AERSP597 Midterm

Ani Perumalla

April 1, 2021

1 Q. #4

[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 = "4-2"
```

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

# Simulation dimensions
alphas = (5, 10, 20) # Number of block rows in Hankel matrices
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(alphas), 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(alphas), n, nt_train]) # Train state vector
X_test = np.zeros([len(alphas), cases, n, nt_test + 1]) # Test state vectors
Z_train = np.zeros([len(alphas), m, nt_train]) # Train observation vector
Z_test = np.zeros([len(alphas), cases, m, nt_test]) # Test observation vectors
V_train = np.zeros([len(alphas), r + m, nt_train]) # Train observation input␣
 ↪vectors
V_test = np.zeros([len(alphas), 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 = 45 # Order of OKID algorithm, number of Markov parameters to identify␣
 ↪after the zeroeth
beta = 5 # Number of block columns in Hankel matrices
n_era = 2 # Number of proposed states
```

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

Note that we have set $l_0 = 45$ and $\beta = 5$ for this simulation. We choose $\alpha = \{5, 10, 20\}$ to determine the effect of Hankel matrix height on the results of the system identification.

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

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

[5]:
```python
# Simulation
for i, j in it.product(range(cases), range(len(alphas))):
    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]
        # Identify System Markov parameters and Observer Gain Markov parameters
        Y_okid, Y_og_okid = \
            okid(Z_train[j], U_train,
                 l_0 = order, alpha = alphas[j], 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, alpha = alphas[j], 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.
↪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}}(\\alpha = {alphas[j]})", A_okid[j])
        etch(f"B_{{OKID}}(\\alpha = {alphas[j]})", B_okid[j])
        etch(f"C_{{OKID}}(\\alpha = {alphas[j]})", C_okid[j])
        etch(f"D_{{OKID}}(\\alpha = {alphas[j]})", D_okid[j])
        etch(f"G_{{OKID}}(\\alpha = {alphas[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}}(\\alpha = {alphas[j]})", eig_A_okid[j])
        etch(f"\hat{{\omega}}_{{n}}(\\alpha = {alphas[j]})", np.
↪abs(eig_A_okid[j]))
        etch(f"\hat{{\zeta}}(\\alpha = {alphas[j]})", -np.cos(np.
↪angle(eig_A_okid[j])))
    X_test[j, i], Z_test[j, i] = \
        X_sim[i], Z_sim[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): 4
Rank of H(1): 5
```

```
Rank of H(0): 4
Rank of H(1): 4
Rank of H(0): 2
Rank of H(1): 2
```

$$A_{OKID}(\alpha = 5) = \begin{bmatrix} 0.77841 & 0.2401 \\ -0.51805 & 1.12488 \end{bmatrix}$$

$$B_{OKID}(\alpha = 5) = \begin{bmatrix} -0.2934 \\ -0.18926 \end{bmatrix}$$

$$C_{OKID}(\alpha = 5) = \begin{bmatrix} -0.07136 & 0.08442 \\ -0.31887 & -0.0255 \end{bmatrix}$$

$$D_{OKID}(\alpha = 5) = \begin{bmatrix} 0.0 \\ 1.0 \end{bmatrix}$$

$$G_{OKID}(\alpha = 5) = \begin{bmatrix} -0.02121 & 0.24057 \\ -0.10692 & 0.12472 \end{bmatrix}$$

$$\hat{\lambda}(\alpha = 5) = \begin{bmatrix} 3.1225i \\ -3.1225i \end{bmatrix}$$

$$\hat{\omega}_n(\alpha = 5) = \begin{bmatrix} 3.1225 \\ 3.1225 \end{bmatrix}$$

$$\hat{\zeta}(\alpha = 5) = \begin{bmatrix} 0.0 \\ 0.0 \end{bmatrix}$$

$$A_{OKID}(\alpha = 10) = \begin{bmatrix} 0.95311 & -0.20813 \\ 0.45344 & 0.95018 \end{bmatrix}$$

$$B_{OKID}(\alpha = 10) = \begin{bmatrix} -0.27391 \\ 0.29319 \end{bmatrix}$$

$$C_{OKID}(\alpha = 10) = \begin{bmatrix} -0.06627 & -0.045 \\ -0.20839 & 0.14087 \end{bmatrix}$$

$$D_{OKID}(\alpha = 10) = \begin{bmatrix} 0.0 \\ 1.0 \end{bmatrix}$$

$$G_{OKID}(\alpha = 10) = \begin{bmatrix} 0.03373 & 0.24208 \\ 0.11727 & -0.20901 \end{bmatrix}$$

$$\hat{\lambda}(\alpha = 10) = \begin{bmatrix} 3.1225i \\ -3.1225i \end{bmatrix}$$

$$\hat{\omega}_n(\alpha = 10) = \begin{bmatrix} 3.1225 \\ 3.1225 \end{bmatrix}$$

$$\hat{\zeta}(\alpha = 10) = \begin{bmatrix} 0.0 \\ 0.0 \end{bmatrix}$$

$$A_{OKID}(\alpha = 20) = \begin{bmatrix} 0.9531 & -0.20812 \\ 0.45346 & 0.95019 \end{bmatrix}$$

$$B_{OKID}(\alpha = 20) = \begin{bmatrix} -0.32574 \\ 0.34867 \end{bmatrix}$$

$$C_{OKID}(\alpha = 20) = \begin{bmatrix} -0.05573 & -0.03784 \\ -0.17524 & 0.11845 \end{bmatrix}$$

$$D_{OKID}(\alpha = 20) = \begin{bmatrix} 0.0 \\ 1.0 \end{bmatrix}$$

$$G_{OKID}(\alpha = 20) = \begin{bmatrix} 0.04011 & 0.28788 \\ 0.13946 & -0.24855 \end{bmatrix}$$

$$\hat{\lambda}(\alpha = 20) = \begin{bmatrix} 3.1225i \\ -3.1225i \end{bmatrix}$$

$$\hat{\omega}_n(\alpha = 20) = \begin{bmatrix} 3.1225 \\ 3.1225 \end{bmatrix}$$

$$\hat{\zeta}(\alpha = 20) = \begin{bmatrix} 0.0 \\ 0.0 \end{bmatrix}$$

In the absence of noise, regardless of the order selected, the eigenvalues, natural frequencies, and the system as a whole are able to be identified essentially perfectly. The realizations are slightly different numerically, but as shown visually, the results are essentially the same.

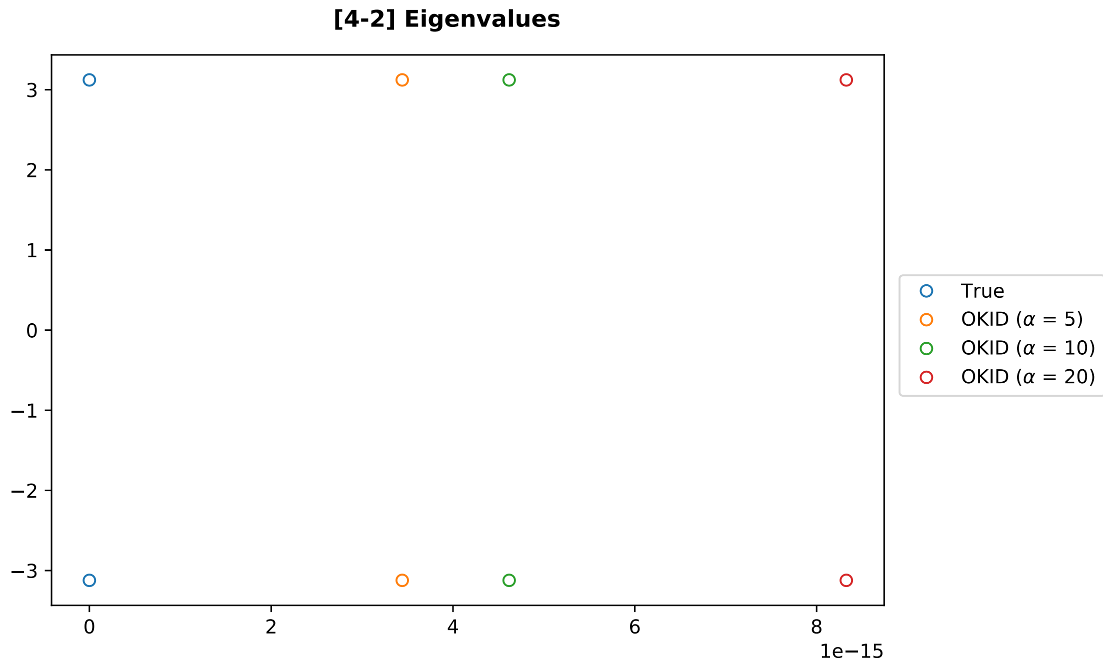
[6]:
```
RMS_train = np.zeros([len(alphas), m])
RMS_test = np.zeros([len(alphas), cases, m])
for j in range(len(alphas)):
    RMS_train[j] = np.sqrt(np.mean((Z_okid_train[j] - Z_train[j])**2, axis = 1))
    print(f"RMS Error of sim. for system found via OKID for train data, alpha =␣
↪{alphas[j]}: {RMS_train[j]}")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

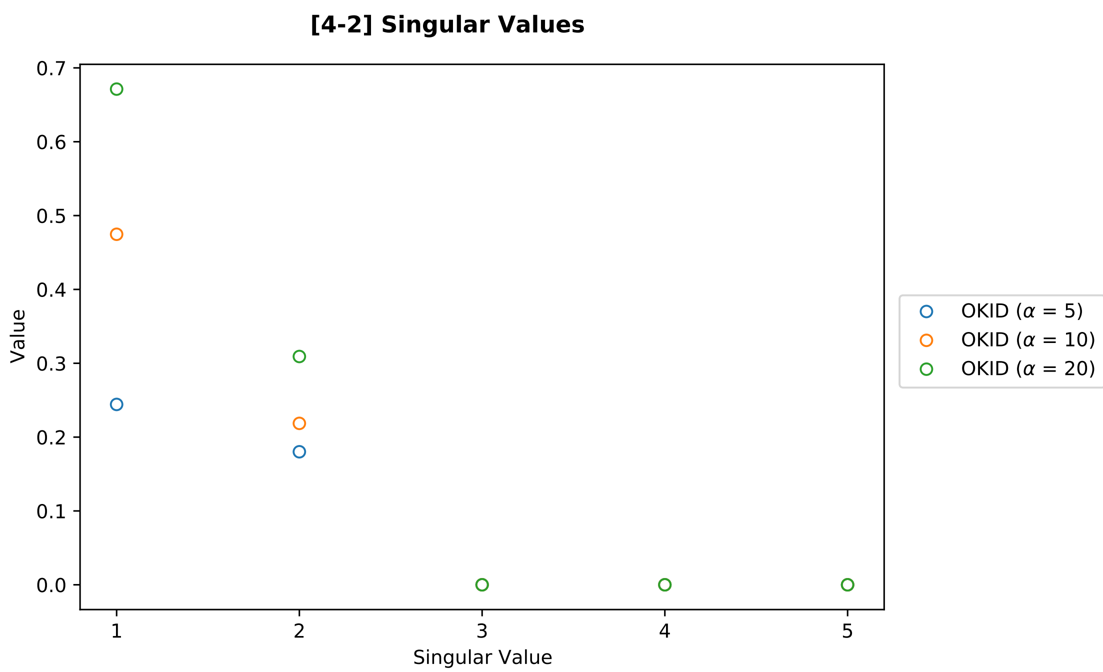
```
RMS Error of sim. for system found via OKID for train data, alpha = 5:
[1.40514174e-14 4.23549696e-14]
RMS Error of sim. for system found via OKID for test data, alpha = 5, case 0:
[4.39729475e-14 1.37597663e-13]
RMS Error of sim. for system found via OKID for test data, alpha = 5, case 1:
[1.3813405e-13 4.2306013e-13]
RMS Error of sim. for system found via OKID for test data, alpha = 5, case 2:
[1.39473264e-14 4.37100348e-14]
RMS Error of sim. for system found via OKID for train data, alpha = 10:
[1.74191125e-15 4.18464686e-15]
RMS Error of sim. for system found via OKID for test data, alpha = 10, case 0:
[5.20732732e-15 1.44675269e-14]
RMS Error of sim. for system found via OKID for test data, alpha = 10, case 1:
[1.65406691e-14 3.95777401e-14]
```

