# AERSP597 Midterm

Ani Perumalla

April 1, 2021

## 1 Q. #1

[1]:
```
# Import necessary packages
import warnings # Ignore user warnings
import itertools as it # Readable nested for loops
from pathlib import Path # Filepaths
import typing # Argument / output type checking
import numpy as np # N-dim arrays + math
import scipy.linalg as spla # Complex linear algebra
import matplotlib.pyplot as plt # Plots
import matplotlib.figure as figure # Figure documentation
import scipy.signal as spsg # Signal processing
import sympy # Symbolic math + pretty printing

# Logistics
warnings.simplefilter("ignore", UserWarning)
sympy.init_printing()
figs_dir = (Path.cwd() / "figs")
figs_dir.mkdir(parents = True, exist_ok = True)
prob = 1
```

[2]:
```
def etch(sym: str, mat: np.ndarray):
    display(sympy.Eq(sympy.Symbol(sym),
                     sympy.Matrix(mat.round(5)),
                     evaluate = False))
    pass


def d2c(A: np.ndarray, B: np.ndarray,
        dt: float) \
        -> typing.Tuple[np.ndarray, np.ndarray]:
    """Convert discrete linear state space model to continuous linear state␣
 ↪space model.

    :param np.ndarray A:
    :param np.ndarray B:
    :param float dt: Timestep duration
```

```
    :return: (A_c, B_c) Continuous-time linear state space model
    """
    A_c = spla.logm(A)/dt
    if np.linalg.cond(A - np.eye(*A.shape)) < 1/np.spacing(1):
        B_c = A_c @ spla.inv(A - np.eye(*A.shape)) @ B
    else:
        B_temp = np.zeros(A_c.shape)
        for i in range(200):
            B_temp += (1/((i + 1)*np.math.factorial(i)))*np.linalg.
↪matrix_power(A_c, i)*(dt**(i + 1))
        B_c = B @ spla.inv(B_temp)
    return A_c, B_c


def c2d(A_c: np.ndarray, B_c: np.ndarray,
        dt: float) \
        -> typing.Tuple[np.ndarray, np.ndarray]:
    """Convert continuous linear state space model to discrete linear state␣
↪space model.

    :param np.ndarray A_c:
    :param np.ndarray B_c:
    :param float dt: Timestep duration
    :return: (A, B) Discrete-time linear state space model
    """
    A = spla.expm(A_c*dt)
    if np.linalg.cond(A_c) < 1/np.spacing(1):
        B = (A - np.eye(*A.shape)) @ spla.inv(A_c) @ B_c
    else:
        B_temp = np.zeros(A_c.shape)
        for i in range(200):
            B_temp += (1/((i + 1)*np.math.factorial(i)))*np.linalg.
↪matrix_power(A_c, i)*(dt**(i + 1))
        B = B_temp @ B_c
    return A, B


def sim_ss(A: np.ndarray, B: np.ndarray, C: np.ndarray, D: np.ndarray,
           X_0: np.ndarray, U: np.ndarray,
           nt: int) \
        -> typing.Tuple[np.ndarray, np.ndarray]:
    """Simulate linear state space model via ZOH.

    :param np.ndarray A:
    :param np.ndarray B:
    :param np.ndarray C:
    :param np.ndarray D:
```

```
    :param np.ndarray X_0: Initial state condition
    :param np.ndarray U: Inputs, either impulse or continual
    :param nt: Number of timesteps to simulate
    :return: (X) State vector array over duration; (Z) Observation vector array␣
↪over duration
    """
    assert D.shape == (C @ A @ B).shape
    assert X_0.shape[-2] == A.shape[-1]
    assert U.shape[-2] == B.shape[-1]
    assert A.shape[-2] == B.shape[-2]
    assert C.shape[-2] == D.shape[-2]
    assert A.shape[-1] == C.shape[-1]
    assert B.shape[-1] == D.shape[-1]
    assert (U.shape[-1] == 1) or (U.shape[-1] == nt) or (U.shape[-1] == nt - 1)

    X = np.concatenate([X_0, np.zeros([X_0.shape[-2], nt])], 1)
    Z = np.zeros([C.shape[-2], nt])
    if U.shape[-1] == 1: # Impulse
        X[:, 1] = (A @ X[:, 0]) + (B @ U[:, 0])
        Z[:, 0] = (C @ X[:, 0]) + (D @ U[:, 0])
        for i in range(1, nt):
            X[:, i + 1] = (A @ X[:, i])
            Z[:, i] = (C @ X[:, i])
    else: # Continual
        for i in range(nt):
            X[:, i + 1] = (A @ X[:, i]) + (B @ U[:, i])
            Z[:, i] = (C @ X[:, i]) + (D @ U[:, i])
    return X, Z


def markov_sim(Y: np.ndarray, U: np.ndarray) \
        -> np.ndarray:
    """Obtain observations from Markov parameters and inputs, for zero initial␣
↪conditions

    :param np.ndarray Y: Markov parameter matrix
    :param np.ndarray U: Continual inputs
    :return: (Z) Observation vector array over duration
    :rtype: np.ndarray
    """
    l, m, r = Y.shape
    Y_2_Z = np.zeros([r*l, l])
    Y_2_Z[:r, :] = U
    for i in range(1, l):
        Y_2_Z[r*i:r*(i + 1), :] = np.concatenate([np.zeros([r, i]), U[:, 0:
↪(-i)]], 1)
    Z = np.concatenate(Y, 1) @ Y_2_Z
```

```
    return Z


def ss2markov(A: np.ndarray, B: np.ndarray, C: np.ndarray, D: np.ndarray,
              nt: int) \
        -> np.ndarray:
    """Get Markov parameters from state space model.

    :param np.ndarray A:
    :param np.ndarray B:
    :param np.ndarray C:
    :param np.ndarray D:
    :param nt: Number of Markov parameters to generate (i.e., length of␣
 ↪simulation)
    :return: (Y) 3D array of Markov parameters
    :rtype: np.ndarray
    """
    assert D.shape == (C @ A @ B).shape
    Y = np.zeros([nt, *D.shape])
    Y[0] = D
    for i in range(1, nt):
        Y[i] = C @ (np.linalg.matrix_power(A, i - 1)) @ B
    return Y


def Hankel(Y: np.ndarray, alpha: int, beta: int, i: int = 0) \
        -> np.ndarray:
    """Hankel matrix.

    :param Y: Markov parameter matrix
    :param alpha: Num. of rows of Markov parameters in Hankel matrix
    :param beta: Num. of columns of Markov parameters in Hankel matrix
    :param i: Start node of Hankel matrix
    :return: Block Hankel matrix.
    :rtype: np.ndarray
    """
    assert (len(Y) - 1) >= (i + alpha + beta - 1)
    m, r = Y.shape[-2:]
    H = np.zeros([alpha*m, beta*r])
    for j in range(beta):
        H[:, (j*r):((j + 1)*r)] = Y[(i + 1 + j):(i + alpha + 1 + j)].
 ↪reshape([alpha*m, r])
    return H


def era(Y: np.ndarray, alpha: int, beta: int, n: int) \
        -> typing.Tuple[np.ndarray,
```

```
                        np.ndarray,
                        np.ndarray,
                        np.ndarray,
                        np.ndarray]:
    """Eigensystem Realization Algorithm (ERA).

    :param np.ndarray Y: Markov parameter matrix
    :param int alpha: Num. of rows of Markov parameters in Hankel matrix
    :param int beta: Num. of columns of Markov parameters in Hankel matrix
    :param int n: Order of proposed linear state space system
    :returns: (A, B, C, D) - State space of proposed linear state space system;␣
↪(S) - Singular Values of H(0)
    :rtype: (np.ndarray, np.ndarray, np.ndarray, np.ndarray, np.ndarray, np.
↪ndarray)
    """
    assert (len(Y) - 1) >= (alpha + beta - 1)
    m, r = Y.shape[-2:]
    assert (alpha >= (n/m)) and (beta >= (n/r))
    H_0 = Hankel(Y, alpha, beta, 0)
    print(f"Rank of H(0): {np.linalg.matrix_rank(H_0)}")
    H_1 = Hankel(Y, alpha, beta, 1)
    print(f"Rank of H(1): {np.linalg.matrix_rank(H_1)}")
    U_sim, S, Vh = np.linalg.svd(H_0)
    V = Vh.T
    U_n = U_sim[:, :n]
    V_n = V[:, :n]
    S_n = S[:n]

    E_r = np.concatenate([np.eye(r), np.tile(np.zeros([r, r]), beta - 1)], 1).T
    E_m = np.concatenate([np.eye(m), np.tile(np.zeros([m, m]), alpha - 1)], 1).T
    A = np.diag(S_n**(-1/2)) @ U_n.T @ H_1 @ V_n @ np.diag(S_n**(-1/2))
    B = np.diag(S_n**(1/2)) @ V_n.T @ E_r
    C = E_m.T @ U_n @ np.diag(S_n**(1/2))
    D = Y[0]
    return A, B, C, D, S


def okid(Z: np.ndarray, U: np.ndarray,
         l_0: int,
         alpha: int, beta: int,
         n: int):
    """Observer Kalman Identification Algorithm (OKID).

    :param np.ndarray Z: Observation vector array over duration
    :param np.ndarray U: Continual inputs
    :param int l_0: Order of OKID to execute (i.e., number of Markov parameters␣
↪to generate via OKID)
```

```
    :param int alpha: Num. of rows of Markov parameters in Hankel matrix
    :param int beta: Num. of columns of Markov parameters in Hankel matrix
    :param int n: Number of proposed states to use for ERA
    :return: (Y) Markov parameters
    :rtype: np.ndarray
    """
    r, l_u = U.shape
    m, l  = Z.shape
    assert l == l_u
    V = np.concatenate([U, Z], 0)
    assert (max([alpha + beta, (n/m) + (n/r)]) <= l_0) and (l_0 <= (l - r)/(r +␣
↪m)) # Boundary conditions

    # Form observer
    Y_2_Z = np.zeros([r + (r + m)*l_0, l])
    Y_2_Z[:r, :] = U
    for i in range(1, l_0 + 1):
        Y_2_Z[((i*r) + ((i - 1)*m)):(((i + 1)*r) + (i*m)), :] = np.
↪concatenate([np.zeros([r + m, i]), V[:, 0:(-i)]], 1)
    # Find Observer Markov parameters via least-squares
    Y_obs = Z @ spla.pinv2(Y_2_Z)
    Y_bar_1 = np.array(list(it.chain.from_iterable([Y_obs[:, i:(i + r)]
                                                    for i in range(r, r + (r +␣
↪m)*l_0, r + m)]))).reshape([l_0, m, r])
    Y_bar_2 = -np.array(list(it.chain.from_iterable([Y_obs[:, i:(i + m)]
                                                     for i in range(2*r, r + (r␣
↪+ m)*l_0, r + m)]))).reshape([l_0, m, m])

    # Obtain Markov parameters from Observer Markov parameters
    Y = np.zeros([l_0 + 1, m, r])
    Y[0] = Y_obs[:, :r]
    for k in range(1, l_0 + 1):
        Y[k] = Y_bar_1[k - 1] - \
               np.array([Y_bar_2[i] @ Y[k - (i + 1)]
                         for i in range(k)]).sum(axis = 0)
    # Obtain Observer Gain Markov parameters from Observer Markov parameters
    Y_og = np.zeros([l_0, m, m])
    Y_og[0] = Y_bar_2[0]
    for k in range(1, l_0):
        Y_og[k] = Y_bar_2[k] - \
                  np.array([Y_bar_2[i] @ Y_og[k - (i + 1)]
                            for i in range(k - 1)]).sum(axis = 0)
    return Y, Y_og
```

