cases have high RMS error for $\eta > 0.01$.

[7]:
```python
# Eigenvalue plots
fig, ax = plt.subplots(constrained_layout = True) # type:figure.Figure
fig.suptitle(f"[{prob}] Eigenvalues", fontweight = "bold")

ax.plot(np.real(eig_A), np.imag(eig_A),
         "o", mfc = "None")
for j in range(len(noises)):
    ax.plot(np.real(eig_A_okid[j]), np.imag(eig_A_okid[j]),
            "o", mfc = "None")

fig.legend(labels = ("True", *[f"OKID ($\eta$ = {noise})" for noise in noises]),
           bbox_to_anchor = (1, 0.5), loc = 6)
fig.savefig(figs_dir / f"midterm_{prob}_eigval.pdf",
            bbox_inches = "tight")

# Singular Value plots
fig, ax = plt.subplots(constrained_layout = True) # type:figure.Figure
fig.suptitle(f"[{prob}] Singular Values", fontweight = "bold")

for j in range(len(noises)):
    ax.plot(np.linspace(1, len(S_okid[j]), len(S_okid[j])), S_okid[j],
            "o", mfc = "None")
plt.setp(ax, xlabel = f"Singular Value", ylabel = f"Value",
         xticks = np.arange(1, S_okid.shape[-1] + 1))

fig.legend(labels = [f"OKID ($\eta$ = {noise})" for noise in noises],
           bbox_to_anchor = (1, 0.5), loc = 6)
fig.savefig(figs_dir / f"midterm_{prob}_singval.pdf",
            bbox_inches = "tight")
```

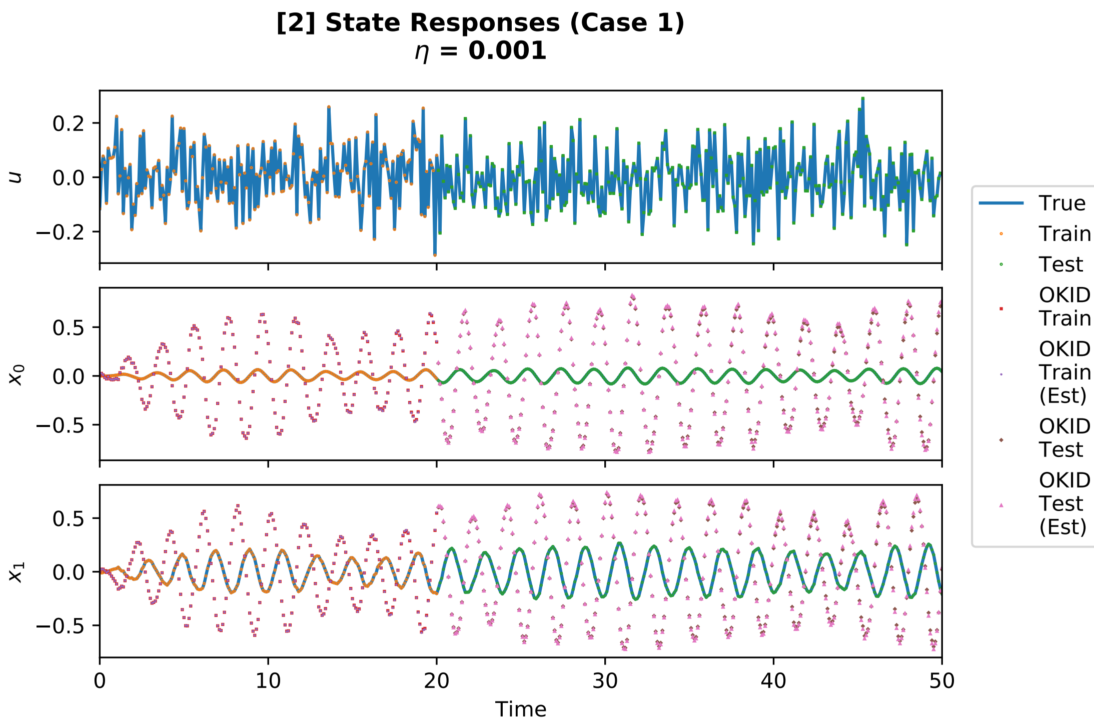


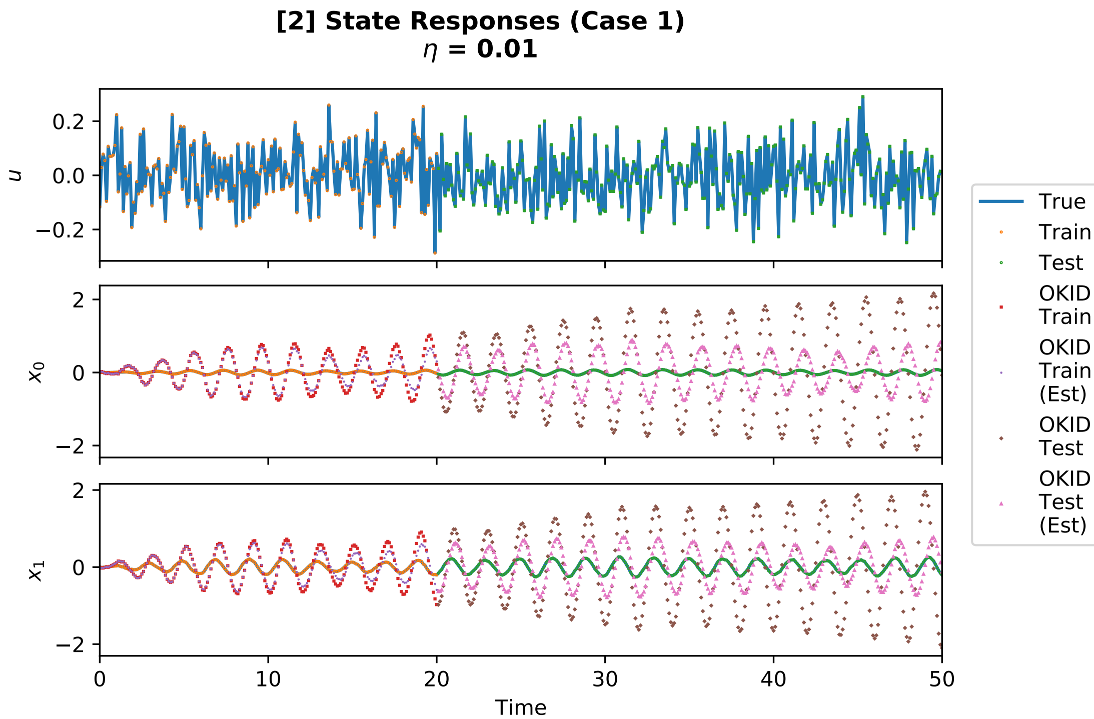


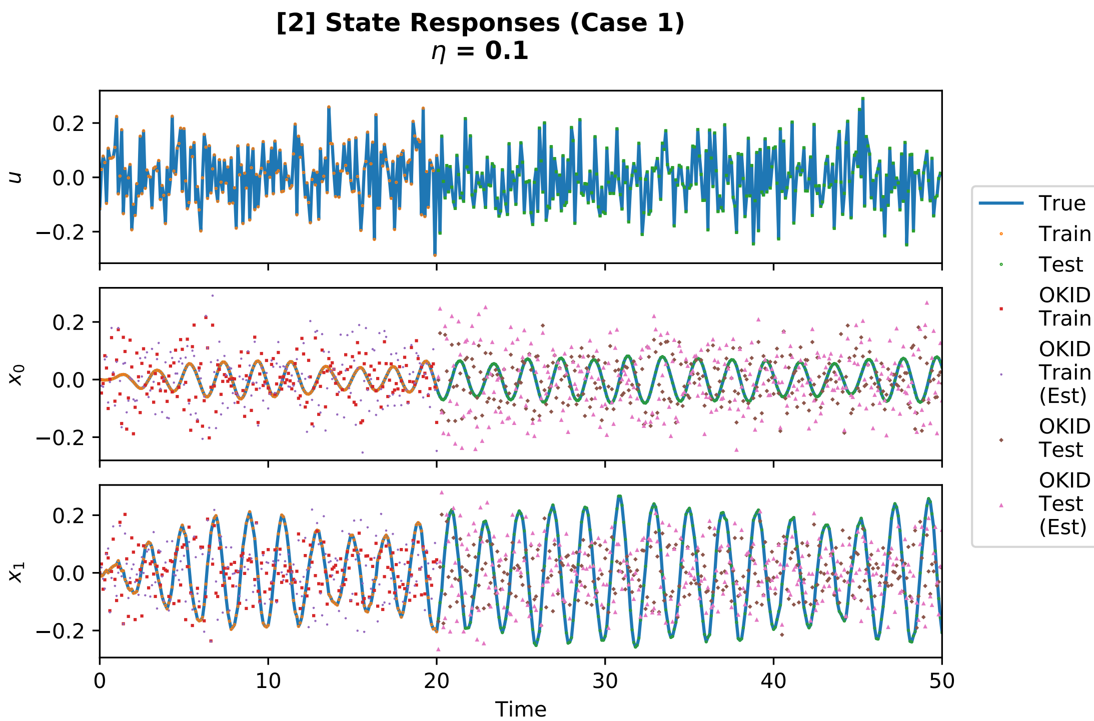
As shown by the singular value plot, the singular values remain somewhat disjoint for $\eta < 0.1$, but as η increases, the singular values converge and the dropoff between the 2nd and 3rd singular values decreases sharply, making it harder to identify that the system has a true order n of 2.

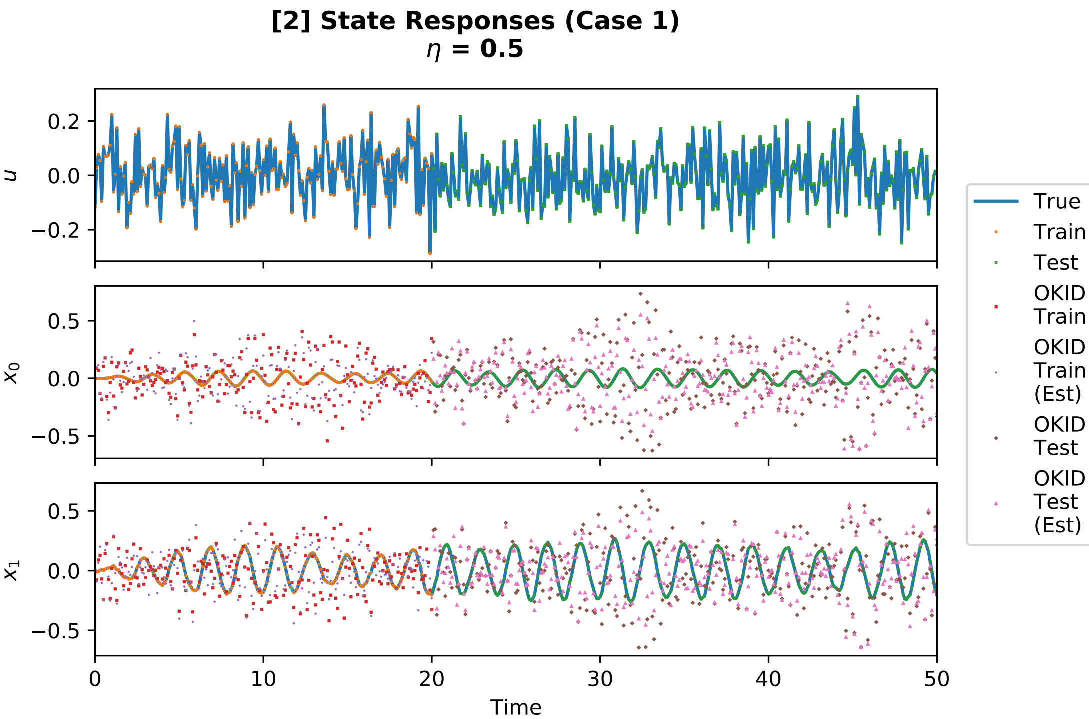
[8]:
```
# Response plots
ms = 0.5 # Marker size
for i, k in it.product(range(cases), range(len(noises))):
    fig, axs = plt.subplots(1 + n, 1,
                            sharex = "col",
                            constrained_layout = True) # type:figure.Figure
    fig.suptitle(f"[{prob}] State Responses (Case {i + 1})\n$\eta$ =␣
↪{noises[k]}",
                 fontweight = "bold")

    if i == 0:
        axs[i].plot(t_sim[:-1], U_sim[i, 0])
        axs[i].plot(t_train, U_train[0],
                    "o", ms = ms, mfc = "None")
        axs[i].plot(t_test[train_cutoff:-1], U_test[i, 0, train_cutoff:],
                    "s", ms = ms, mfc = "None")
        plt.setp(axs[i], ylabel = f"$u$", xlim = [0, t_max])

        for j in range(n):
            axs[j + 1].plot(t_sim, X_sim[i, j])
            axs[j + 1].plot(t_train, X_train[k, j],
                            "o", ms = ms, mfc = "None")
            axs[j + 1].plot(t_test[train_cutoff:], X_test[k, i, j, train_cutoff:
↪],
                            "o", ms = ms, mfc = "None")
            axs[j + 1].plot(t_train, X_okid_train[k, j, :-1],
                            "s", ms = ms, mfc = "None")
            axs[j + 1].plot(t_train, X_okid_train_obs[k, j, :-1],
                            "*", ms = ms, mfc = "None")
            axs[j + 1].plot(t_test[train_cutoff:], X_okid_test[k, i, j,␣
↪train_cutoff:],
                            "D", ms = ms, mfc = "None")
            axs[j + 1].plot(t_test[train_cutoff:], X_okid_test_obs[k, i, j,␣
↪train_cutoff:],
                            "^", ms = ms, mfc = "None")
            plt.setp(axs[j + 1], ylabel = f"$x_{j}$", xlim = [0, t_max])
            if j == 1:
                plt.setp(axs[j + 1], xlabel = f"Time")
        fig.legend(labels = ["_", "_", "_", "True", "Train", "Test",
                             "OKID\nTrain", "OKID\nTrain\n(Est)",
                             "OKID\nTest", "OKID\nTest\n(Est)"],
                   bbox_to_anchor = (1, 0.5), loc = 6)
    else:
        axs[0].plot(t_sim[:-1], U_sim[i, 0])
        axs[0].plot(t_test[:-1], U_test[i, 0],
                    "o", ms = ms, mfc = "None")
        plt.setp(axs[0], ylabel = f"$u$", xlim = [0, t_max])
```