```
RMS Error of sim. for system found via OKID for test data, alpha = 10, case 2:
[1.64914885e-15 4.63060369e-15]
RMS Error of sim. for system found via OKID for train data, alpha = 20:
[3.32558258e-15 9.65804903e-15]
RMS Error of sim. for system found via OKID for test data, alpha = 20, case 0:
[1.04803050e-14 3.18994544e-14]
RMS Error of sim. for system found via OKID for test data, alpha = 20, case 1:
[3.14195495e-14 9.69634848e-14]
RMS Error of sim. for system found via OKID for test data, alpha = 20, case 2:
[3.33938205e-15 1.01005250e-14]
```

The RMS error for the test cases is essentially zero regardless of Hankel height.

[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(alphas)):
    ax.plot(np.real(eig_A_okid[j]), np.imag(eig_A_okid[j]),
            "o", mfc = "None")

fig.legend(labels = ("True", *[f"OKID ($\\alpha$ = {q})" for q in alphas]),
           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(alphas)):
    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 ($\\alpha$ = {q})" for q in alphas],
           bbox_to_anchor = (1, 0.5), loc = 6)
fig.savefig(figs_dir / f"midterm_{prob}_singval.pdf",
            bbox_inches = "tight")
```



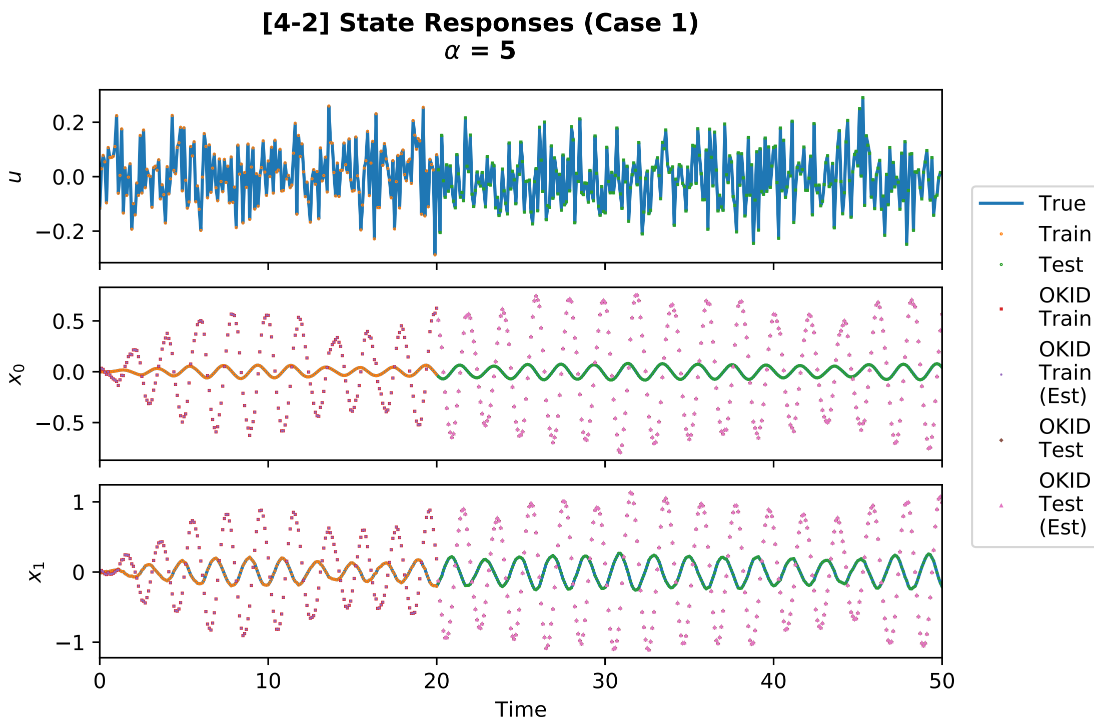


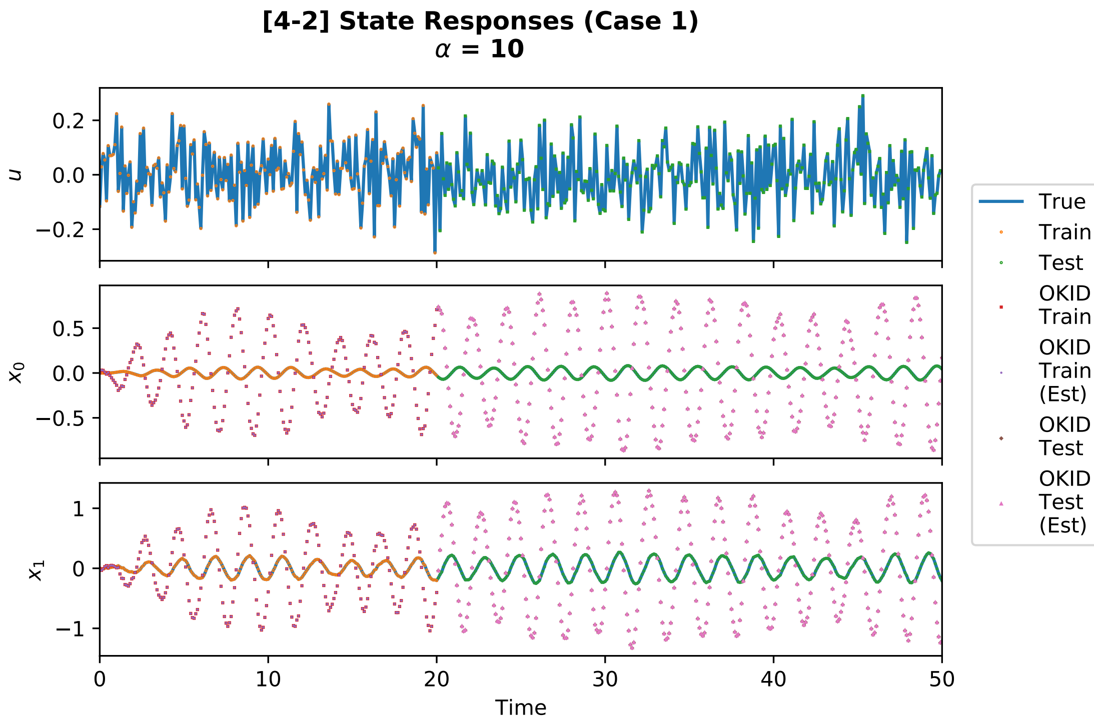
The main impact of the Hankel height is on the singular values. Although in this case the realization wasn't affected, increasing α led the singular values to become more disjointed, allowing us to more easily identify the order of the underlying system. As seen in the plot above, the difference between the 1st and 2nd singular values drastically increased for higher α, and the difference between the

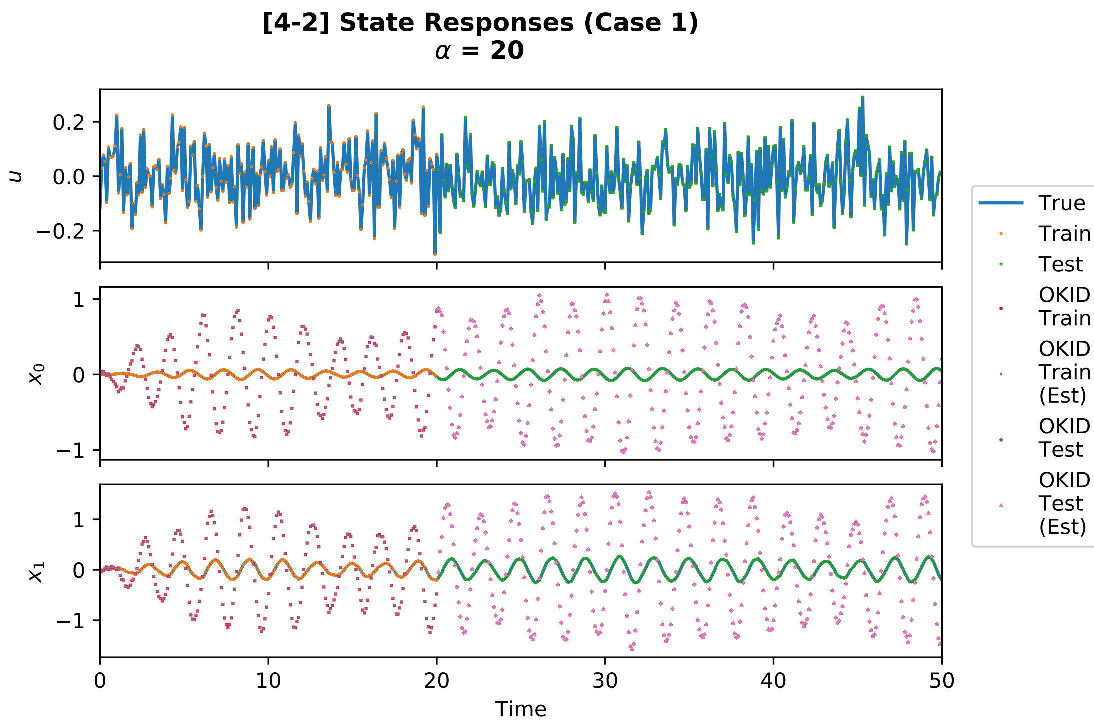
2nd and 3rd singular values slighly increased for higher α.