[3]:
```
# Set seed for consistent results
rng = np.random.default_rng(seed = 100)
```

```
# Simulation dimensions
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)))
# Note that damping is always positive even when it is displayed as negative.

# 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

# 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([n, nt_train + 1]) # Train state vector
```

```
X_test = np.zeros([cases, n, nt_test + 1]) # Test state vectors
Z_train = np.zeros([m, nt_train]) # Train observation vector
Z_test = np.zeros([cases, m, nt_test]) # Test observation vectors
V_train = np.zeros([r + m, nt_train]) # Train observation input vectors
V_test = np.zeros([cases, r + m, nt_test]) # Test observation input vectors
```

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

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

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

[4]:
```
# 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

print(f"Min. OKID Order: {max([alpha + beta, (n_era/m) + (n_era/r)]):n}")
print(f"Max. OKID Order: {(nt_train - r)/(r + m):n}")
print(f"Proposed OKID Order: {order:n}")
```

```
Min. OKID Order: 35
Max. OKID Order: 66.6667
Proposed OKID Order: 50
```

We propose to use an OKID order $l_0$ of 50, since this value falls inside the range of permitted OKID orders.

The Hankel matrix $H(0)$ must be at least rank $n$. In addition, the sum of the numbers of block rows ($\alpha$) and columns ($\beta$) should be at most the number of estimated Markov parameters (i.e., the OKID order $l_0 = 50$). Moreover, to meet the full rank condition for $H(0)$, $\alpha \geq \frac{n}{m} = 1$ and $\beta \geq \frac{n}{r} = 2$. To meet these criteria, we choose $\alpha = 15$ and $\beta = 20$.

As shown by the SVD plot below, the order of the system appears to be 2, since there is a sharp dropoff in the value of the singular values for $n > 2$. Therefore, we will choose the number of states for the OKID/ERA-proposed system to be 2.

[5]:
```
# OKID System Markov parameters
Y_okid = np.zeros([order + 1, m, r])
# OKID Observer Markov Gain parameters
Y_og_okid = np.zeros([order, m, m])
# OKID state vector, drawn from state space model derived from OKID/ERA
X_okid_train = np.zeros([n_era, nt_train + 1])
X_okid_test = np.zeros([cases, n_era, nt_test + 1])
X_okid_train_obs = np.zeros([n_era, nt_train + 1])
```