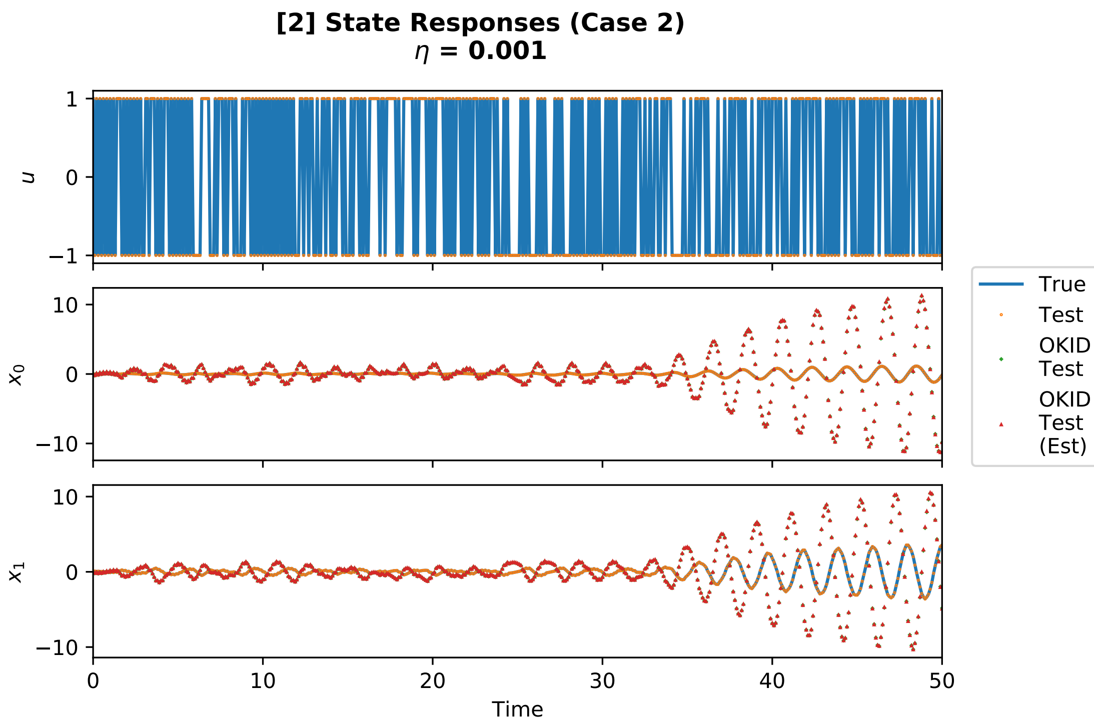
```
    for j in range(n):
        axs[j + 1].plot(t_sim, X_sim[i, j])
        axs[j + 1].plot(t_test, X_test[k, i, j],
                        "o", ms = ms, mfc = "None")
        axs[j + 1].plot(t_test, X_okid_test[k, i, j],
                        "D", ms = ms, mfc = "None")
        axs[j + 1].plot(t_test, X_okid_test_obs[k, i, j],
                        "^", ms = ms, mfc = "None")
        plt.setp(axs[j + 1], ylabel = f"$x_{j}$", xlim = [0, t_max])
        if j == 1:
            plt.setp(axs[j + 1], xlabel = f"Time")
    fig.legend(labels = ["_", "_", "True", "Test",
                         "OKID\nTest", "OKID\nTest\n(Est)"],
               bbox_to_anchor = (1, 0.5), loc = 6)
fig.savefig(figs_dir / f"midterm_{prob}_states_case{i + 1}_noise{k}.pdf",
            bbox_inches = "tight")
```

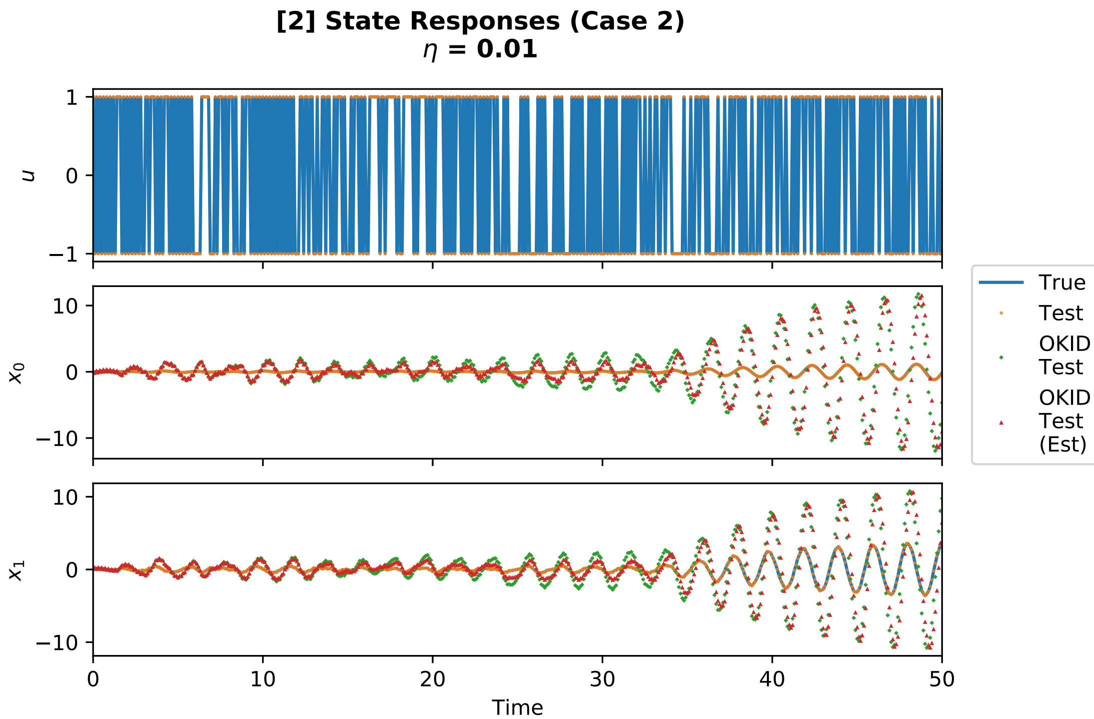


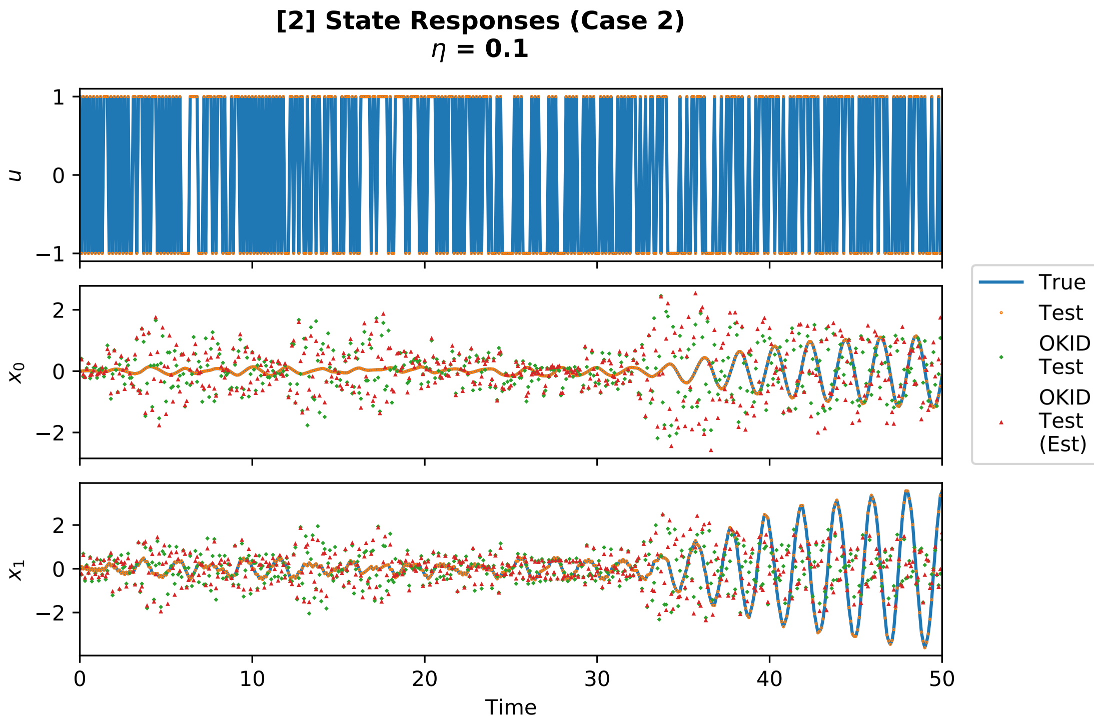

[2] State Responses (Case 1)
η = 0.01
u
x_0
x_1
Time
True
Train
Test
OKID Train
OKID Train (Est)
OKID Test
OKID Test (Est)

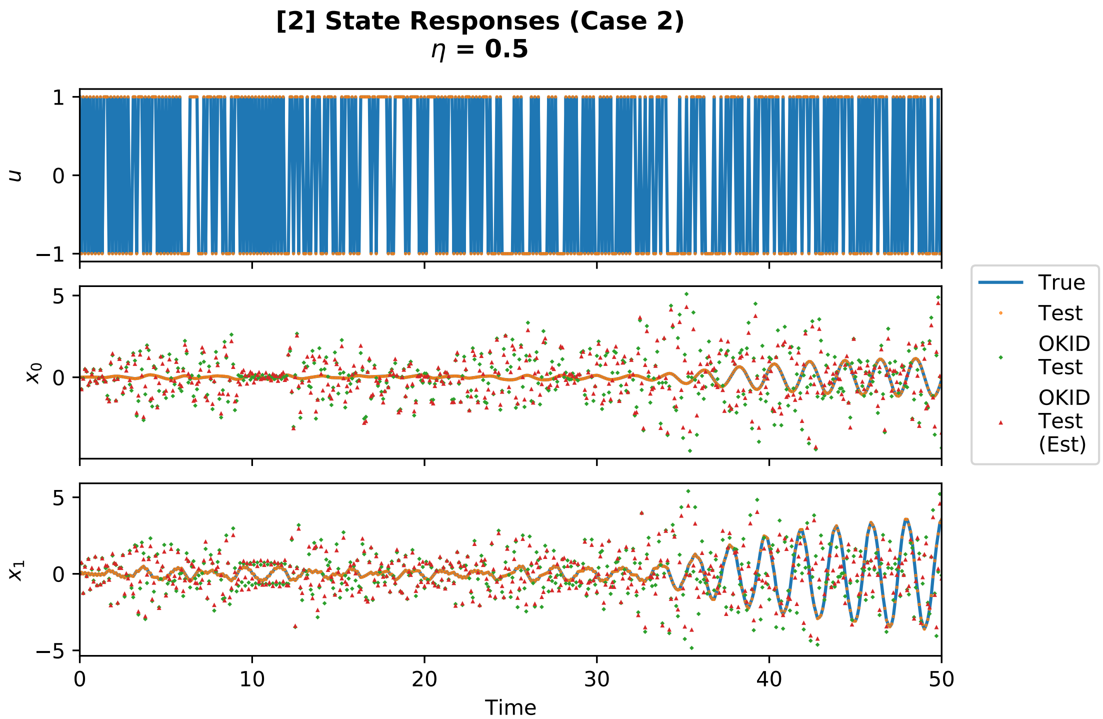

[2] State Responses (Case 1)
η = 0.1
u
x_0
x_1
Time
True
Train
Test
OKID Train
OKID Train (Est)
OKID Test
OKID Test (Est)