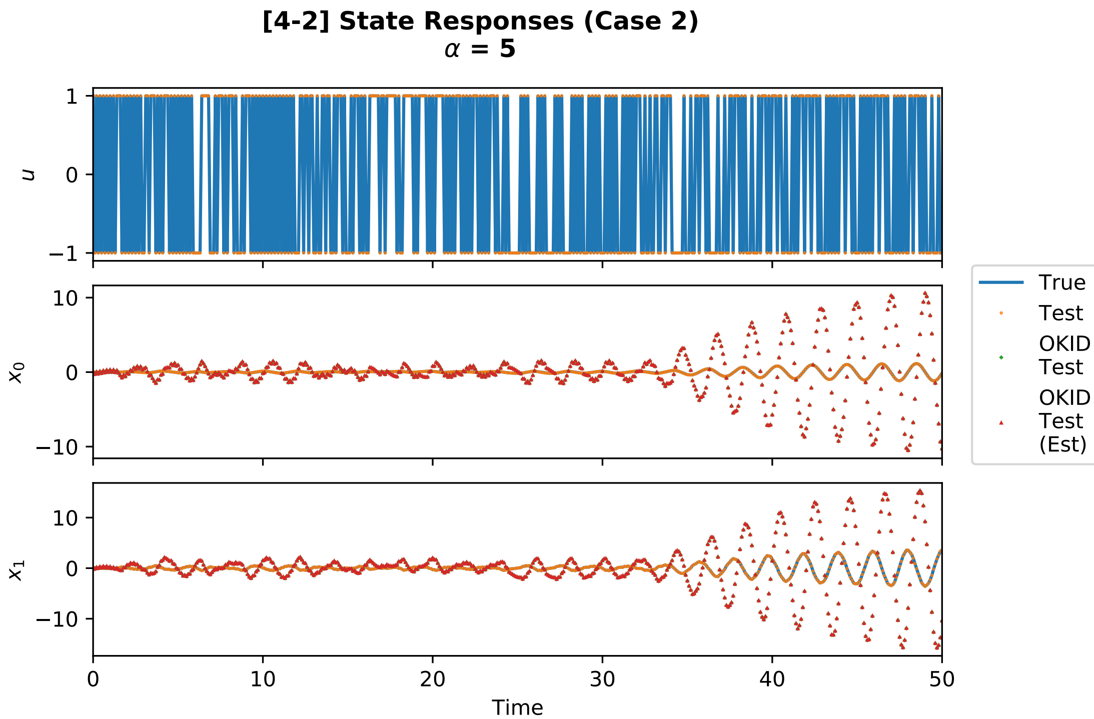
[8]:
```
# Response plots
ms = 0.5 # Marker size
for i, k in it.product(range(cases), range(len(alphas))):
    fig, axs = plt.subplots(1 + n, 1,
                            sharex = "col",
                            constrained_layout = True) # type:figure.Figure
    fig.suptitle(f"[{prob}] State Responses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

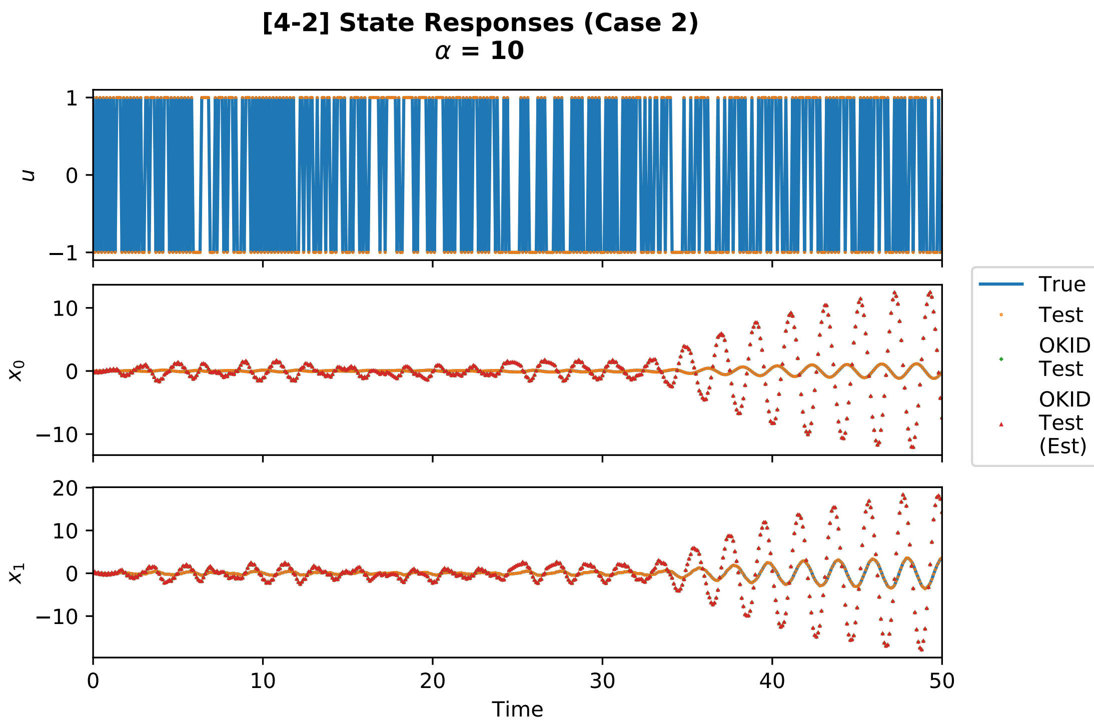
```
                    "o", ms = ms, mfc = "None")
        plt.setp(axs[0], ylabel = f"$u$", xlim = [0, t_max])