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

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

[6]:
```
# Simulation
for i in range(cases):
    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, Z_train = 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, U_train,
                                 l_0 = order, alpha = alpha, beta = beta, n =␣
↪n_era)
        # Identify state space model using System Markov parameters for ERA
        A_okid, B_okid, C_okid, D_okid, S_okid = \
            era(Y_okid, alpha = alpha, beta = beta, n = n_era)
        # Construct observability matrix
        O_p_okid = np.array([C_okid @ np.linalg.matrix_power(A_okid, i)
                             for i in range(order)])
        # Find observer gain matrix
        G_okid = 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 = A_okid + G_okid @ C_okid
        B_okid_obs = np.concatenate([B_okid + G_okid @ D_okid, -G_okid], 1)
```

```
        C_okid_obs = C_okid
        D_okid_obs = np.concatenate([D_okid, np.zeros([m, m])], 1)
        V_train = np.concatenate([U_train, Z_train], 0)
        # Simulate OKID realization with "raw" state and OKID realization with␣
↪estimated state
        X_okid_train, Z_okid_train = \
            sim_ss(A_okid, B_okid, C_okid, D_okid,
                   X_0 = X_0_okid, U = U_train, nt = nt_train)
        X_okid_train_obs, Z_okid_train_obs = \
            sim_ss(A_okid_obs, B_okid_obs, C_okid_obs, D_okid_obs,
                   X_0 = X_0_okid, U = V_train, nt = nt_train)
        # Display outputs
        etch(f"A_{{OKID}}", A_okid)
        etch(f"B_{{OKID}}", B_okid)
        etch(f"C_{{OKID}}", C_okid)
        etch(f"D_{{OKID}}", D_okid)
        etch(f"G_{{OKID}}", G_okid)
        # Calculate and display eigenvalues
        eig_A_okid = spla.eig(d2c(A_okid, B_okid, dt)[0])[0] # Eigenvalues of␣
↪identified system
        etch(f"\hat{{\lambda}}", eig_A_okid)
        etch(f"\hat{{\omega}}_{{n}}", np.abs(eig_A_okid))
        etch(f"\hat{{\zeta}}", -np.cos(np.angle(eig_A_okid)))
    X_test[i], Z_test[i] = X_sim[i], Z_sim[i]
    X_okid_test[i], Z_okid_test[i] = \
        sim_ss(A_okid, B_okid, C_okid, D_okid,
               X_0 = X_0_okid, U = U_test[i], nt = nt_test)
    V_test[i] = np.concatenate([U_test[i], Z_test[i]], 0)
    X_okid_test_obs[i], Z_okid_test_obs[i] = \
        sim_ss(A_okid_obs, B_okid_obs, C_okid_obs, D_okid_obs,
               X_0 = X_0_okid, U = V_test[i], nt = nt_test)
```

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

$$A_{OKID} = \begin{bmatrix} 0.95344 & -0.33621 \\ 0.2807 & 0.94985 \end{bmatrix}$$

$$B_{OKID} = \begin{bmatrix} -0.22189 \\ -0.20365 \end{bmatrix}$$

$$C_{OKID} = \begin{bmatrix} 0.06029 & -0.09004 \\ -0.2558 & -0.20438 \end{bmatrix}$$

$$D_{OKID} = \begin{bmatrix} 0.0 \\ 1.0 \end{bmatrix}$$

$$G_{OKID} = \begin{bmatrix} -0.05416 & 0.15911 \\ 0.03786 & 0.1723 \end{bmatrix}$$

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

$$\hat{\omega}_n = \begin{bmatrix} 3.1225 \\ 3.1225 \end{bmatrix}$$

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

The eigenvalues of the system are accurately identified via OKID. The natural frequencies and damping ratios are essentially exactly identified.

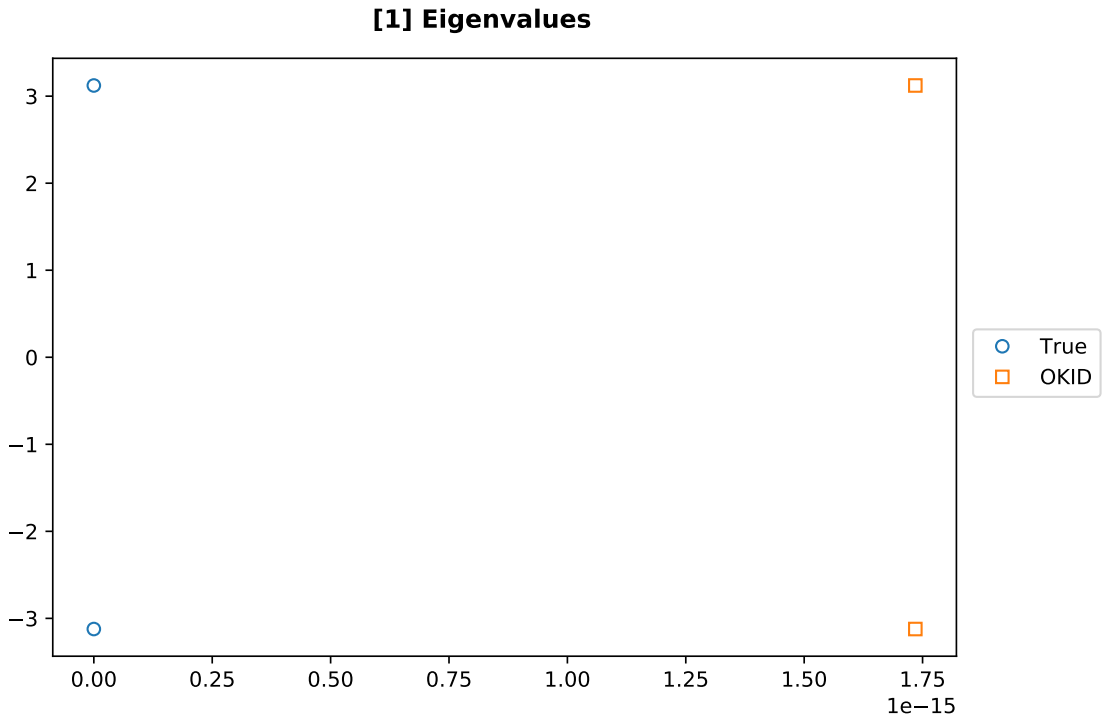
[7]:
```
RMS_train = np.sqrt(np.mean((Z_okid_train - Z_train)**2, axis = 1))
print(f"RMS Error of sim. for system found via OKID for train data:␣
 ↪{RMS_train}")
RMS_test = np.zeros([cases, m])
for i in range(cases):
    RMS_test[i] = np.sqrt(np.mean((Z_okid_test[i] - Z_test[i])**2, axis = 1))
    print(f"RMS Error of sim. for system found via OKID for test data, case {i}:
 ↪ {RMS_test[i]}")
```