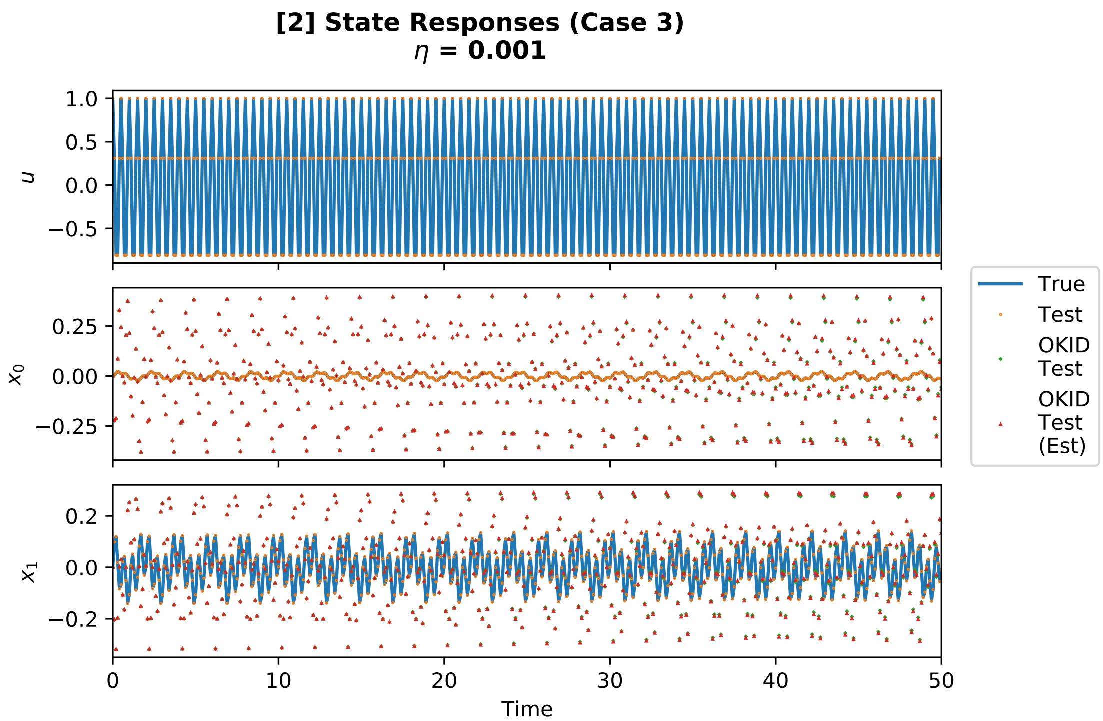

[2] State Responses (Case 1)
η = 0.5
u
x0
x1
Time
True
Train
Test
OKID Train
OKID Train (Est)
OKID Test
OKID Test (Est)

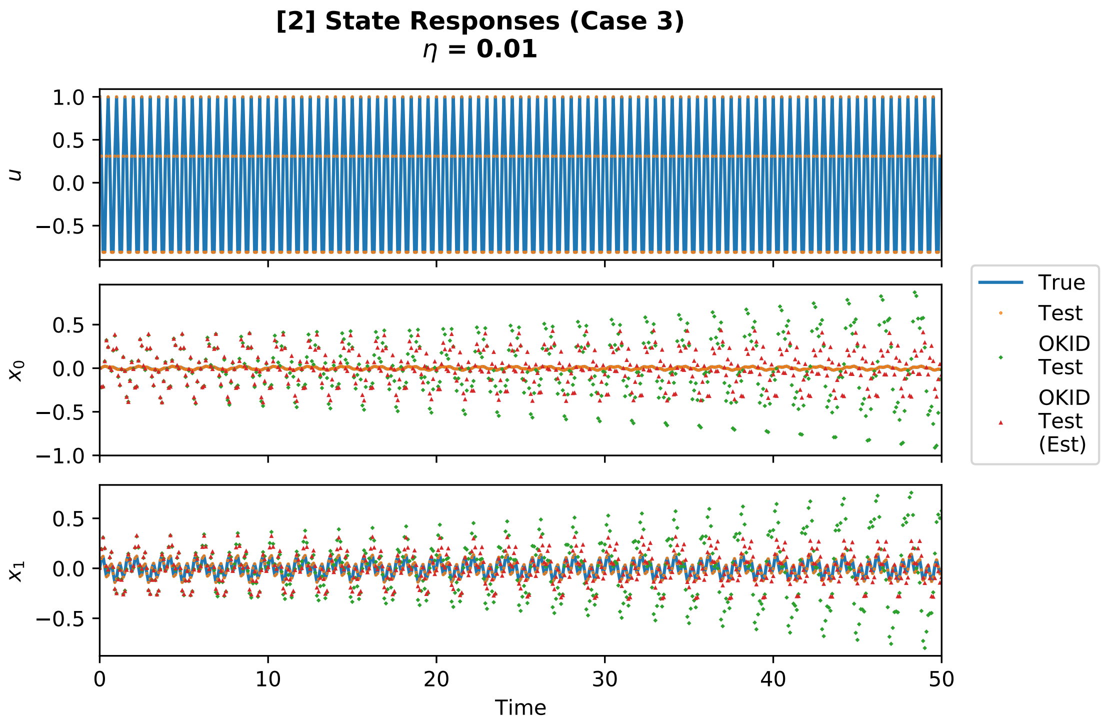

[2] State Responses (Case 2)
η = 0.001
u
x0
x1
Time
True
Test
OKID Test
OKID Test (Est)

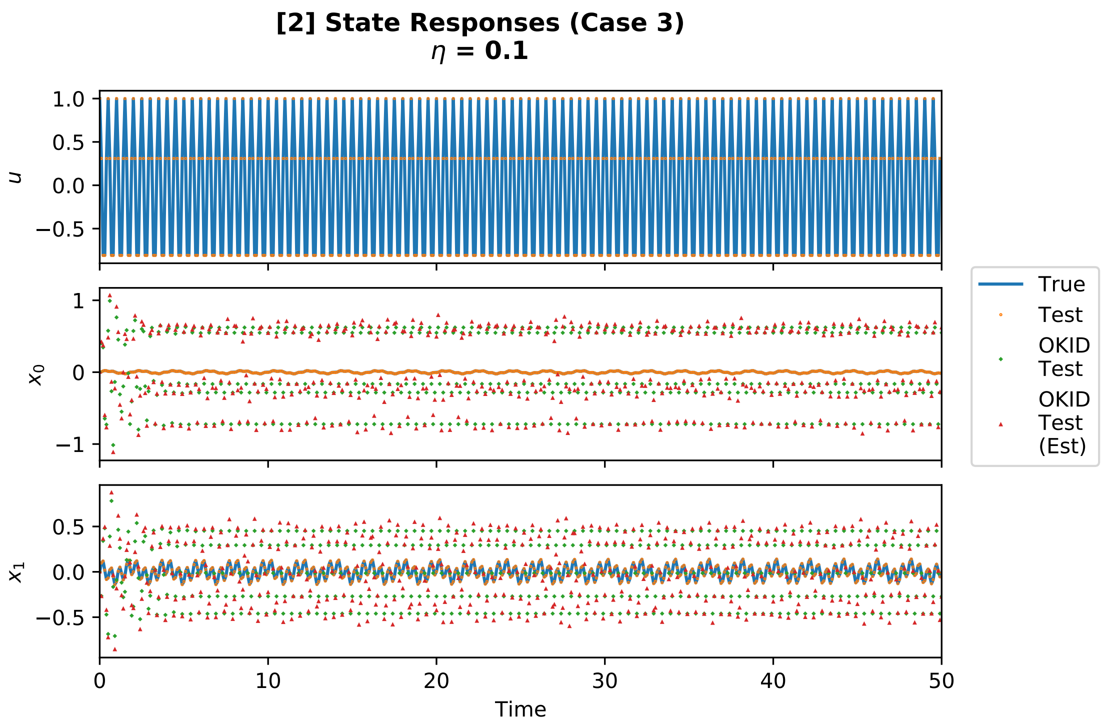

[2] State Responses (Case 2)
η = 0.01
True
Test
OKID Test
OKID Test (Est)
Time

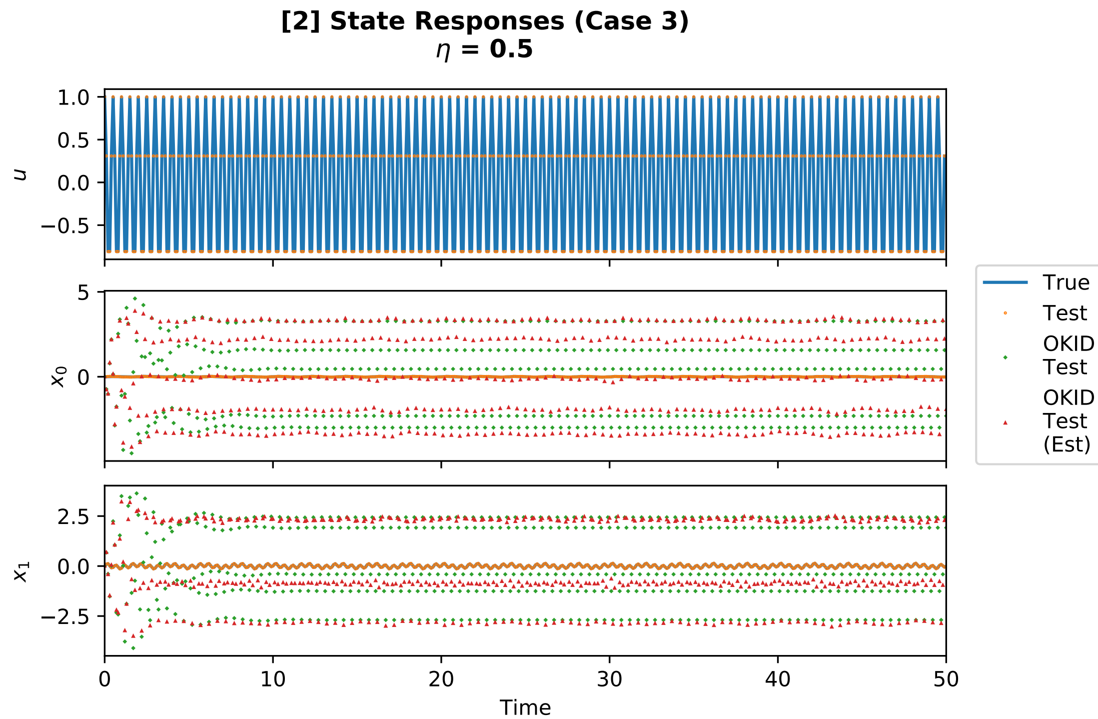

[2] State Responses (Case 2)
η = 0.1
True
Test
OKID Test
OKID Test (Est)
Time


[2] State Responses (Case 2)
η = 0.5
u
x0
x1
Time
True
Test
OKID Test
OKID Test (Est)


[2] State Responses (Case 3)
η = 0.001
u
x0
x1
Time
True
Test
OKID Test
OKID Test (Est)


[2] State Responses (Case 3)
η = 0.01
u
x_0
x_1
Time
True
Test
OKID Test
OKID Test (Est)


[2] State Responses (Case 3)
η = 0.1
u
x_0
x_1
Time
True
Test
OKID Test
OKID Test (Est)

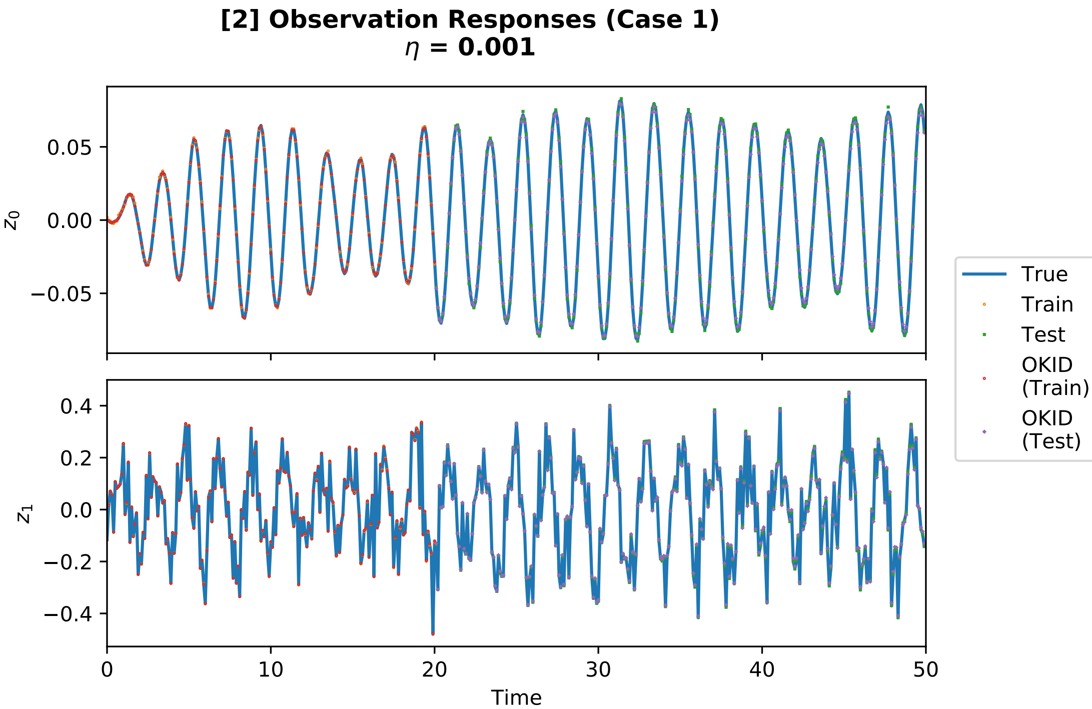


The observer does help the stability of the estimated state when $\eta = 0.01$. However, for $\eta > 0.01$, the observer does not help the estimation in any case examined, as the identified systems are very far from the true systems to begin with.