        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}_alpha{k}.pdf",
                bbox_inches = "tight")
```

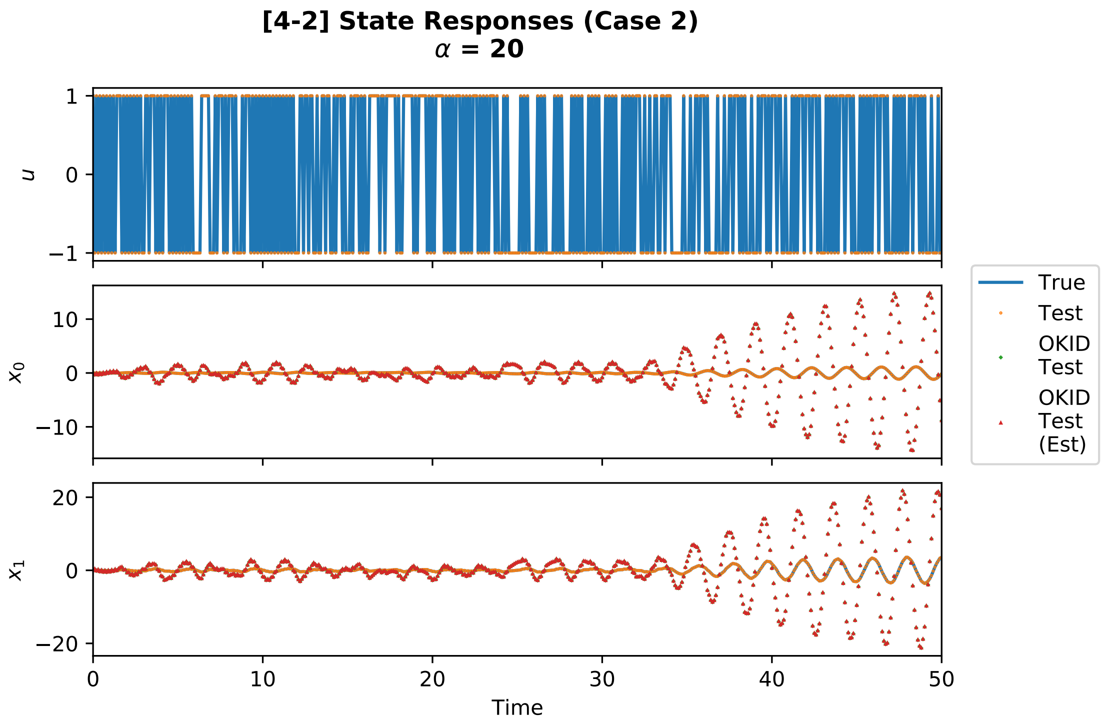


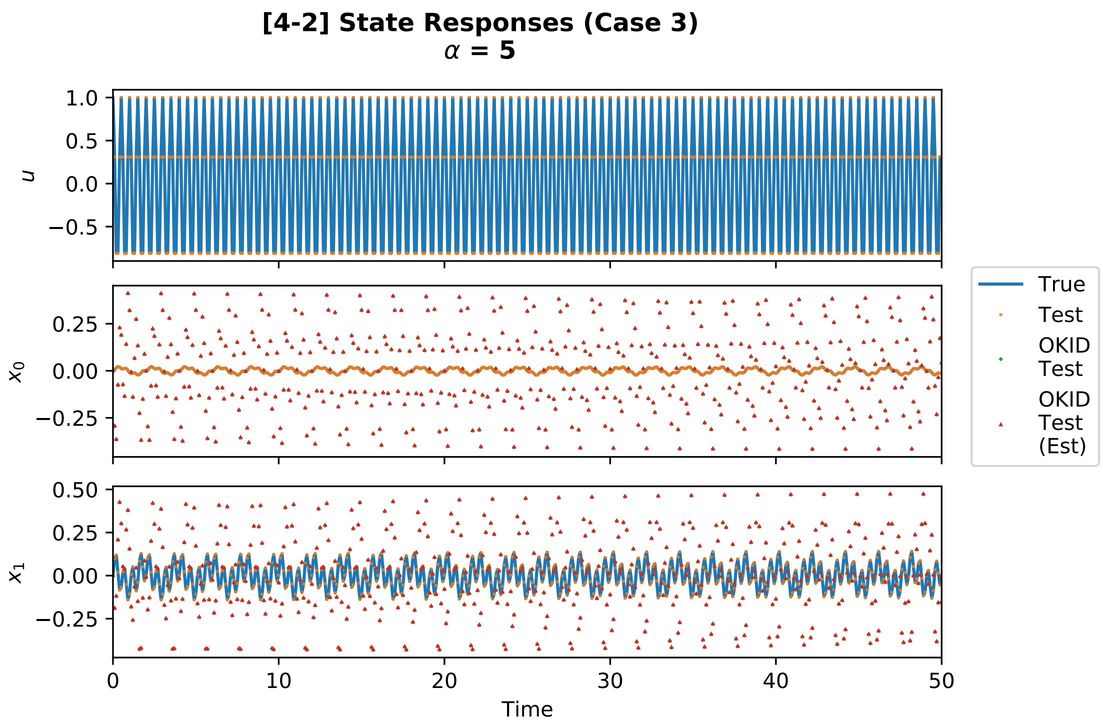

[4-2] State Responses (Case 1)
α = 10
u
x0
x1
Time
True
Train
Test
OKID Train
OKID Train (Est)
OKID Test
OKID Test (Est)

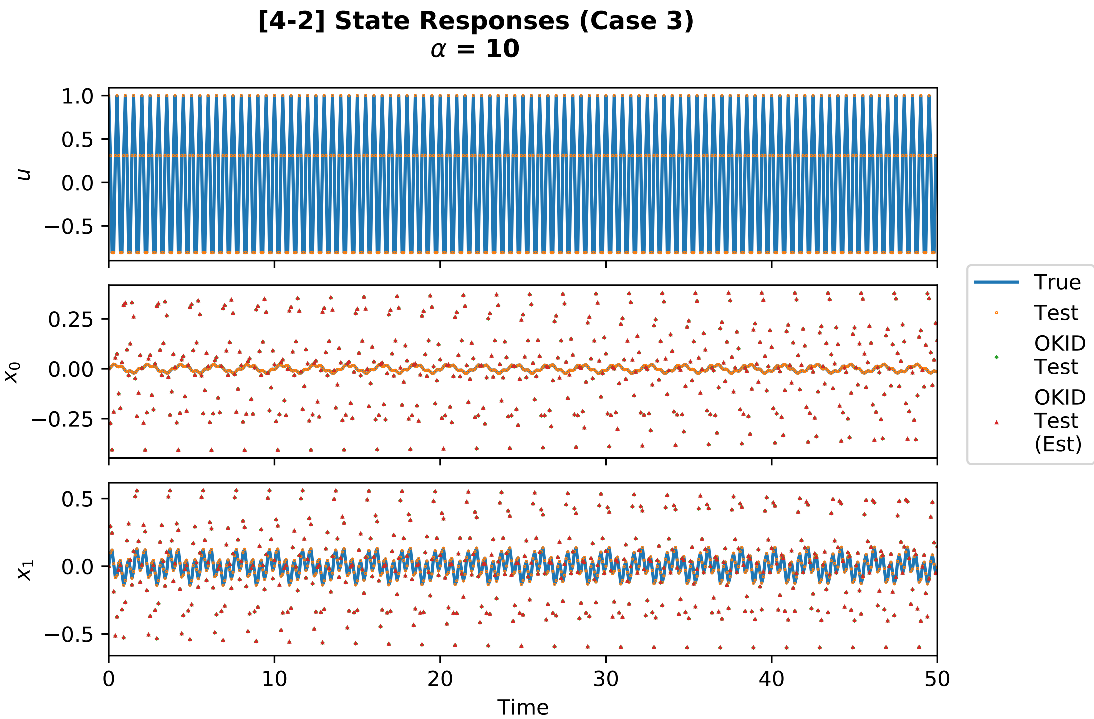

[4-2] State Responses (Case 1)
α = 20
u
x0
x1
Time
True
Train
Test
OKID Train
OKID Train (Est)
OKID Test
OKID Test (Est)

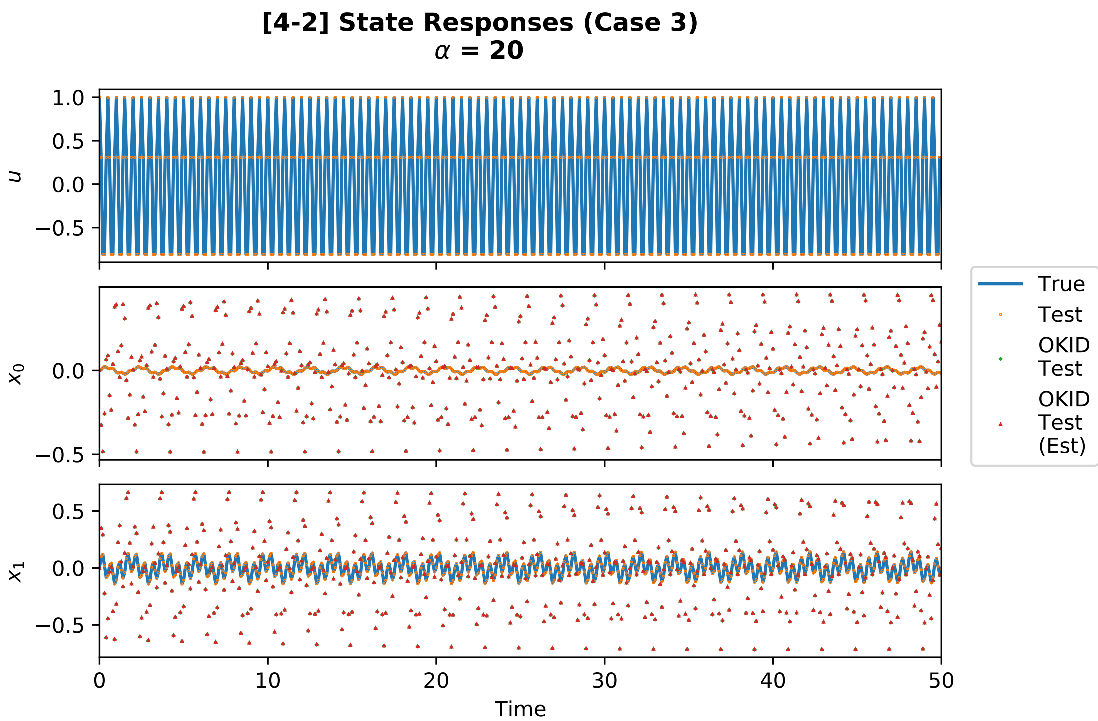

[4-2] State Responses (Case 2)
α = 5
u
x0
x1
Time
True
Test
OKID Test
OKID Test (Est)


[4-2] State Responses (Case 2)
α = 10
u
x0
x1
Time
True
Test
OKID Test
OKID Test (Est)


[4-2] State Responses (Case 2)
α = 20
u
x0
x1
Time
True
Test
OKID
Test
OKID
Test
(Est)


[4-2] State Responses (Case 3)
α = 5
u
x0
x1
Time
True
Test
OKID
Test
OKID
Test
(Est)

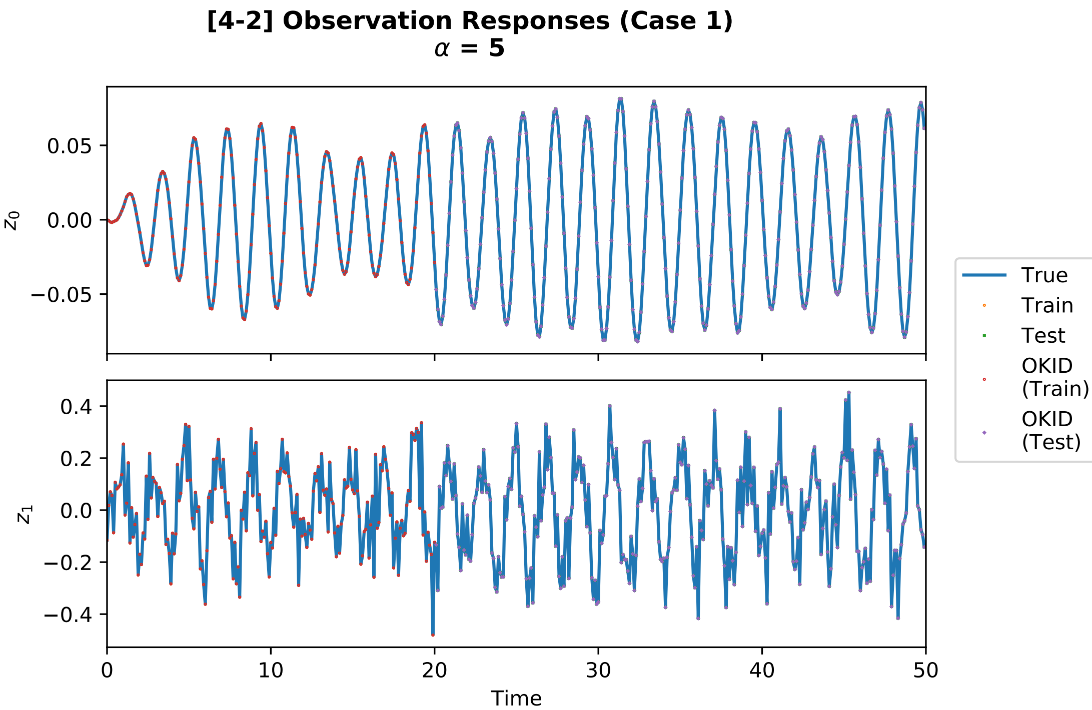

[4-2] State Responses (Case 3)
α = 10
u