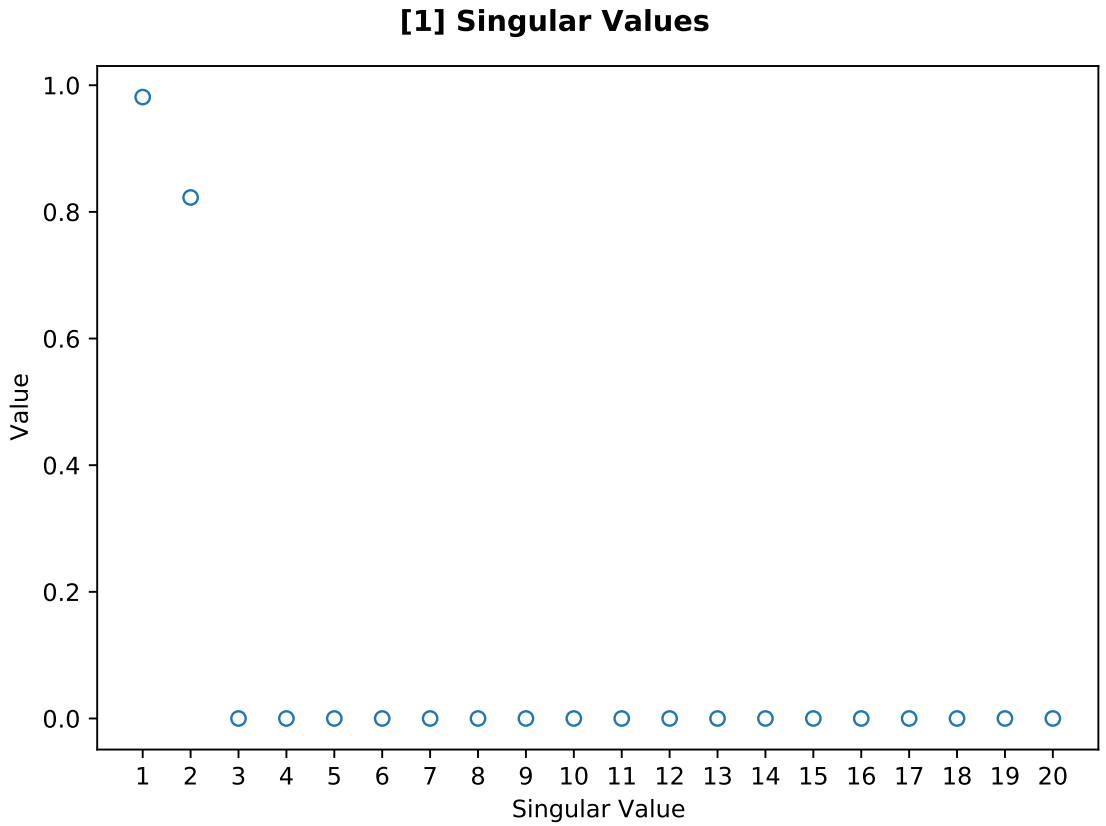
```
RMS Error of sim. for system found via OKID for train data: [5.64018678e-16
1.81879758e-15]
RMS Error of sim. for system found via OKID for test data, case 0:
[1.99758691e-15 6.19485013e-15]
RMS Error of sim. for system found via OKID for test data, case 1:
[6.28794915e-15 1.90445491e-14]
RMS Error of sim. for system found via OKID for test data, case 2:
[6.30791837e-16 2.17401648e-15]
```

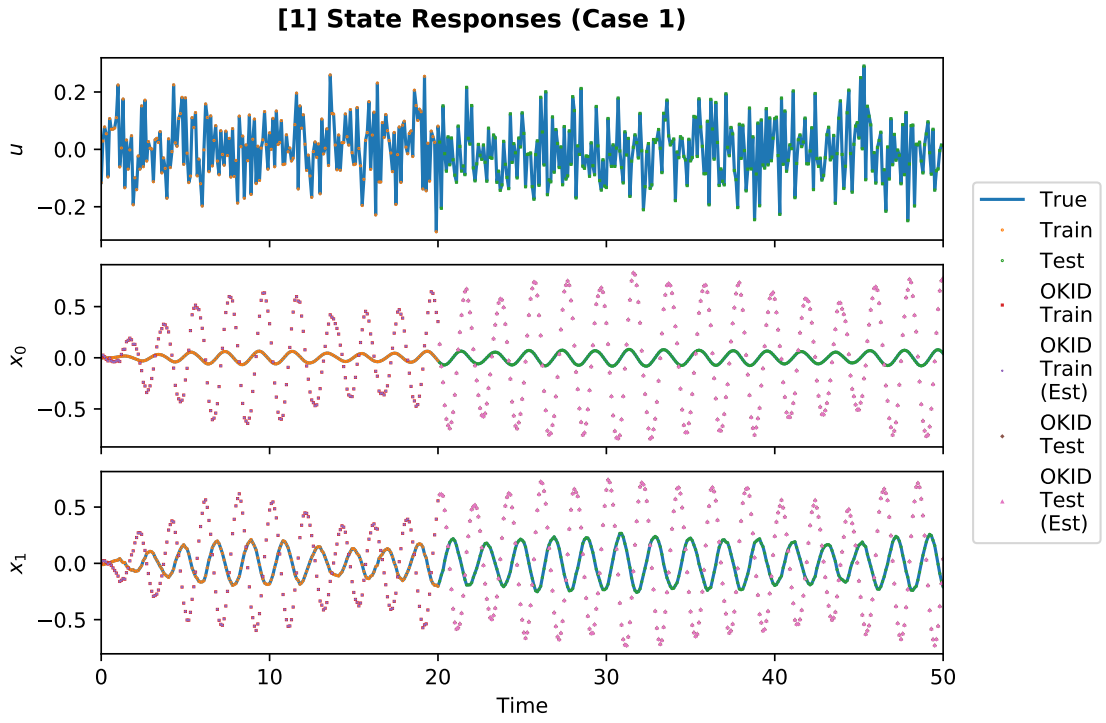
[8]:
```
# Eigenvalue plots
fig, ax = plt.subplots(constrained_layout = True) # type:figure.Figure
fig.suptitle(f"[{prob}] Eigenvalues", fontweight = "bold")

ax.plot(np.real(eig_A), np.imag(eig_A),
         "o", mfc = "None")
ax.plot(np.real(eig_A_okid), np.imag(eig_A_okid),
         "s", mfc = "None")

fig.legend(labels = ("True", "OKID"),
           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")

ax.plot(np.linspace(1, len(S_okid), len(S_okid)), S_okid,
```

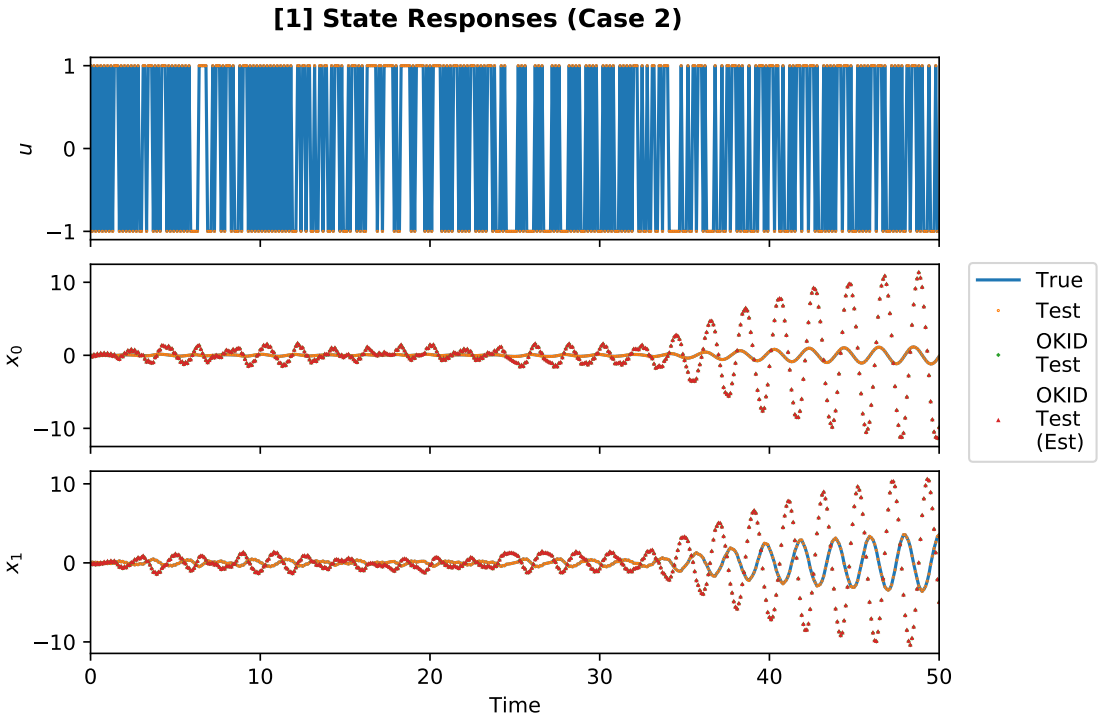
```
        "o", mfc = "None")
plt.setp(ax, xlabel = f"Singular Value", ylabel = f"Value",
        xticks = np.arange(1, len(S_okid) + 1))

fig.savefig(figs_dir / f"midterm_{prob}_singval.pdf",
            bbox_inches = "tight")

# Response plots
ms = 0.5 # Marker size
for i in range(cases):
    fig, axs = plt.subplots(1 + n, 1,
                            sharex = "col",
                            constrained_layout = True) # type:figure.Figure
    fig.suptitle(f"[{prob}] State Responses (Case {i + 1})",
                 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[j],
                            "o", ms = ms, mfc = "None")
            axs[j + 1].plot(t_test[train_cutoff:], X_test[i, j, train_cutoff:],
                            "o", ms = ms, mfc = "None")
            axs[j + 1].plot(t_train, X_okid_train[j, :-1],
                            "s", ms = ms, mfc = "None")
            axs[j + 1].plot(t_train, X_okid_train_obs[j, :-1],
                            "*", ms = ms, mfc = "None")
            axs[j + 1].plot(t_test[train_cutoff:], X_okid_test[i, j,␣
↪train_cutoff:],
                            "D", ms = ms, mfc = "None")
            axs[j + 1].plot(t_test[train_cutoff:], X_okid_test_obs[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)
```