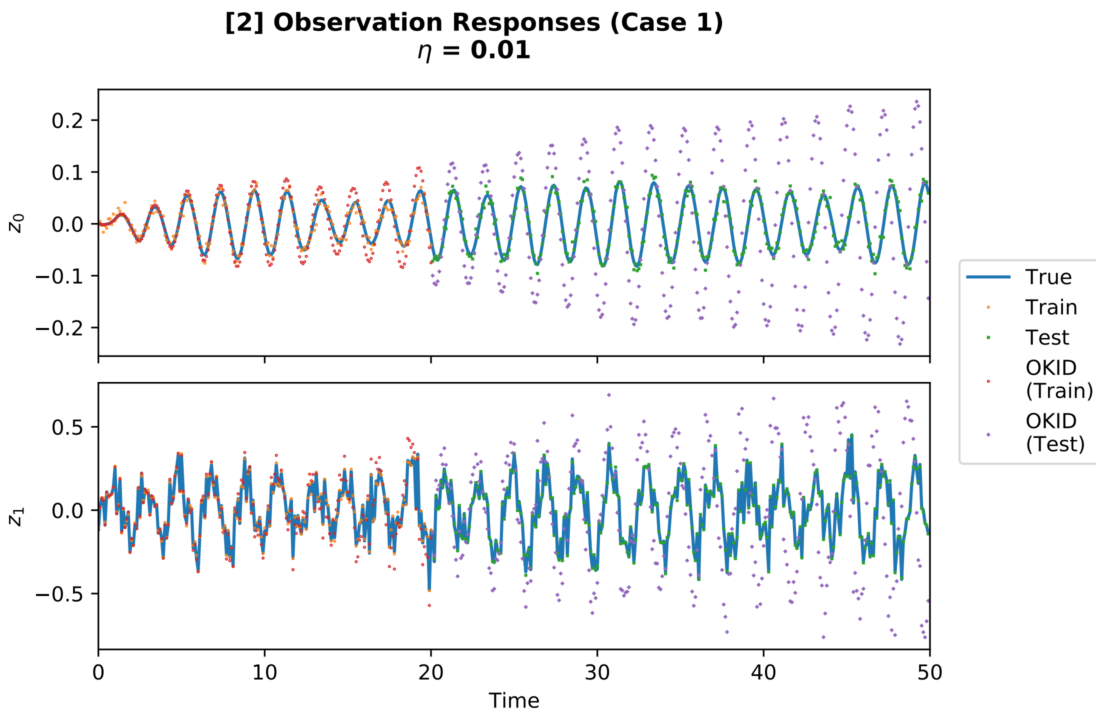
[9]:
```
# Observation plots
for i, k in it.product(range(cases), range(len(noises))):
    # Raw observations
    raw_fig, axs = plt.subplots(m, 1,
                                sharex = "col", constrained_layout = True) #␣
↪type:figure.Figure
    raw_fig.suptitle(f"[{prob}] Observation Responses (Case {i + 1})\n$\eta$ =␣
↪{noises[k]}",
                     fontweight = "bold")
    if i == 0:
        for j in range(m):
            axs[j].plot(t_sim[:-1], Z_sim[i, j])
            axs[j].plot(t_train, Z_train[k, j],
                        "o", ms = ms, mfc = "None")
            axs[j].plot(t_test[train_cutoff:-1], Z_test[k, i, j, train_cutoff:],
                        "s", ms = ms, mfc = "None")
            axs[j].plot(t_train, Z_okid_train[k, j],
                        "o", ms = ms, mfc = "None")
            axs[j].plot(t_test[train_cutoff:-1], Z_okid_test[k, i, j,␣
↪train_cutoff:],
```

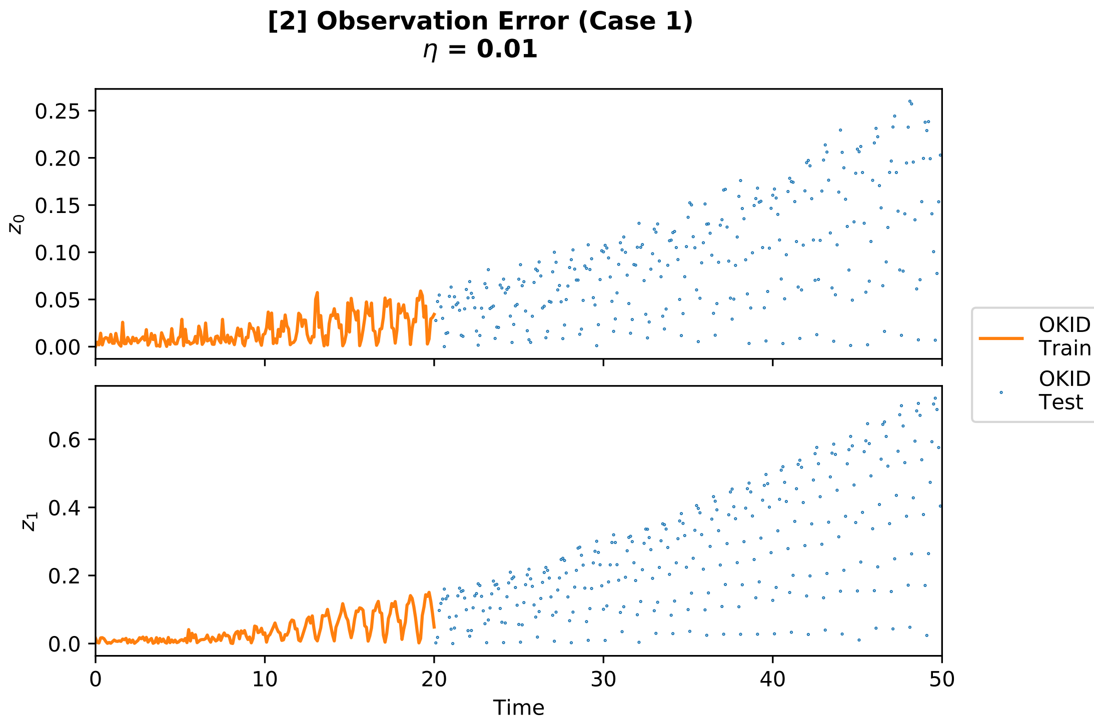
```python
                        "D", ms = ms, mfc = "None")
            plt.setp(axs[j], ylabel = f"$z_{j}$",
                     xlim = [0, t_max])
            if j == (m - 1):
                plt.setp(axs[j], xlabel = f"Time")
        raw_fig.legend(labels = ["True", "Train", "Test",
                                 "OKID\n(Train)", "OKID\n(Test)"],
                       bbox_to_anchor = (1, 0.5), loc = 6)
    else:
        for j in range(m):
            axs[j].plot(t_sim[:-1], Z_sim[i, j])
            axs[j].plot(t_test[:-1], Z_test[k, i, j],
                        "o", ms = ms, mfc = "None")
            axs[j].plot(t_test[:-1], Z_okid_test[k, i, j],
                        "s", ms = ms, mfc = "None")
            plt.setp(axs[j], ylabel = f"$z_{j}$",
                     xlim = [0, t_max])
            if j == (m - 1):
                plt.setp(axs[j], xlabel = f"Time")
        raw_fig.legend(labels = ["True", "Test", "OKID\nTest"],
                       bbox_to_anchor = (1, 0.5), loc = 6)
    raw_fig.savefig(figs_dir / f"midterm_{prob}_obs_case{i + 1}_noise{k}.pdf",
                    bbox_inches = "tight")

    # Observation error
    err_fig, axs = plt.subplots(m, 1,
                                sharex = "col", constrained_layout = True) # type:figure.Figure
    err_fig.suptitle(f"[{prob}] Observation Error (Case {i + 1})\n$\eta$ = {noises[k]}",
                     fontweight = "bold")
    if i == 0:
        for j in range(m):
            axs[j].plot(t_train, np.abs(Z_okid_train[k, j] - Z_train[k, j]),
                        c = "C1")
            axs[j].plot(t_test[train_cutoff:-1], np.abs(Z_okid_test[k, i, j, train_cutoff:] - Z_test[k, i, j, train_cutoff:]),
                        "o", ms = ms, mfc = "None", c = "C0")
            plt.setp(axs[j], ylabel = f"$z_{j}$",
                     xlim = [0, t_max])
            if j == (m - 1):
                plt.setp(axs[j], xlabel = f"Time")
        err_fig.legend(labels = ["OKID\nTrain", "OKID\nTest"],
                       bbox_to_anchor = (1, 0.5), loc = 6)
    else:
        for j in range(m):
```

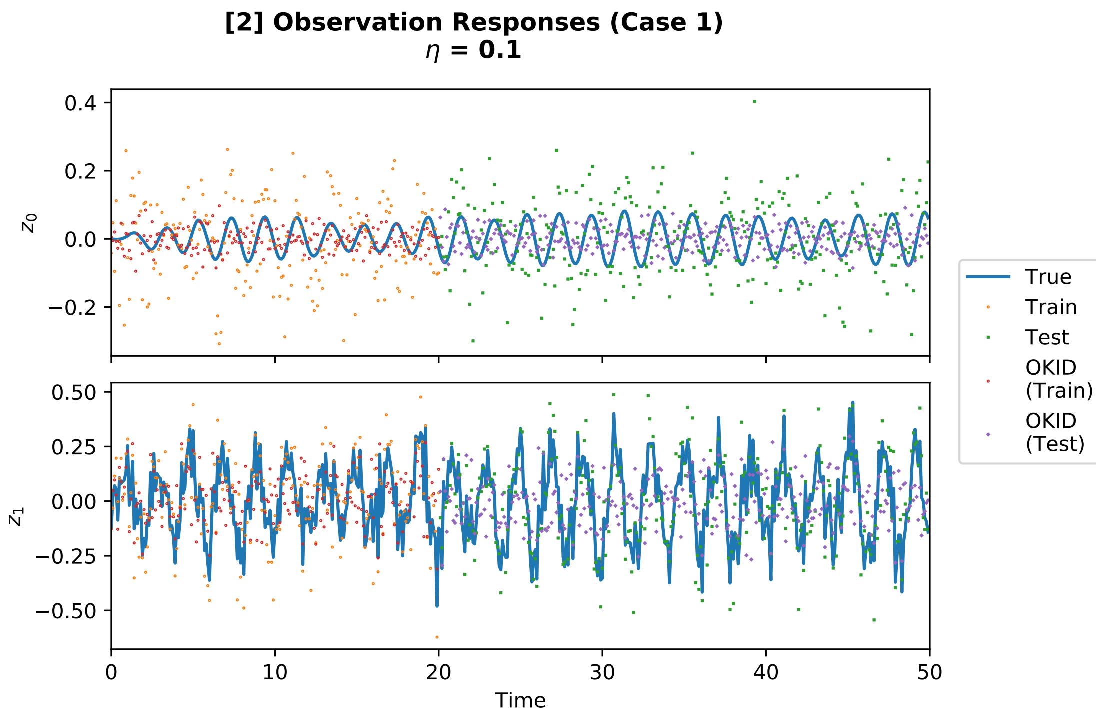
```
            axs[j].plot(t_test[:-1], np.abs(Z_okid_test[k, i, j] - Z_test[k, i,␣
↪j]),
                        "o", ms = ms, mfc = "None")
            plt.setp(axs[j], ylabel = f"$z_{j}$",
                     xlim = [0, t_max])
            if j == (m - 1):
                plt.setp(axs[j], xlabel = f"Time")
        err_fig.legend(labels = ["OKID\nTest"],
                       bbox_to_anchor = (1, 0.5), loc = 6)
    err_fig.savefig(figs_dir / f"midterm_{prob}_obs-error_case{i + 1}_noise{k}.
↪pdf",
                    bbox_inches = "tight")
```