x0
x1
Time
True
Test
OKID Test
OKID Test (Est)

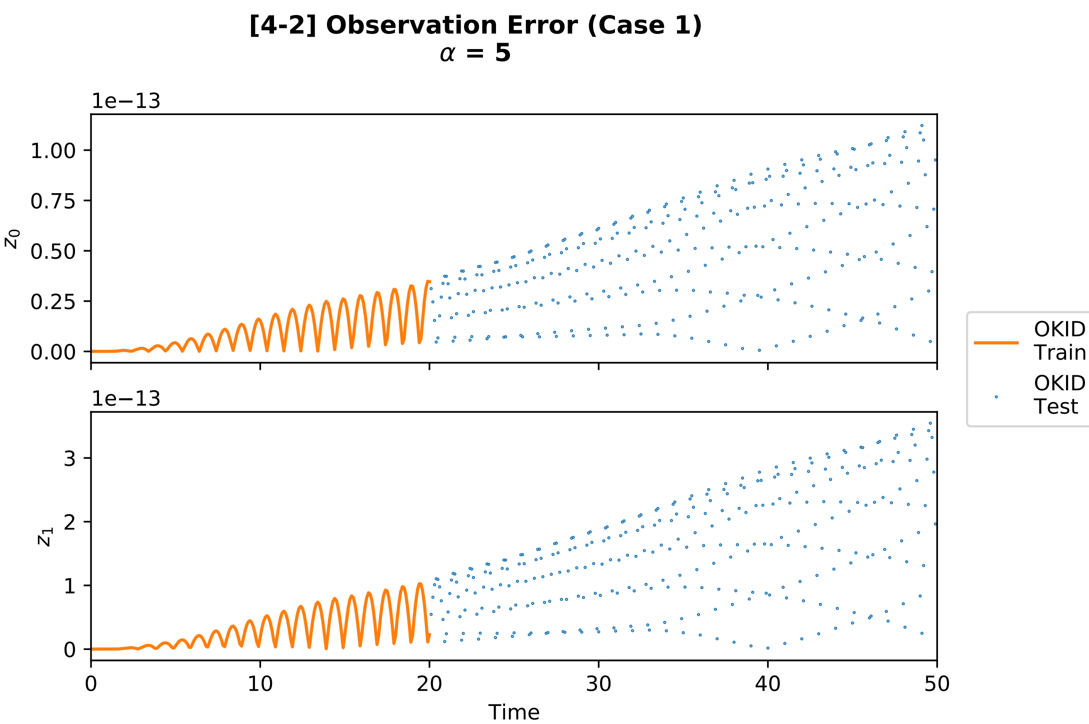

[4-2] State Responses (Case 3)
α = 20
u
x0
x1
Time
True
Test
OKID Test
OKID Test (Est)

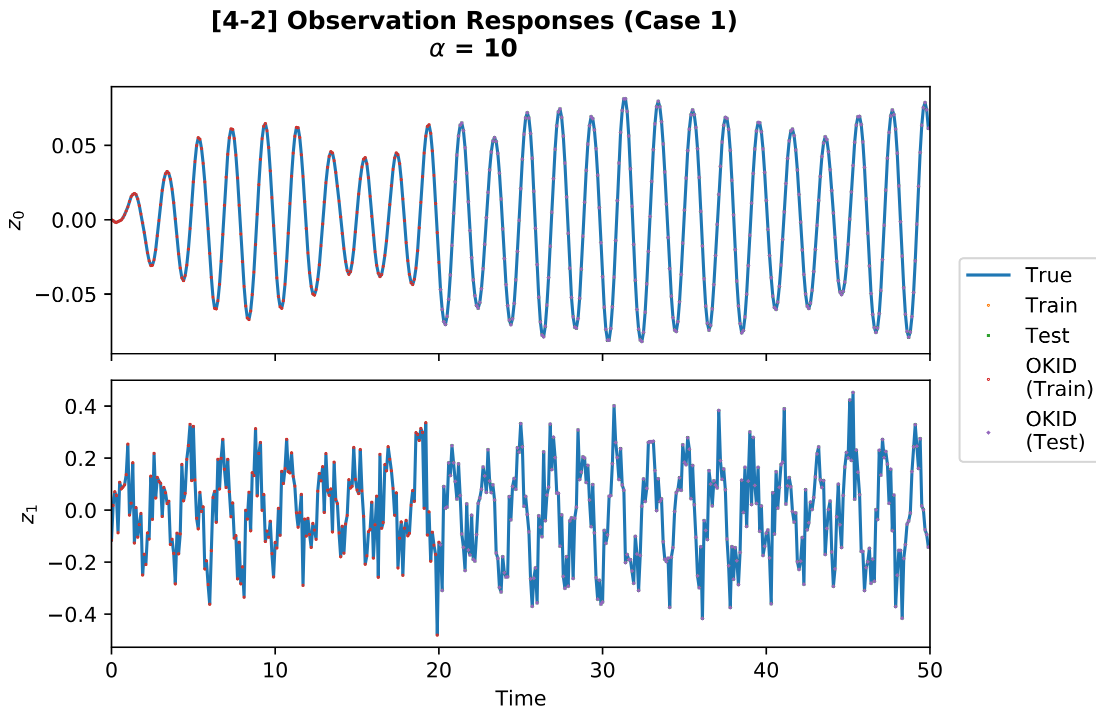
The choice of Hankel height did not affect the (already highly accurate) state estimation in this case.

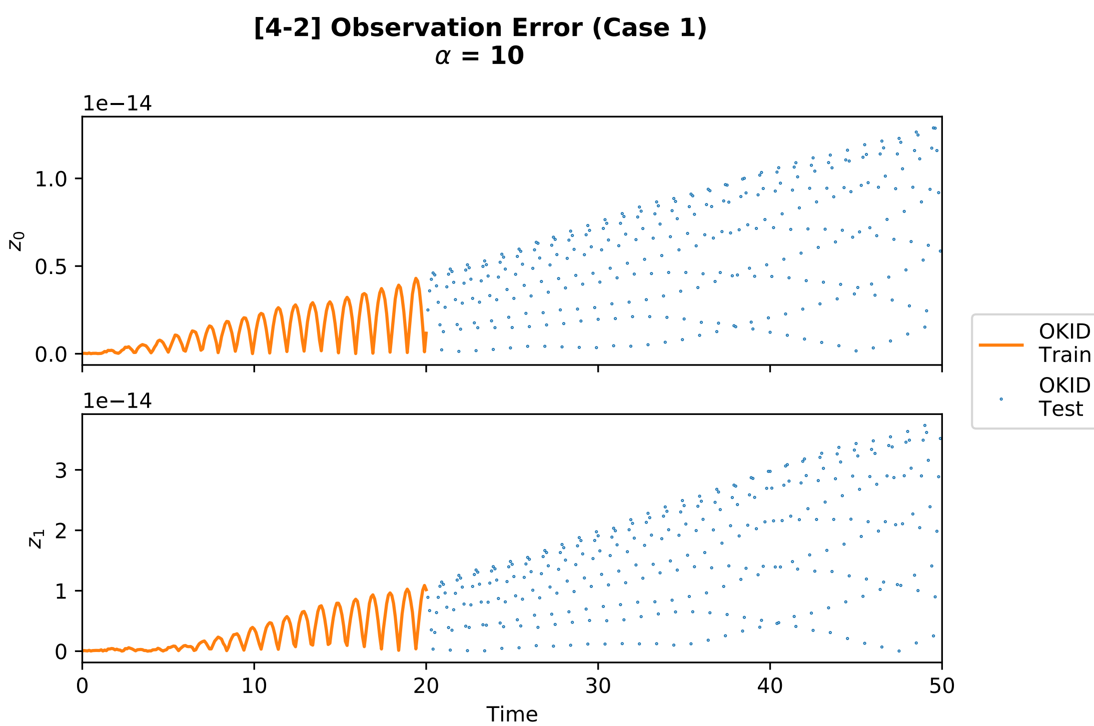
[9]:
```
# Observation plots
for i, k in it.product(range(cases), range(len(alphas))):
    # Raw observations
    raw_fig, axs = plt.subplots(m, 1,
                                sharex = "col", constrained_layout = True) #␣
↪type:figure.Figure
    raw_fig.suptitle(f"[{prob}] Observation Responses (Case {i + 1})\n$\\alpha$␣
↪= {alphas[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:],
                        "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}_alpha{k}.pdf",
                    bbox_inches = "tight")
```

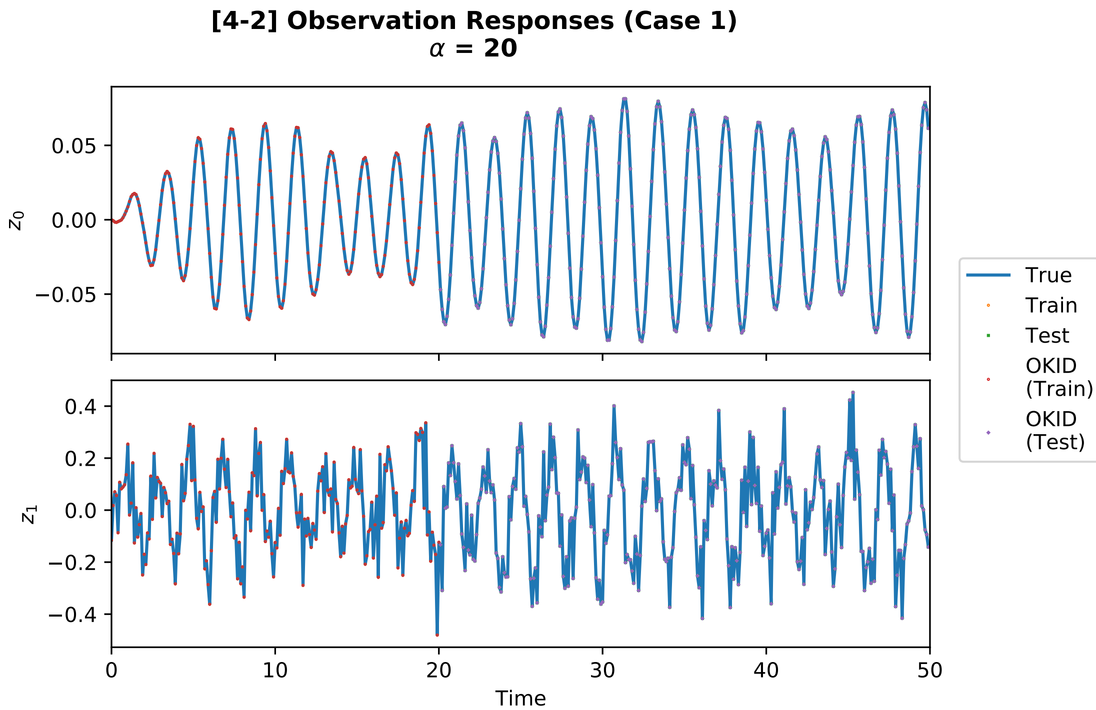
```
    # Observation error