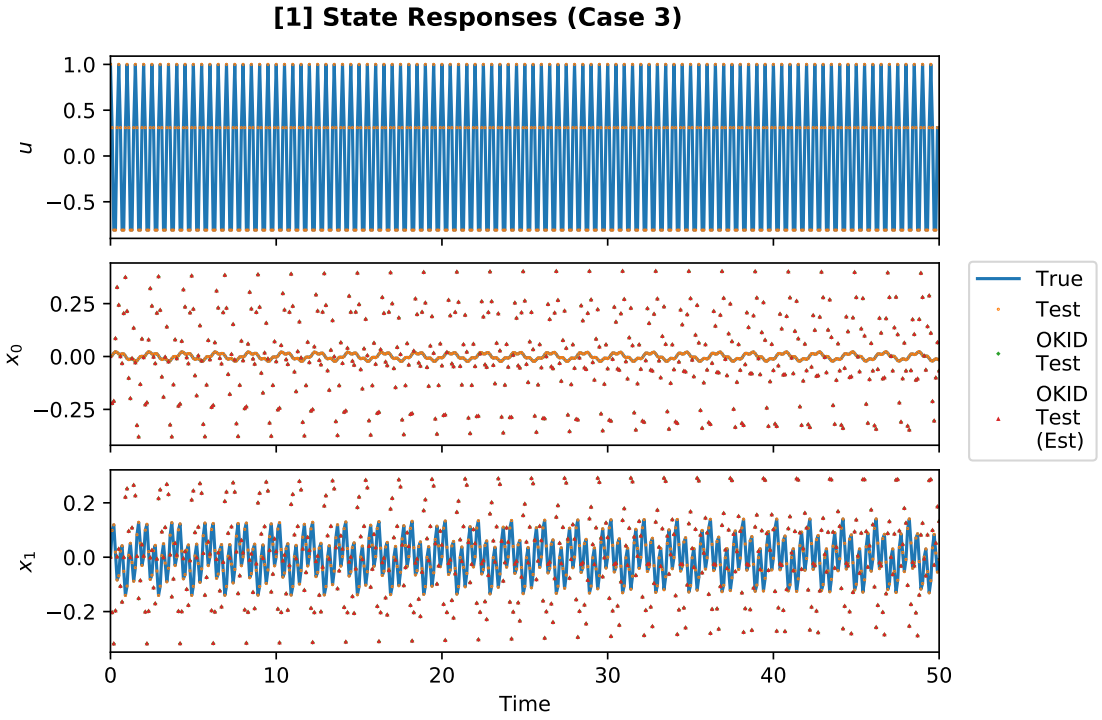
```
    else:
        axs[0].plot(t_sim[:-1], U_sim[i, 0])
        axs[0].plot(t_test[:-1], U_test[i, 0],
                    "o", ms = ms, mfc = "None")
        plt.setp(axs[0], ylabel = f"$u$", xlim = [0, t_max])

        for j in range(n):
            axs[j + 1].plot(t_sim, X_sim[i, j])
            axs[j + 1].plot(t_test, X_test[i, j],
                            "o", ms = ms, mfc = "None")
            axs[j + 1].plot(t_test, X_okid_test[i, j],
                            "D", ms = ms, mfc = "None")
            axs[j + 1].plot(t_test, X_okid_test_obs[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}.pdf",
                bbox_inches = "tight")
```




[1] Singular Values
Value
Singular Value
1.0
0.8
0.6
0.4
0.2
0.0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20


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

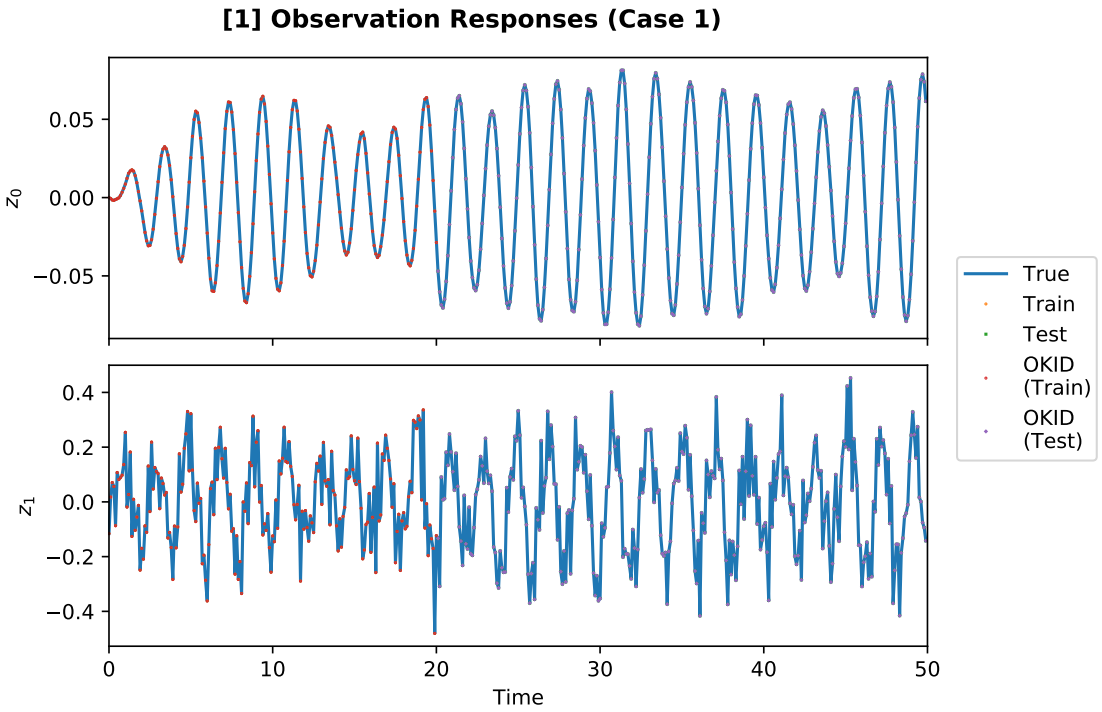

[1] State Responses (Case 2)
u
$x_0$
$x_1$
Time
True
Test
OKID Test
OKID Test (Est)