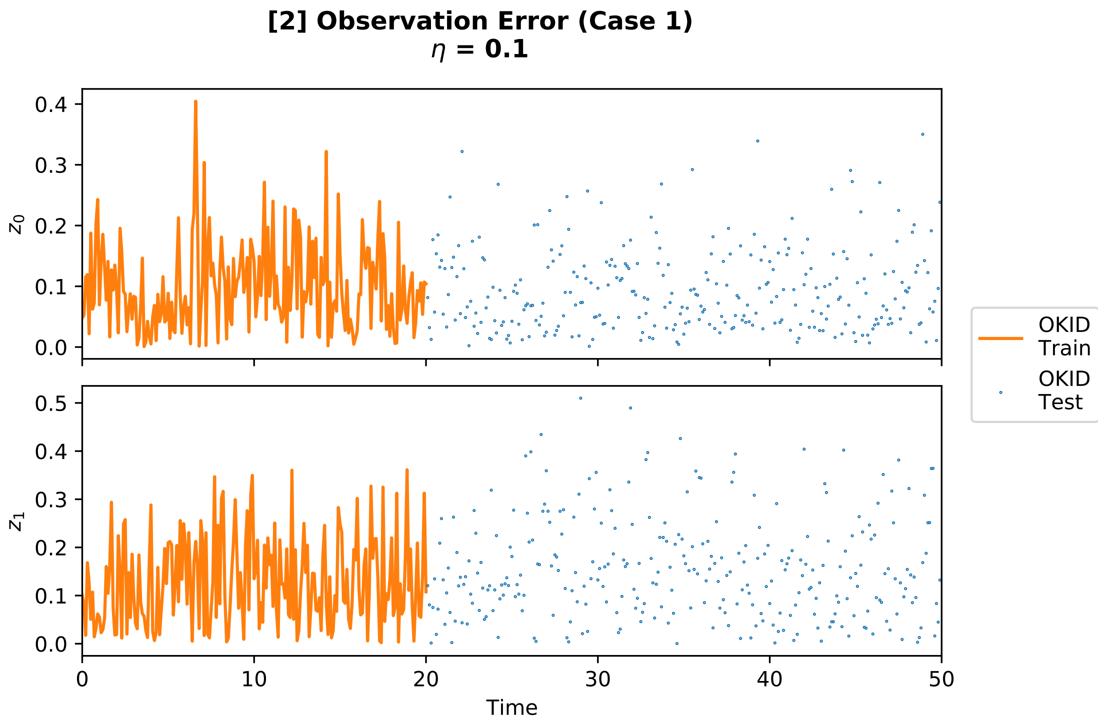



[2] Observation Error (Case 1)
η = 0.001
z_0
z_1
Time
OKID Train
OKID Test

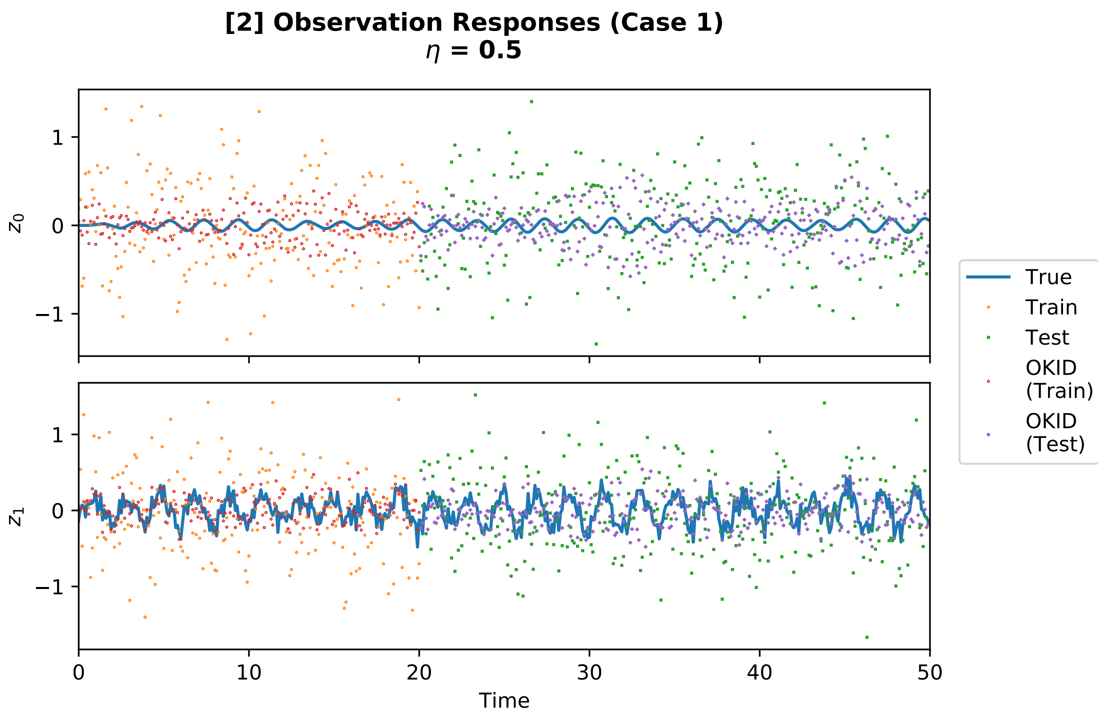

[2] Observation Responses (Case 1)
η = 0.01
z_0
z_1
Time
True
Train
Test
OKID (Train)
OKID (Test)


[2] Observation Error (Case 1)
η = 0.01
OKID Train
OKID Test
z_0
z_1
Time

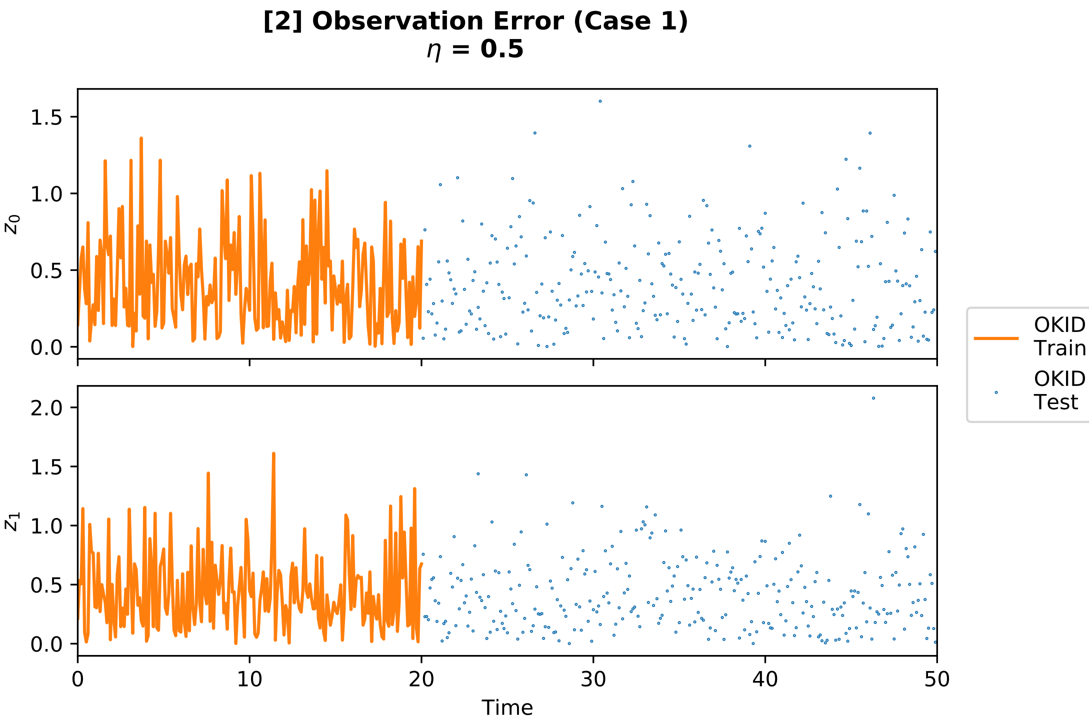

[2] Observation Responses (Case 1)
η = 0.1
True
Train
Test
OKID (Train)
OKID (Test)
z_0
z_1
Time



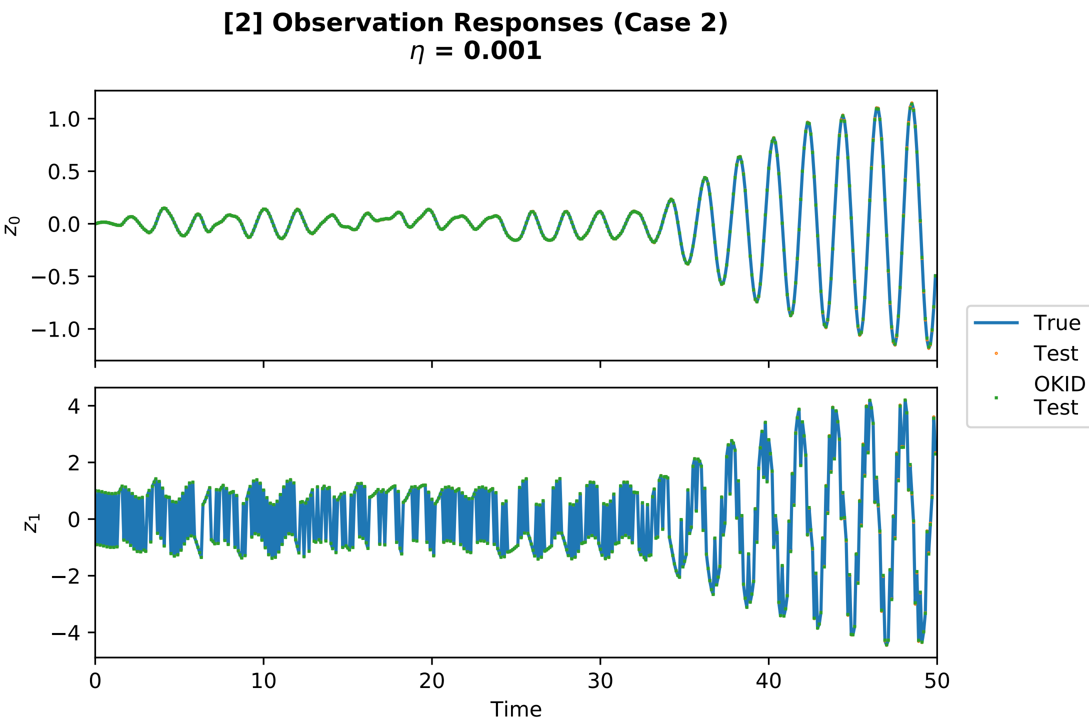
[2] Observation Error (Case 1)
η = 0.1
z_0
z_1
Time
OKID Train
OKID Test



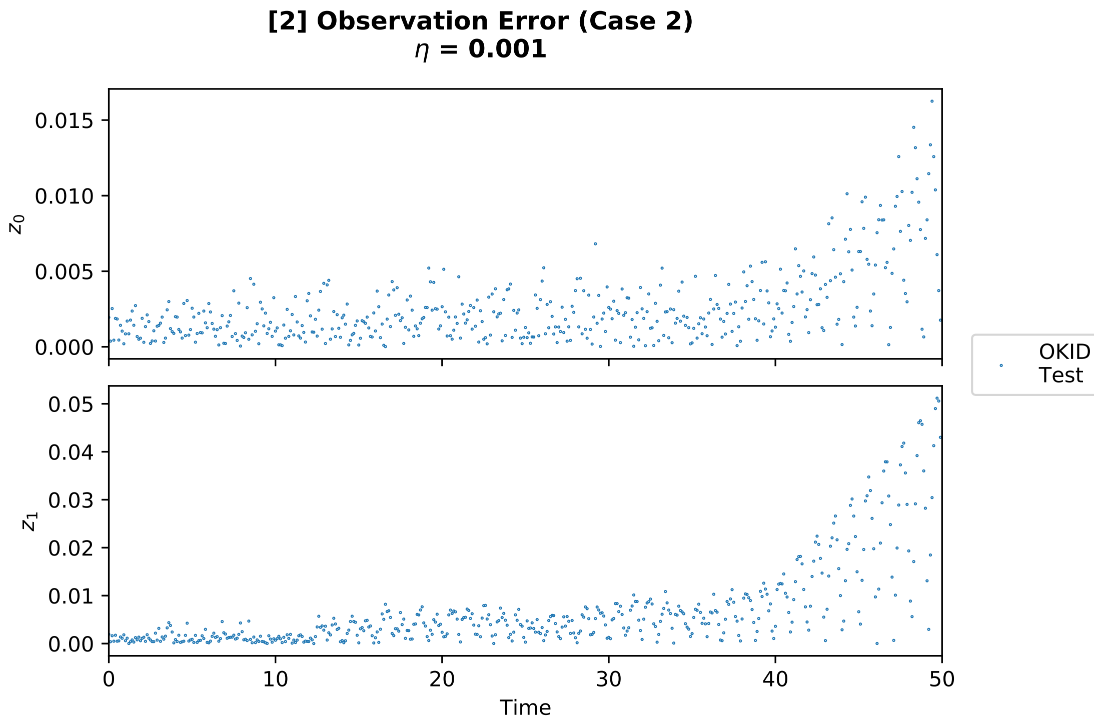
[2] Observation Responses (Case 1)
η = 0.5
z_0
z_1
Time
True
Train
Test
OKID (Train)
OKID (Test)