    err_fig, axs = plt.subplots(m, 1,
                                sharex = "col", constrained_layout = True) #␣
↪type:figure.Figure
    err_fig.suptitle(f"[{prob}] Observation Error (Case {i + 1})\n$\\alpha$ =␣
↪{alphas[k]}",
                     fontweight = "bold")
    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}_alpha{k}.
↪pdf",
                    bbox_inches = "tight")
```

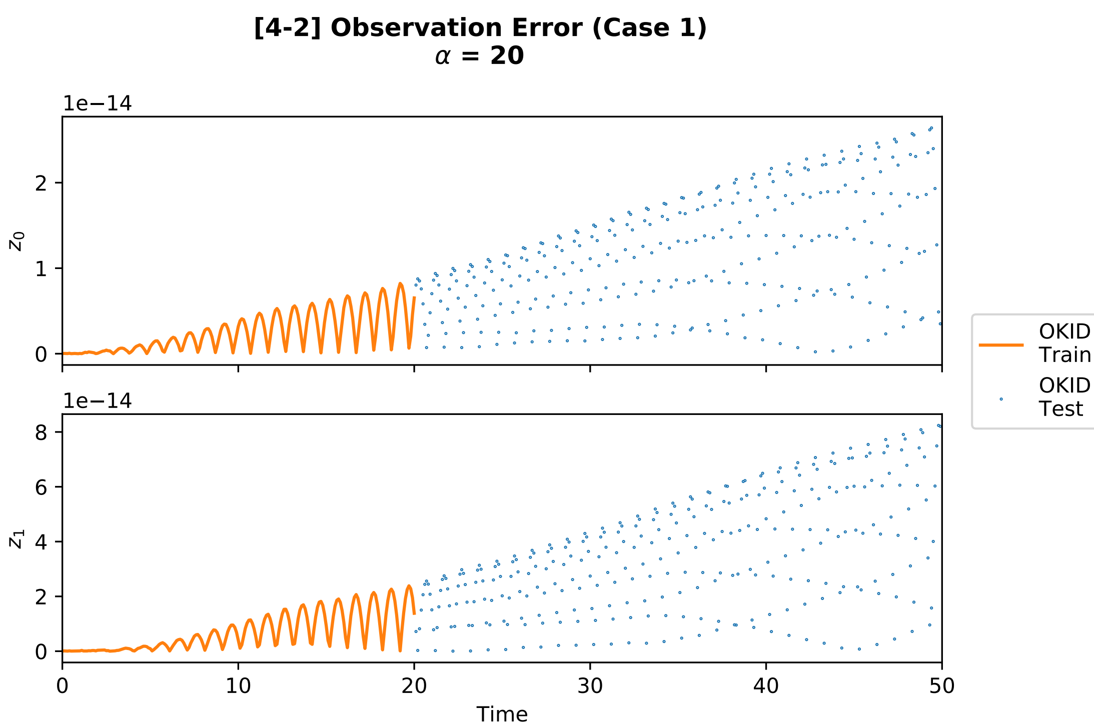

[4-2] Observation Responses (Case 1)
α = 5
z0
z1
Time
True
Train
Test
OKID (Train)
OKID (Test)

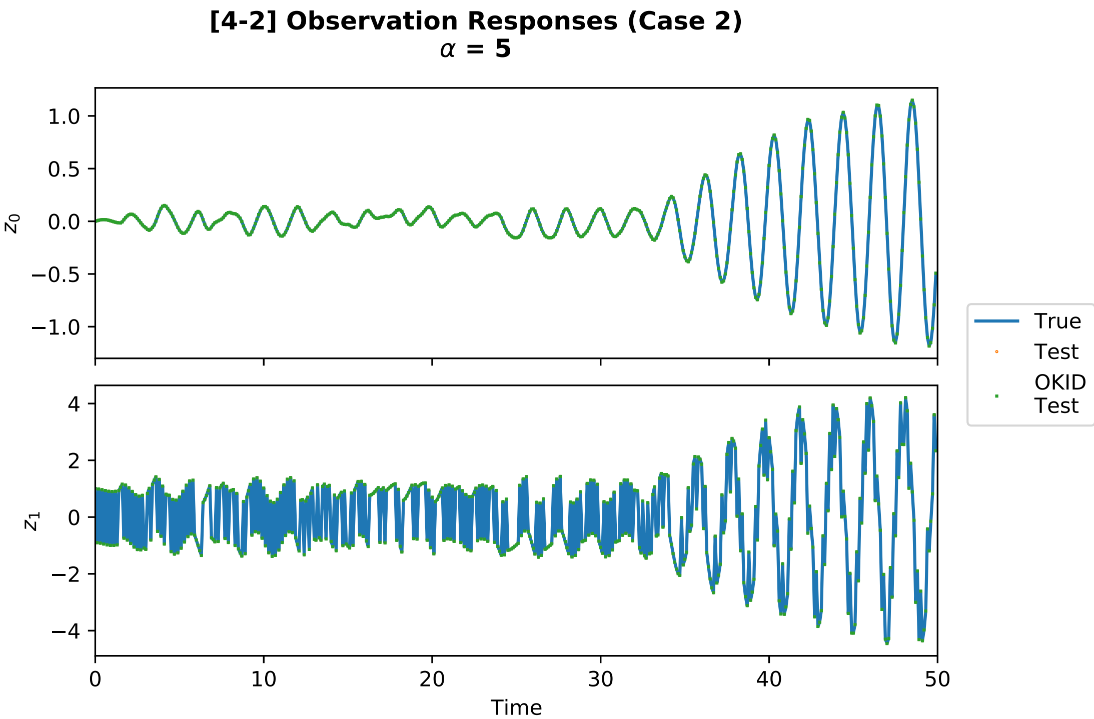

[4-2] Observation Error (Case 1)
α = 5
1e−13
z0
z1
Time
OKID Train
OKID Test

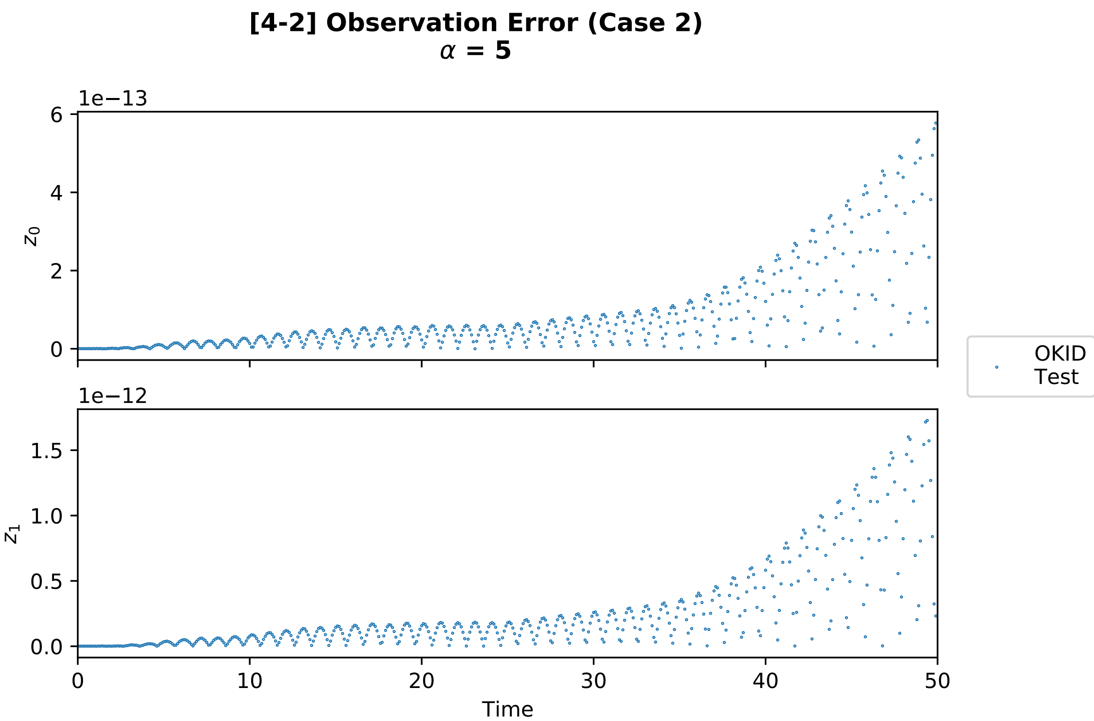

[4-2] Observation Responses (Case 1)
α = 10
0.05
0.00
−0.05
z_0
0.4
0.2
0.0
−0.2
−0.4
z_1
0
10
20
30
40
50
Time
True
Train
Test
OKID (Train)
OKID (Test)

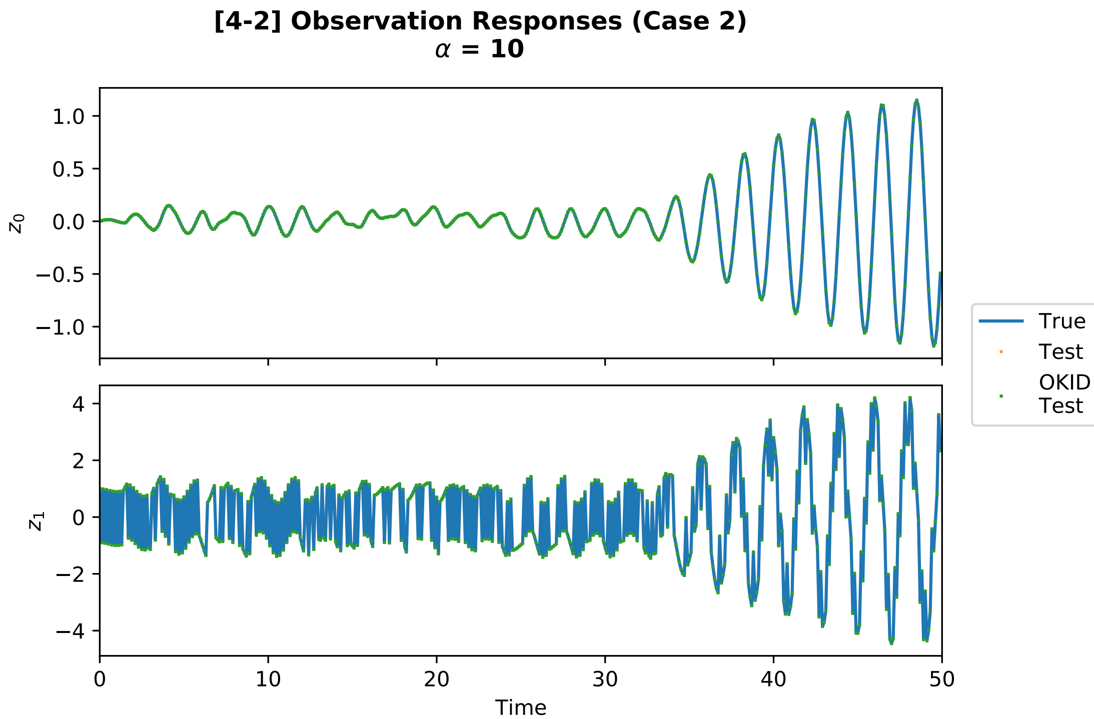

[4-2] Observation Error (Case 1)
α = 10
1e−14
1.0
0.5
0.0
z_0
1e−14
3
2
1
0