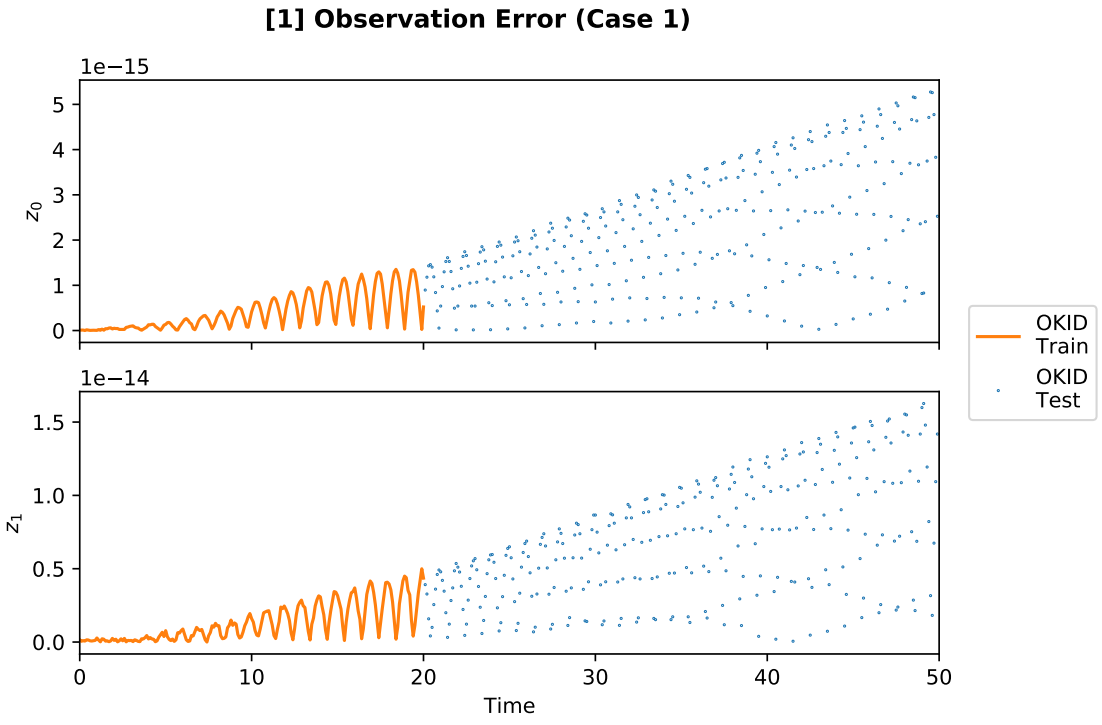

[1] State Responses (Case 3)
u
$x_0$
$x_1$
Time
True
Test
OKID Test
OKID Test (Est)

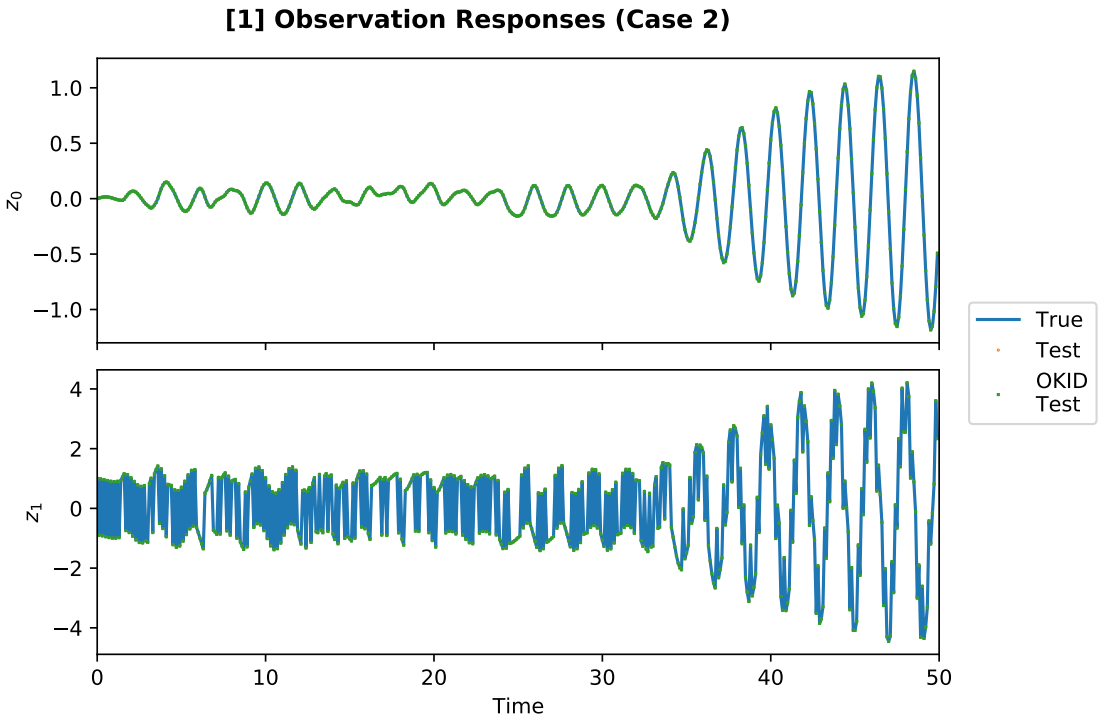
The improvement of the observer to the state estimate is negligible, as the "raw" state is already a perfect realization of the system state that reproduces the test outputs at each sample time.