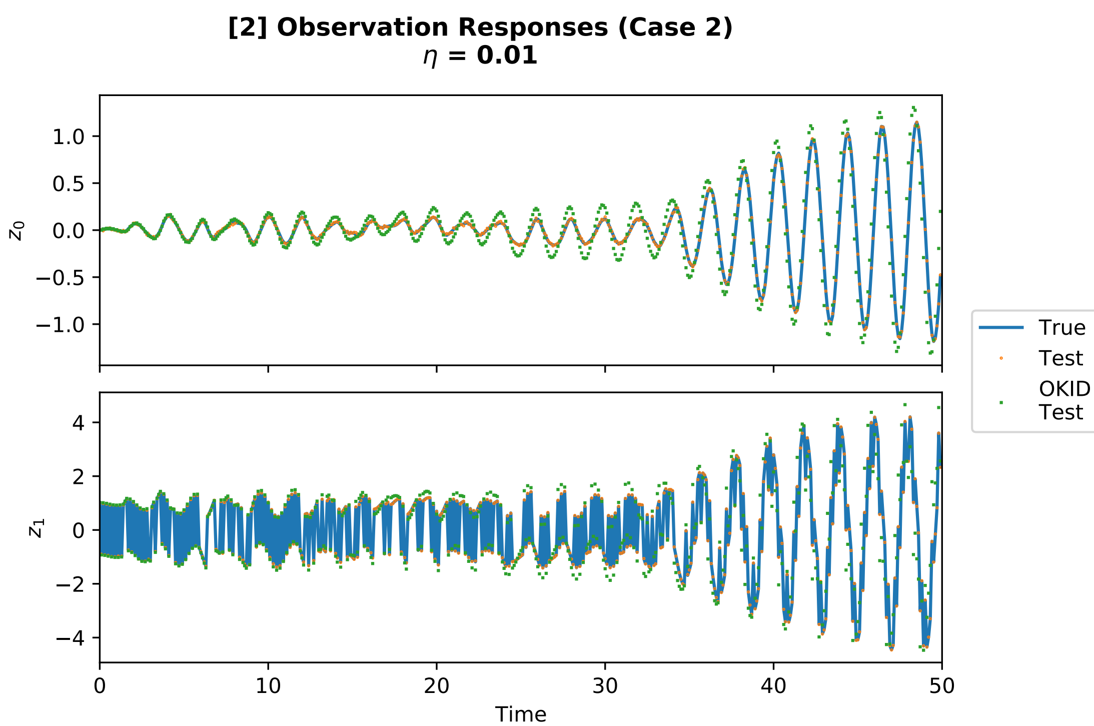


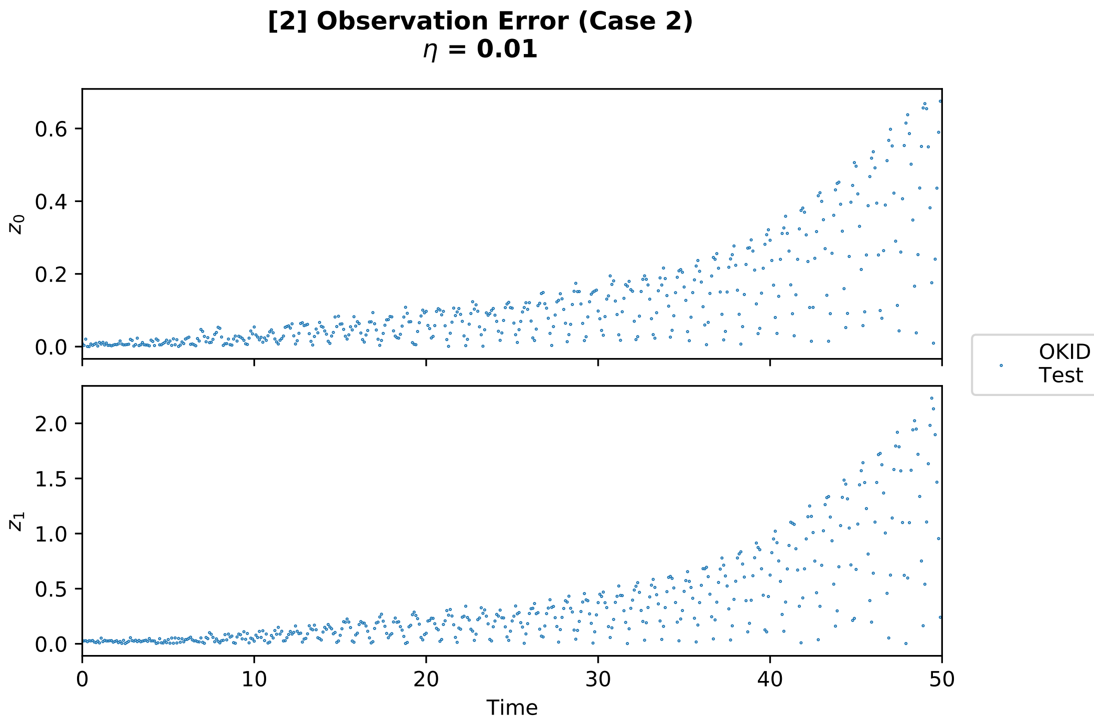
[2] Observation Error (Case 1)
η = 0.5
z_0
z_1
Time
OKID Train
OKID Test

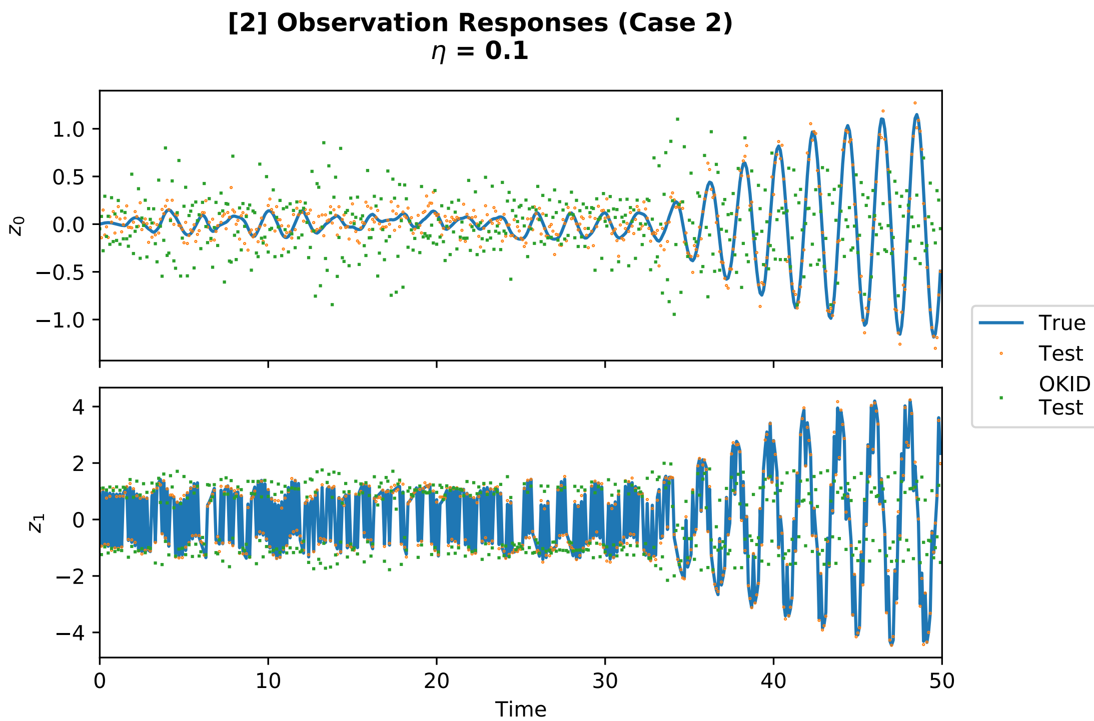


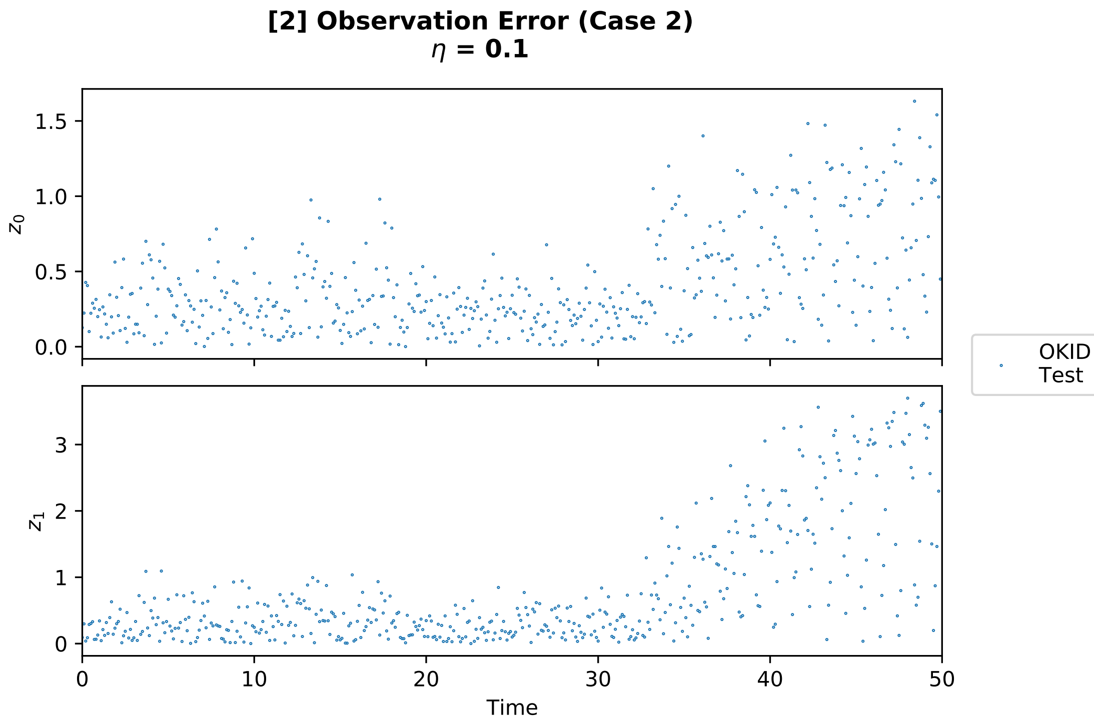
[2] Observation Responses (Case 2)
η = 0.001
z_0
z_1
Time
True
Test
OKID Test

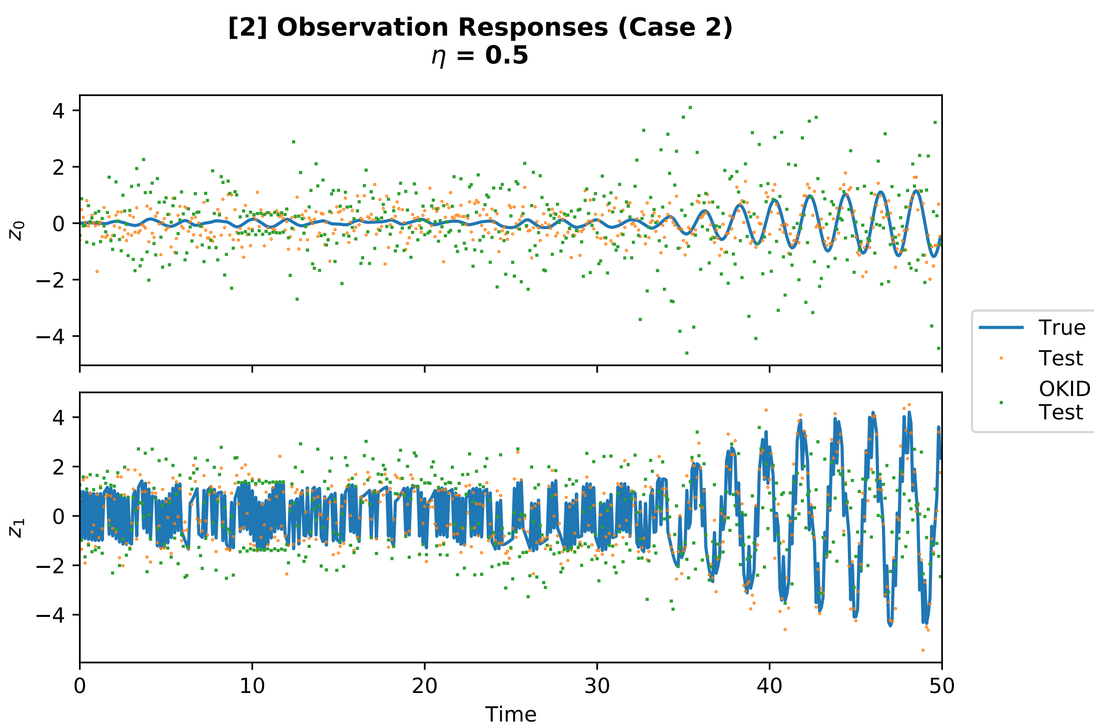

[2] Observation Error (Case 2)
η = 0.001
0.015
0.010
0.005
0.000
z_0
0.05
0.04
0.03
0.02
0.01
0.00
z_1
0
10
20
30
40
50
Time
OKID Test