z_1
0
10
20
30
40
50
Time
OKID Train
OKID Test


[4-2] Observation Responses (Case 1)
α = 20
0.05
0.00
−0.05
z_0
0.4
0.2
0.0
−0.2
−0.4
z_1
0
10
20
30
40
50
Time
True
Train
Test
OKID (Train)
OKID (Test)

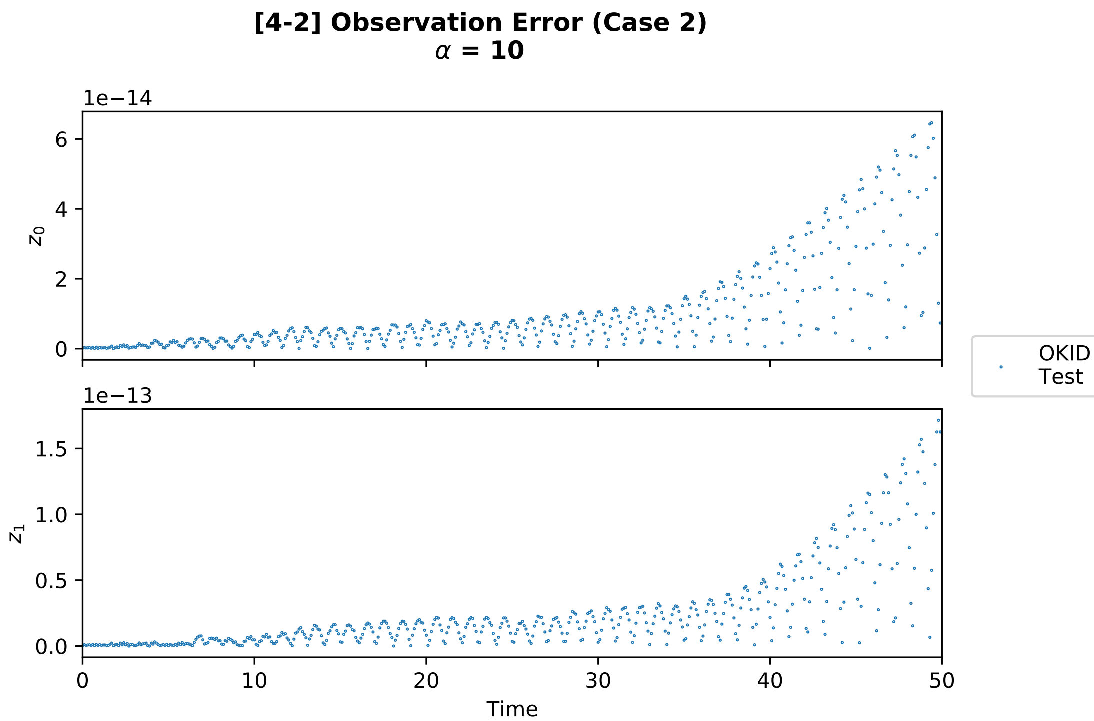

[4-2] Observation Error (Case 1)
α = 20
1e−14
2
1
0
z_0
1e−14
8
6
4
2
0
z_1
0
10
20
30
40
50
Time
OKID Train
OKID Test


[4-2] Observation Responses (Case 2)
α = 5
z0
z1
Time
True
Test
OKID Test
0
10
20
30
40
50

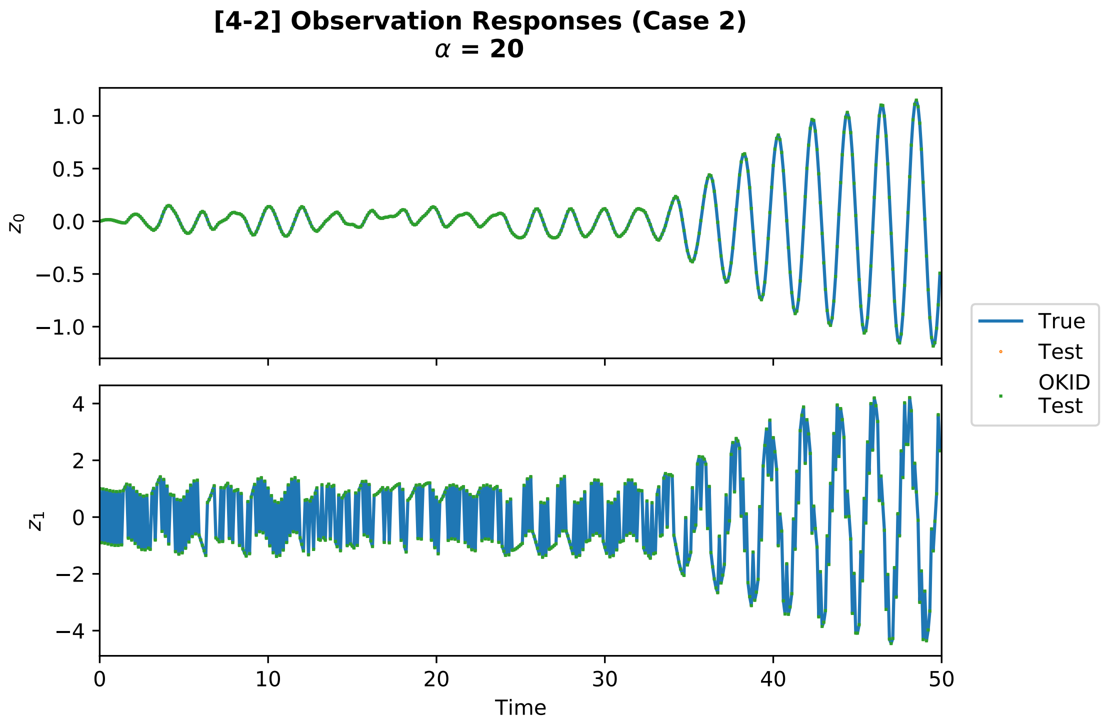

[4-2] Observation Error (Case 2)
α = 5
1e−13
1e−12
z0
z1
Time
OKID Test



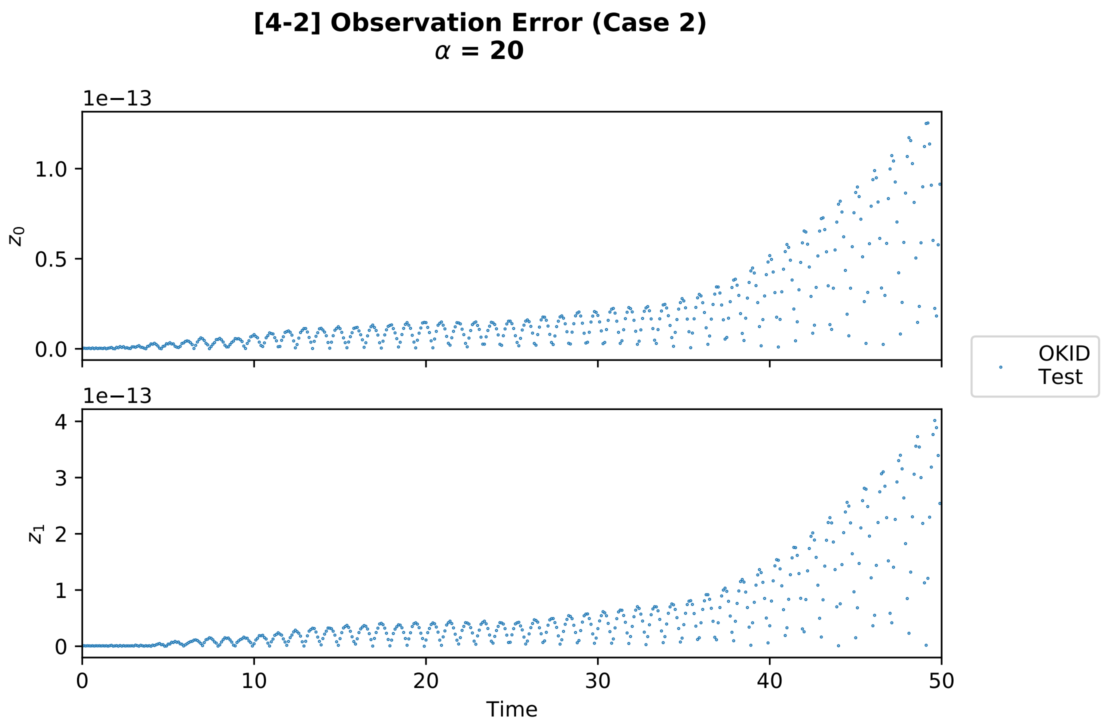
[4-2] Observation Responses (Case 2)
α = 10
True
Test
OKID Test
z_0
z_1
Time



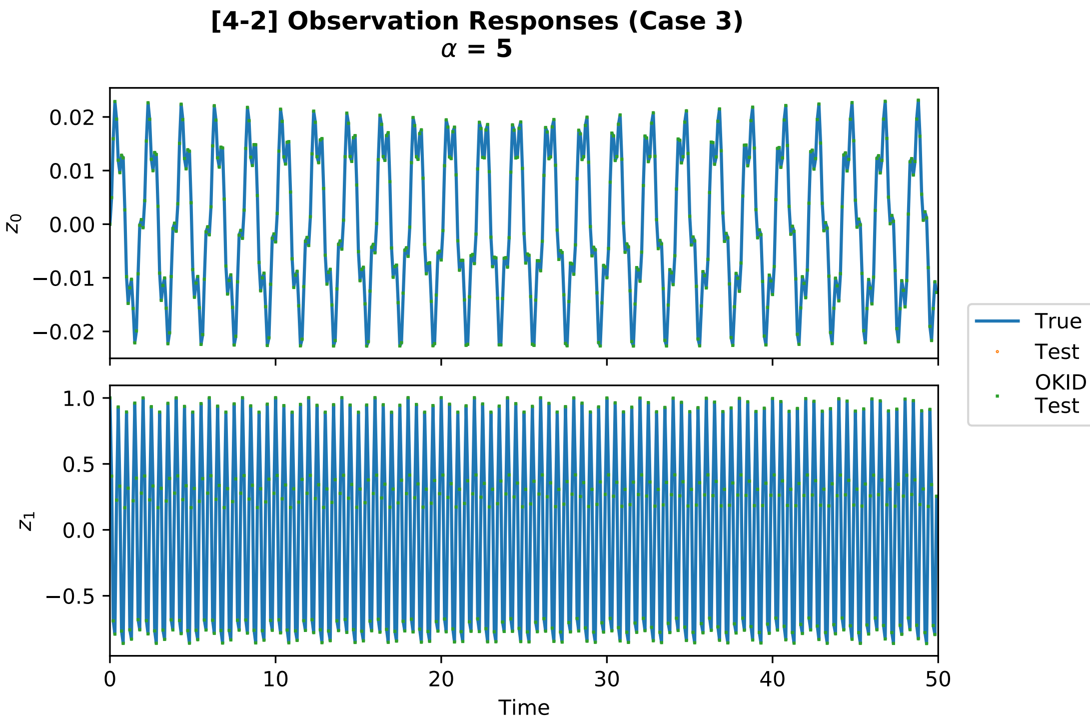
[4-2] Observation Error (Case 2)
α = 10
1e−14
1e−13
OKID Test
z_0
z_1
Time

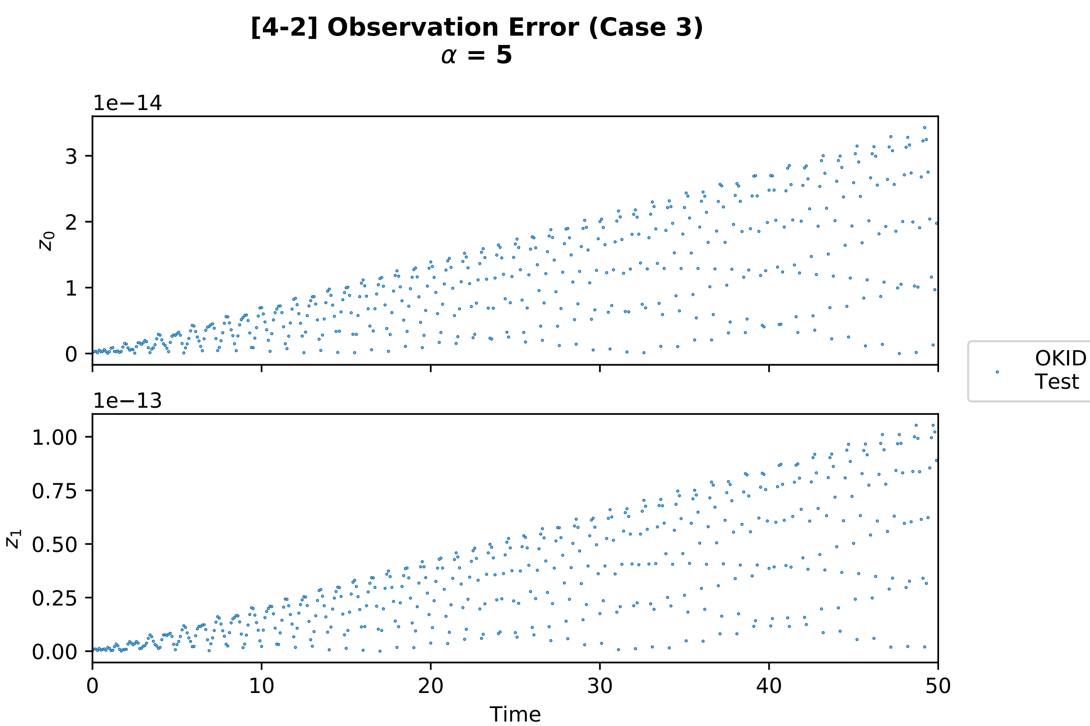


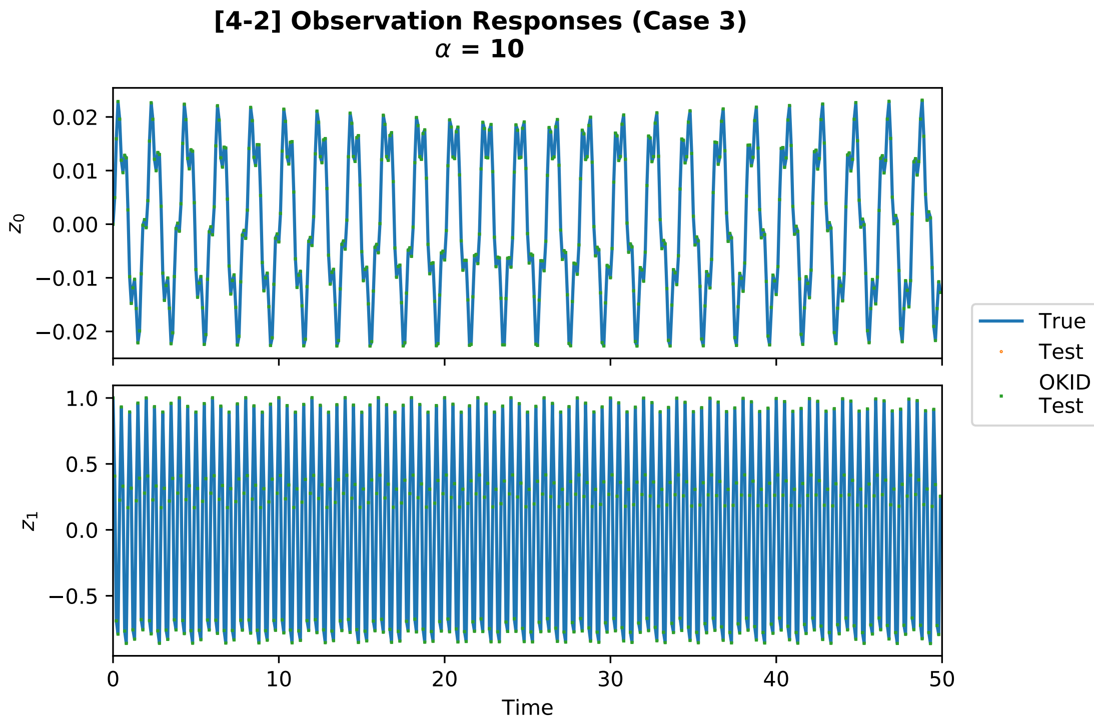
[4-2] Observation Responses (Case 2)
α = 20
z_0
z_1
Time
True
Test
OKID Test