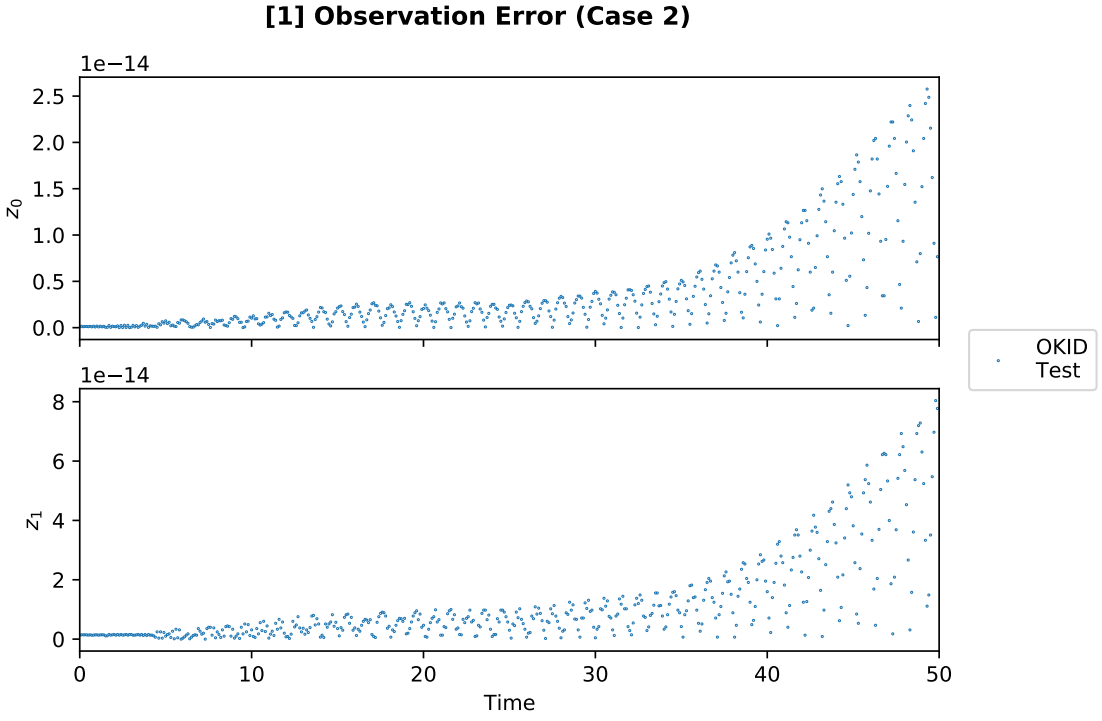
[9]:
```
# Observation plots
for i in range(cases):
    # Raw observations
    fig, axs = plt.subplots(m, 1,
                            sharex = "col",
                            constrained_layout = True) # type:figure.Figure
    fig.suptitle(f"[{prob}] Observation Responses (Case {i + 1})",
                 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[j],
                        "o", ms = ms, mfc = "None")
            axs[j].plot(t_test[train_cutoff:-1], Z_test[i, j, train_cutoff:],
                        "s", ms = ms, mfc = "None")
            axs[j].plot(t_train, Z_okid_train[j],
                        "o", ms = ms, mfc = "None")
            axs[j].plot(t_test[train_cutoff:-1], Z_okid_test[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")
        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[i, j],
                        "o", ms = ms, mfc = "None")
            axs[j].plot(t_test[:-1], Z_okid_test[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")
        fig.legend(labels = ["True", "Test", "OKID\nTest"],
                   bbox_to_anchor = (1, 0.5), loc = 6)
    fig.savefig(figs_dir / f"midterm_{prob}_obs_case{i + 1}.pdf",
                bbox_inches = "tight")
```

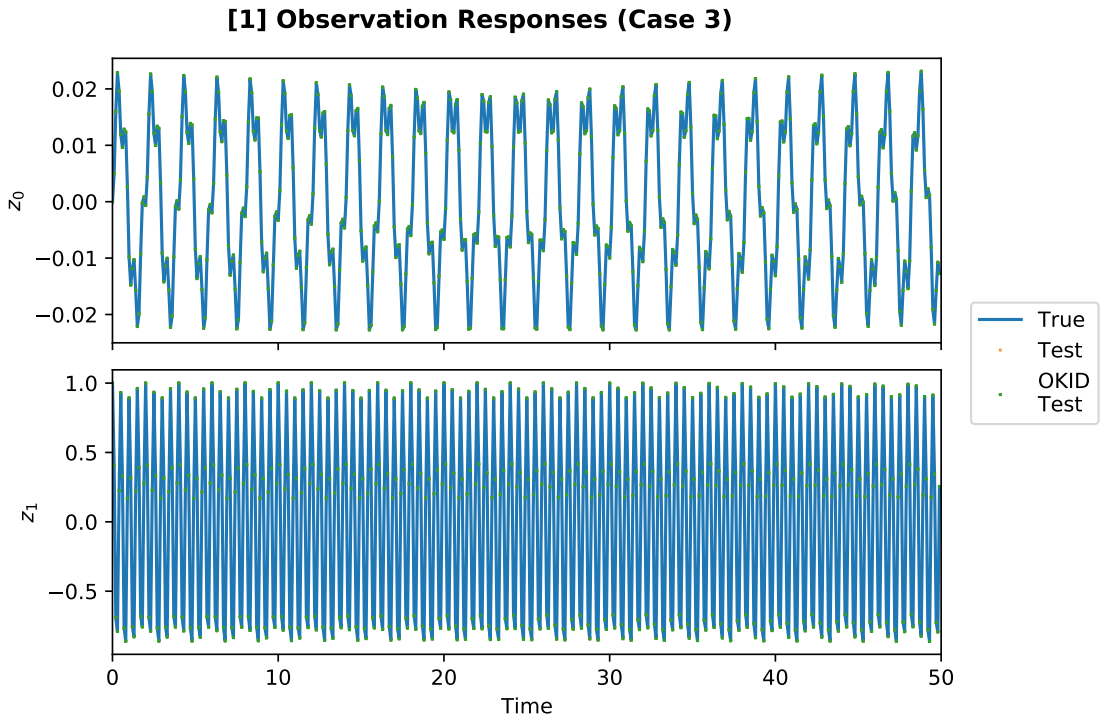
```
    # Observation error
    fig, axs = plt.subplots(m, 1,
                            sharex = "col",
                            constrained_layout = True) # type:figure.Figure
    fig.suptitle(f"[{prob}] Observation Error (Case {i + 1})",
                 fontweight = "bold")
    if i == 0:
        for j in range(m):
            axs[j].plot(t_train, np.abs(Z_okid_train[j] - Z_train[j]),
                        c = "C1")
            axs[j].plot(t_test[train_cutoff:-1], np.abs(Z_okid_test[i, j,␣
↪train_cutoff:] - Z_test[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")
        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[i, j] - Z_test[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")
        fig.legend(labels = ["OKID\nTest"],
                   bbox_to_anchor = (1, 0.5), loc = 6)
    fig.savefig(figs_dir / f"midterm_{prob}_obs-error_case{i + 1}.pdf",
                bbox_inches = "tight")
```

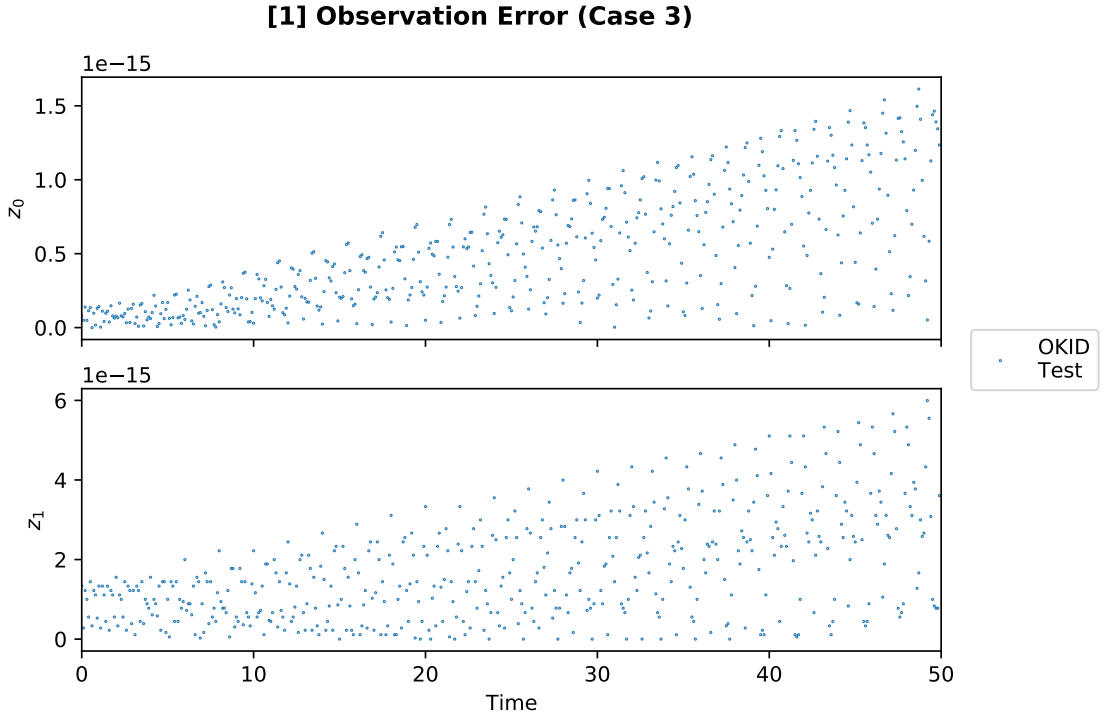

[1] Observation Responses (Case 1)
0.05
0.00
−0.05
$z_0$
0.4
0.2
0.0
−0.2
−0.4
$z_1$
0
10
20
30
40
50
Time
True
Train
Test
OKID (Train)
OKID (Test)