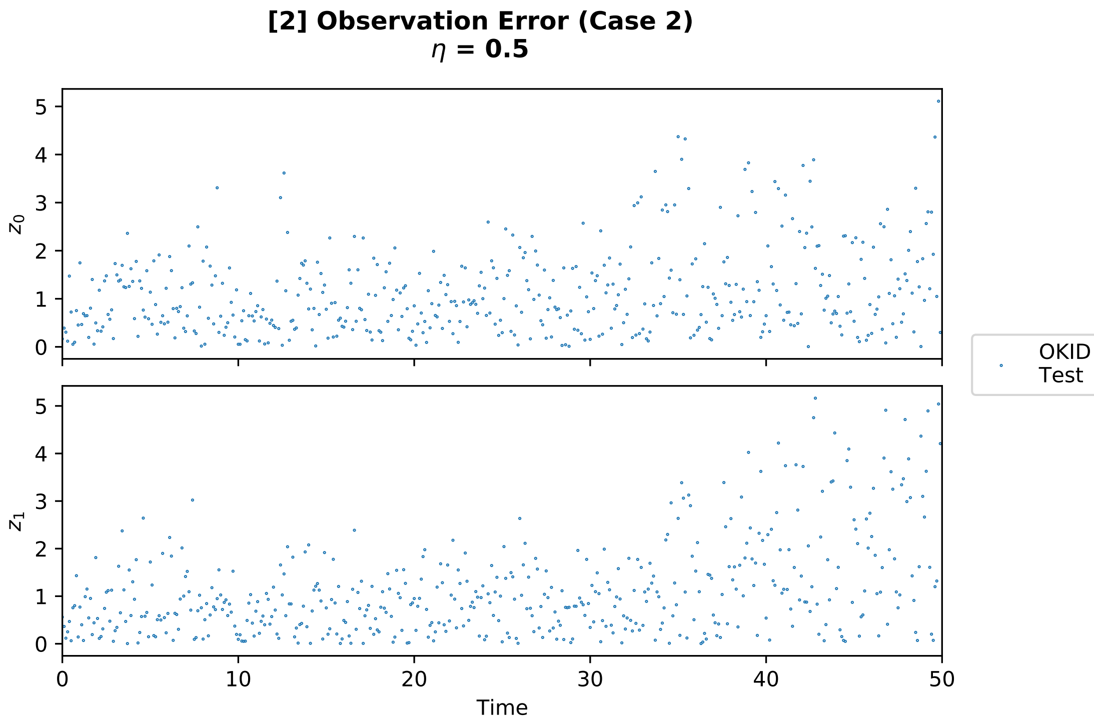

[2] Observation Responses (Case 2)
η = 0.01
1.0
0.5
0.0
−0.5
−1.0
z_0
4
2
0
−2
−4
z_1
0
10
20
30
40
50
Time
True
Test
OKID Test

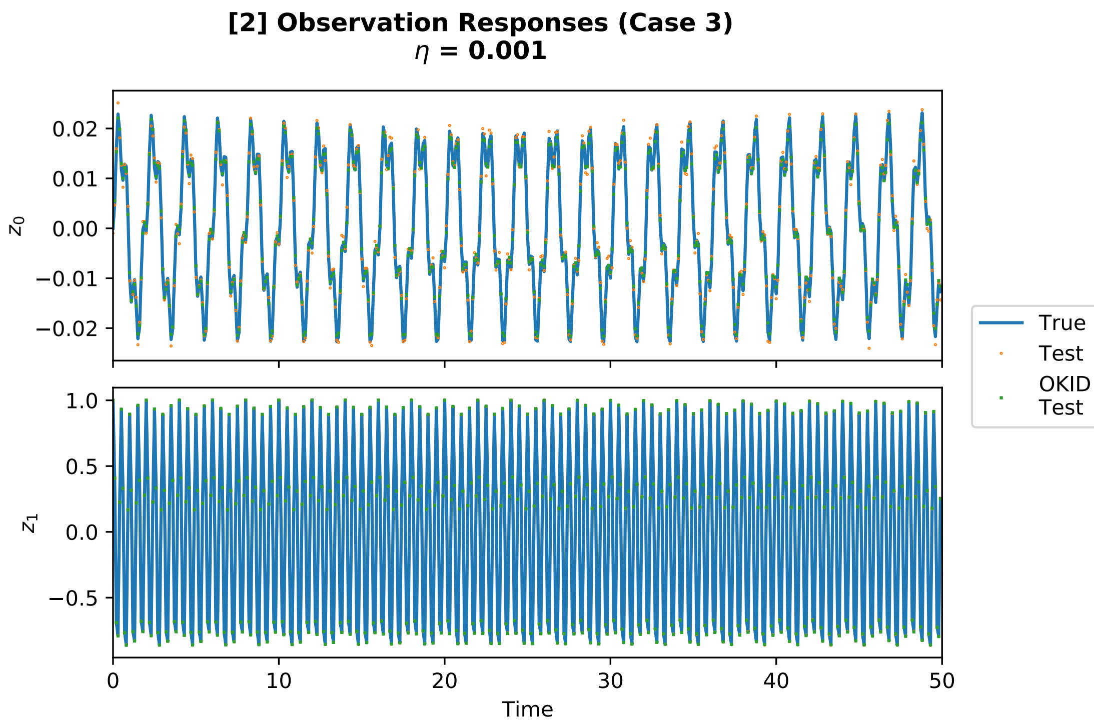

[2] Observation Error (Case 2)
η = 0.01
0.6
0.4
0.2
0.0
z_0
2.0
1.5
1.0
0.5
0.0
z_1
0
10
20
30
40
50
Time
OKID
Test

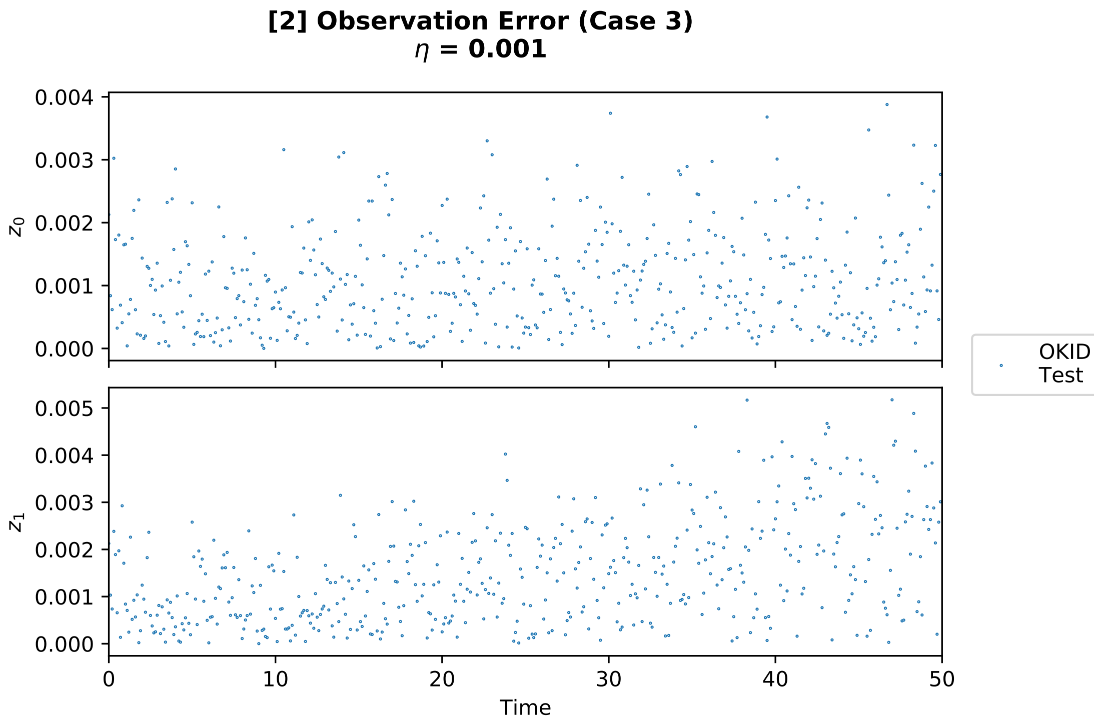

[2] Observation Responses (Case 2)
η = 0.1
1.0
0.5
0.0
−0.5
−1.0
z_0
4
2
0
−2
−4
z_1
0
10
20
30
40
50
Time
True
Test
OKID
Test