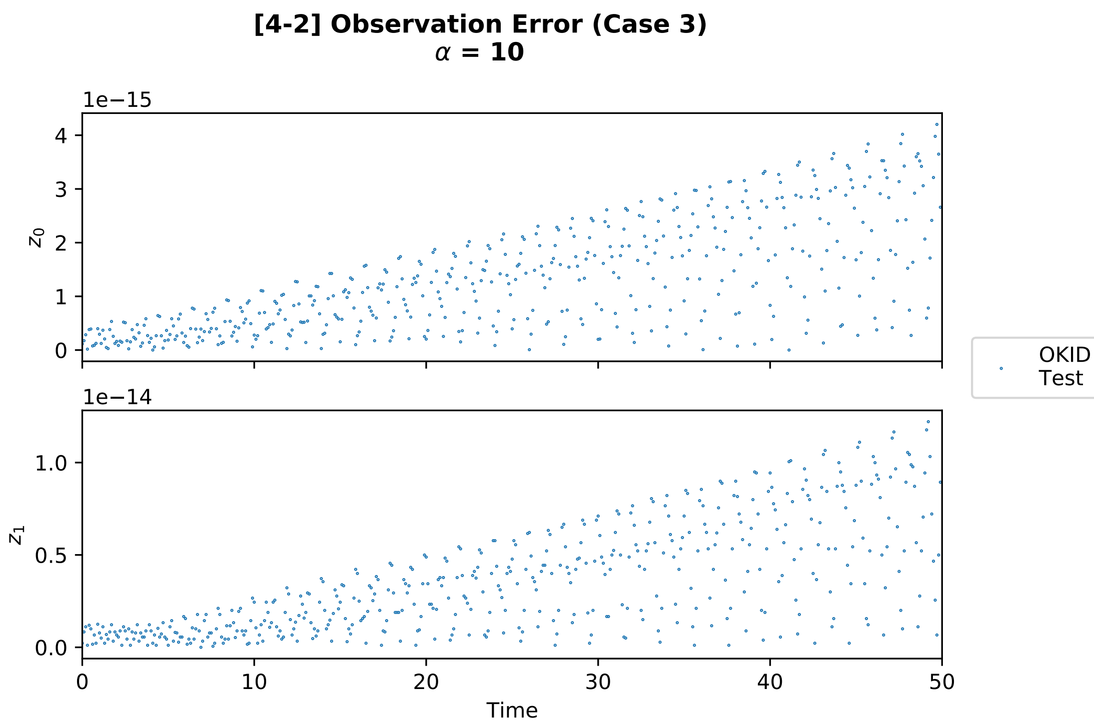


[4-2] Observation Error (Case 2)
α = 20
1e−13
z_0
z_1
Time
OKID Test

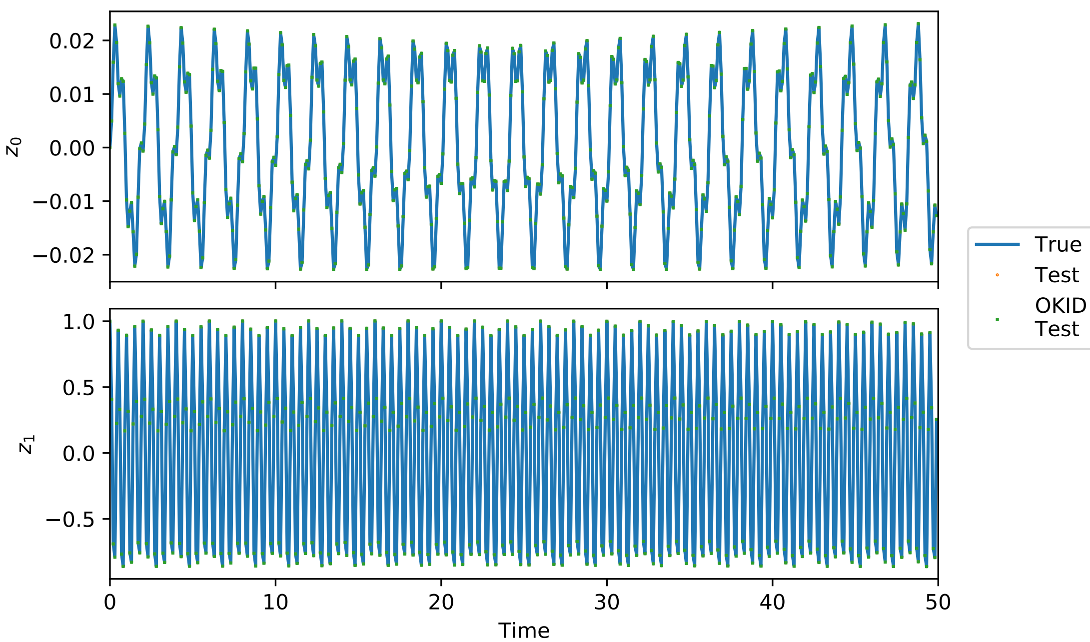

[4-2] Observation Responses (Case 3)
α = 5
z_0
z_1
Time
True
Test
OKID Test


[4-2] Observation Error (Case 3)
α = 5
1e−14
1e−13
z_0
z_1
Time
OKID Test

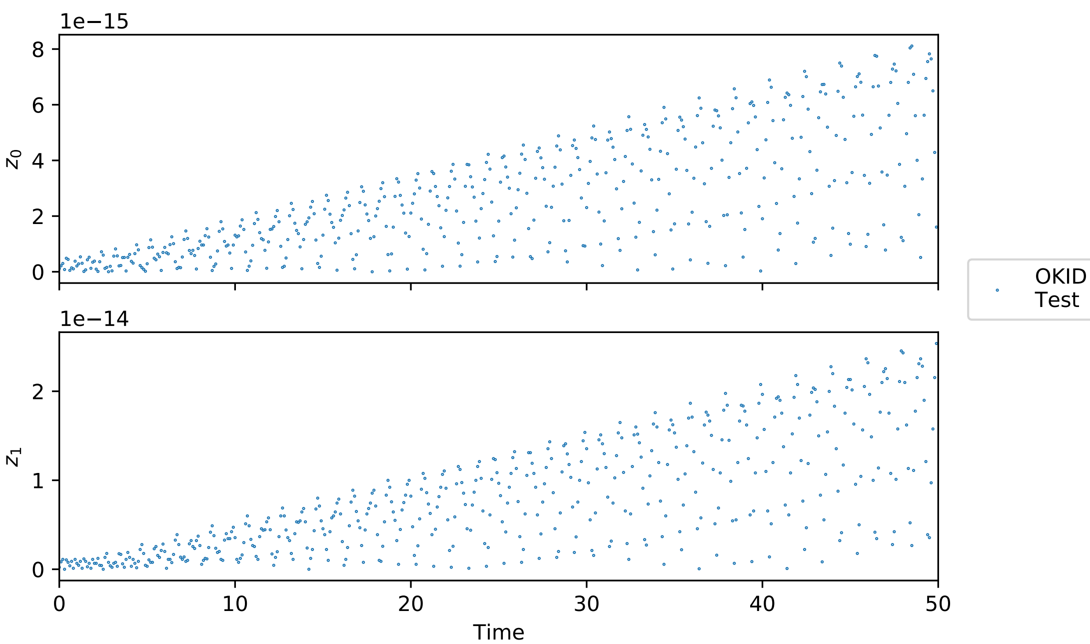

[4-2] Observation Responses (Case 3)
α = 10
0.02
0.01
0.00
−0.01
−0.02
z_0
1.0
0.5
0.0
−0.5
z_1
0
10
20
30
40
50
Time
True
Test
OKID Test


[4-2] Observation Error (Case 3)
α = 10
1e−15
4
3
2
1
0
z_0
1e−14
1.0
0.5
0.0
z_1
0
10
20
30
40
50
Time
OKID Test

[4-2] Observation Responses (Case 3)
α = 20



[4-2] Observation Error (Case 3)
α = 20



The estimation is flawless regardless of the Hankel height chosen. In this case where there is no noise, we conclude that decreasing or increasing the Hankel height did not have any practical negative impact on the high accuracy of the identification of this simple system.