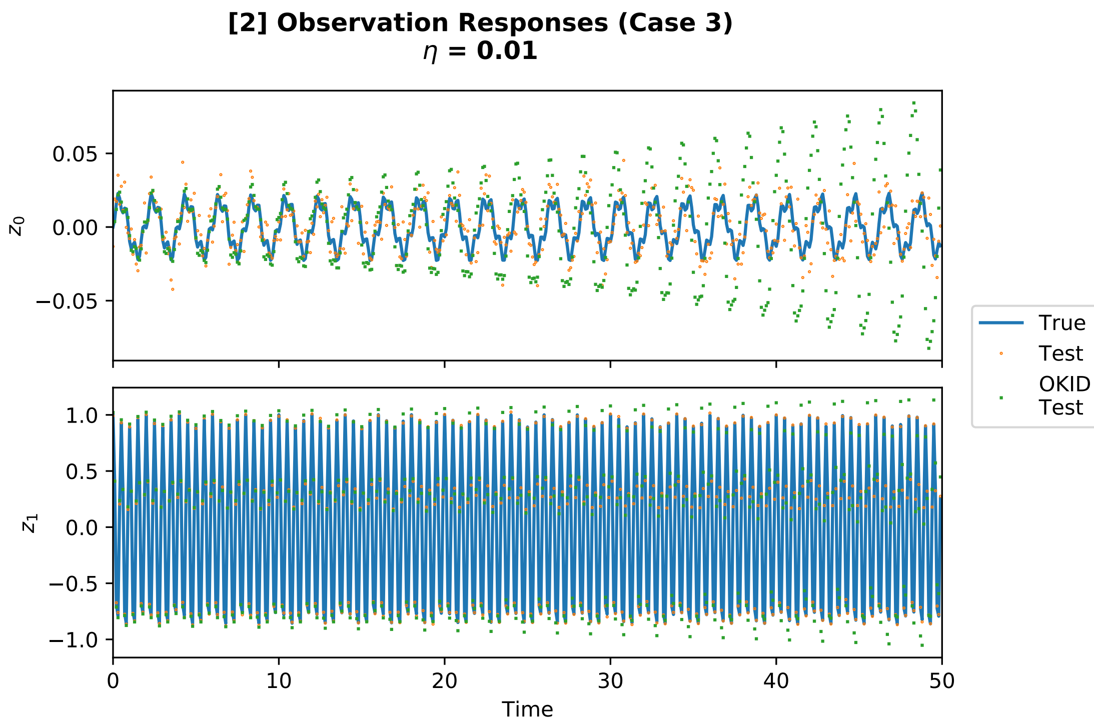

[2] Observation Error (Case 2)
η = 0.1
z_0
z_1
Time
OKID Test

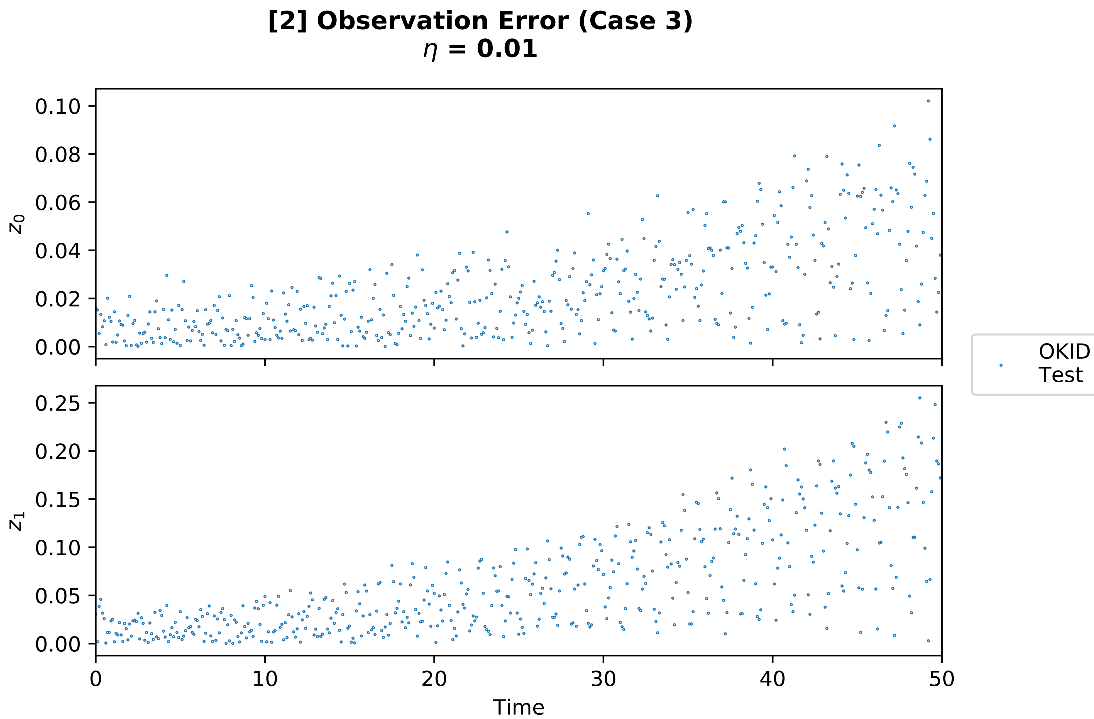

[2] Observation Responses (Case 2)
η = 0.5
z_0
z_1
Time
True
Test
OKID Test

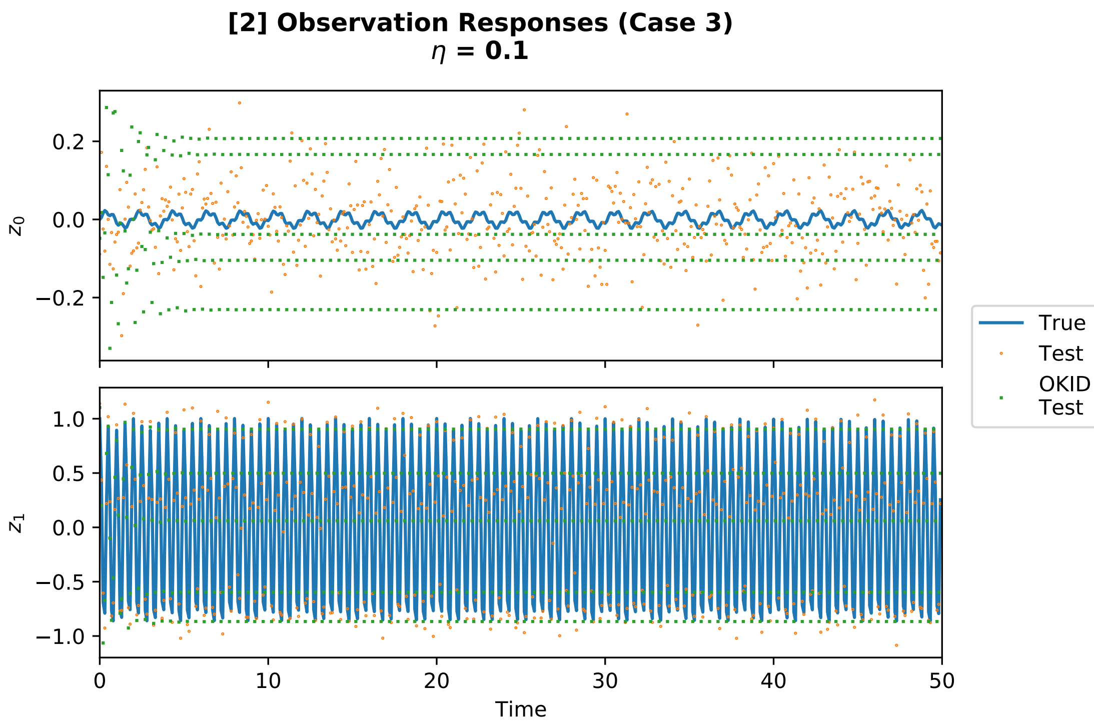

[2] Observation Error (Case 2)
η = 0.5
z0
z1
Time
OKID Test


[2] Observation Responses (Case 3)
η = 0.001
z0
z1
Time
True
Test
OKID Test

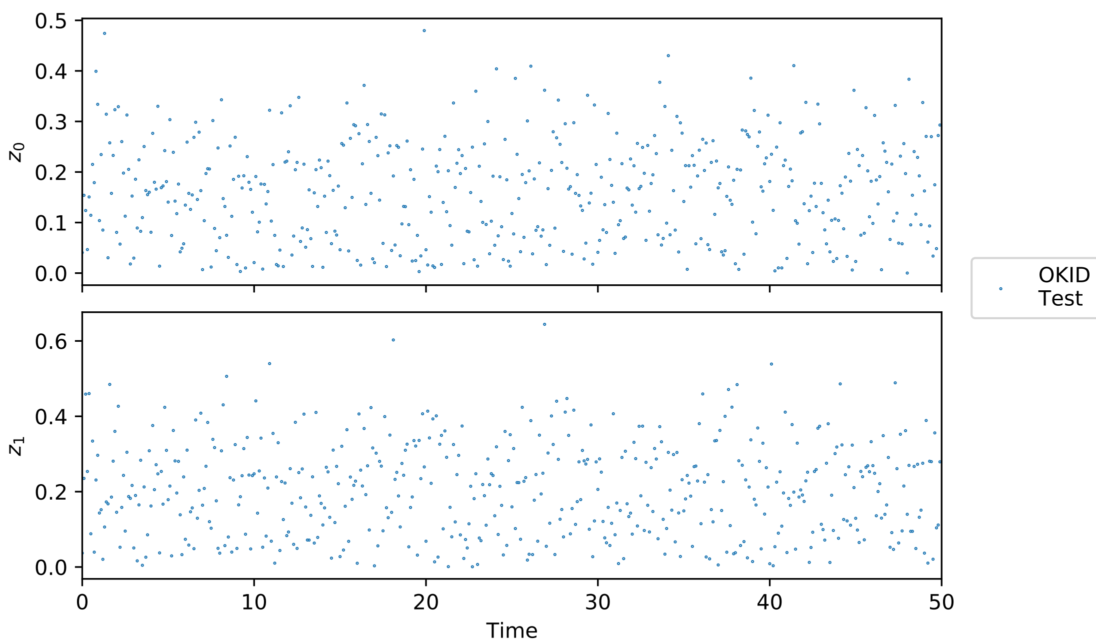

[2] Observation Error (Case 3)
η = 0.001
z_0
z_1
0.004
0.003
0.002
0.001
0.000
0.005
0.004
0.003
0.002
0.001
0.000
0
10
20
30
40
50
Time
OKID Test


[2] Observation Responses (Case 3)
η = 0.01
z_0
z_1
0.05
0.00
−0.05
1.0
0.5
0.0
−0.5
−1.0
0
10
20
30
40
50
Time
True
Test
OKID Test

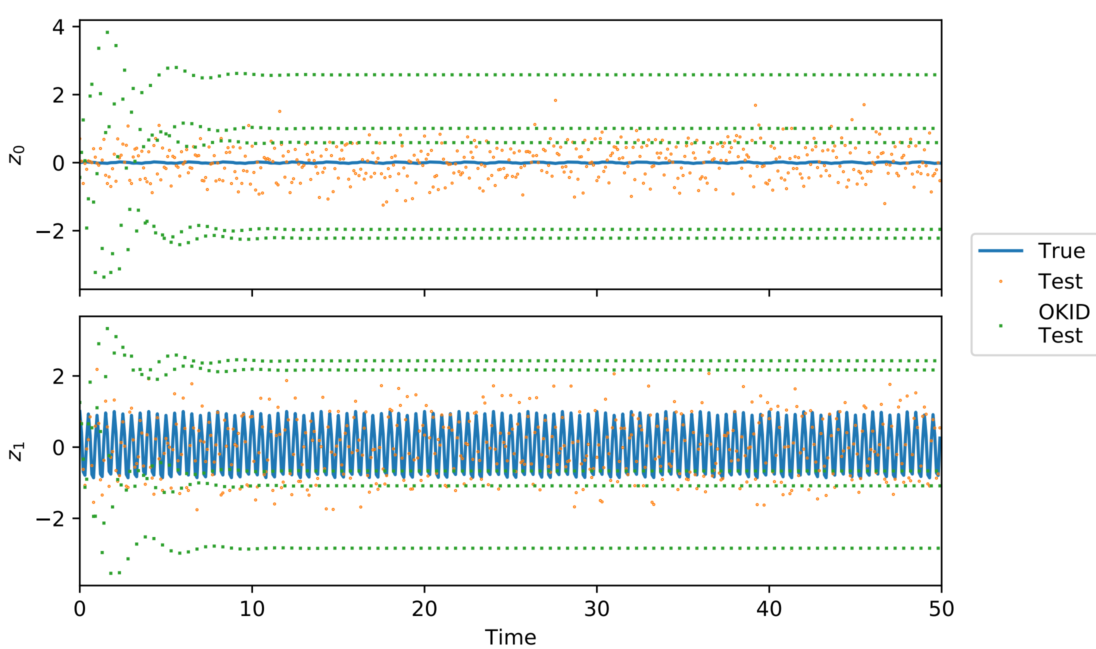

[2] Observation Error (Case 3)
η = 0.01
z_0
z_1
Time
OKID
Test

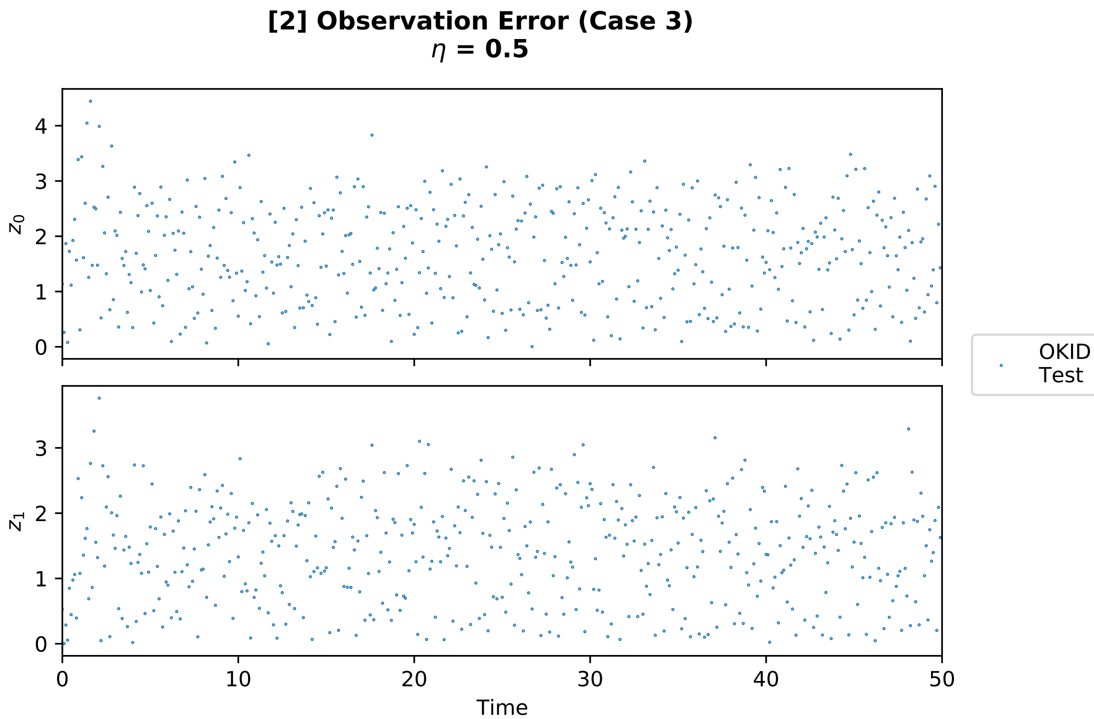

[2] Observation Responses (Case 3)
η = 0.1
z_0
z_1
Time
True
Test
OKID
Test

[2] Observation Error (Case 3)
η = 0.1



[2] Observation Responses (Case 3)
η = 0.5





- $\eta = 0.001$: The identification remains quite accurate, for both training and testing, in all cases.
- $\eta = 0.01$: The identification appears to be strong for the first few seconds of the simulation in each case, but falters eventually.
- $\eta > 0.01$: The identification fails entirely, which makes sense since the observation signal is dominated by the noise in such cases and the true observation data is nearly entirely obscured.