[1] Observation Error (Case 1)
1e−15
5
4
3
2
1
0
$z_0$
1e−14
1.5
1.0
0.5
0.0
$z_1$
0
10
20
30
40
50
Time
OKID Train
OKID Test


[1] Observation Responses (Case 2)
$z_0$
$z_1$
Time
True
Test
OKID
Test


[1] Observation Error (Case 2)
1e−14
1e−14
$z_0$
$z_1$
Time
OKID
Test


[1] Observation Responses (Case 3)
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


[1] Observation Error (Case 3)
1e−15
1.5
1.0
0.5
0.0
$z_0$
1e−15
6
4
2
0
$z_1$
0
10
20
30
40
50
Time
OKID
Test

The observation sequence is reproduced essentially flawlessly for each case.

# 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.

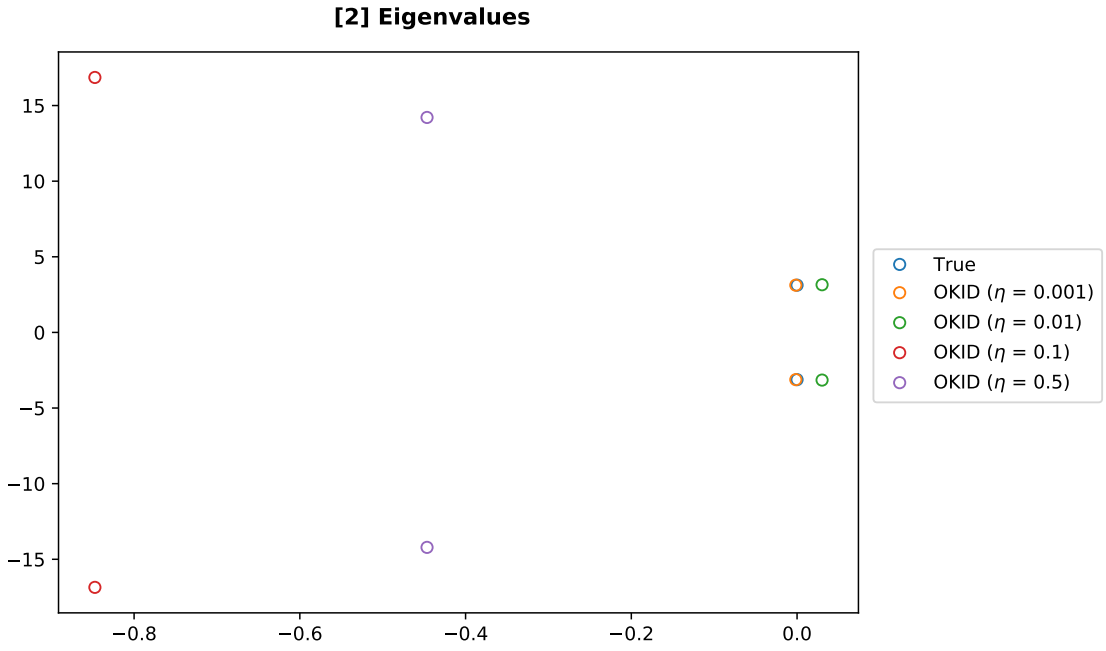
[6]:
```
RMS_train = np.zeros([len(noises), m])
RMS_test = np.zeros([len(noises), cases, m])
for j in range(len(noises)):
    RMS_train[j] = np.sqrt(np.mean((Z_okid_train[j] - Z_train[j])**2, axis = 1))
    print(f"RMS Error of sim. for system found via OKID for train data, noise␣
↪std.dev. = {noises[j]}: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

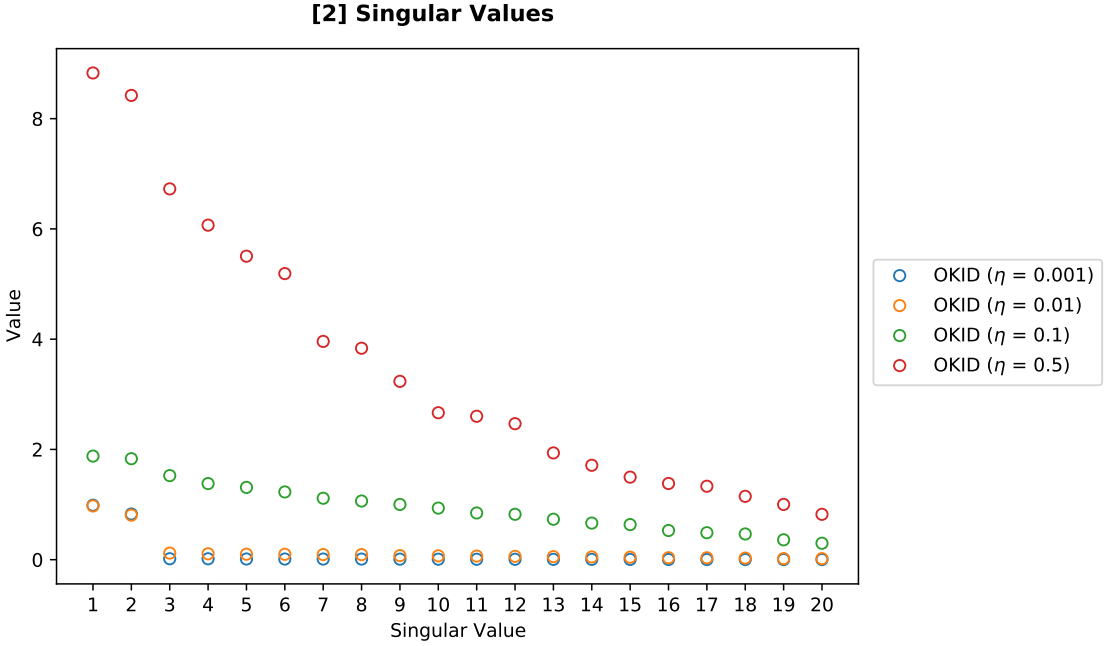
```
RMS Error of sim. for system found via OKID for train data, noise std.dev. =
0.001: [0.00109362 0.00155008]
RMS Error of sim. for system found via OKID for test data, noise std.dev. =
0.001, case 0: [0.00182514 0.00464139]
RMS Error of sim. for system found via OKID for test data, noise std.dev. =
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 $\eta$ 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]:
```
# 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")
```

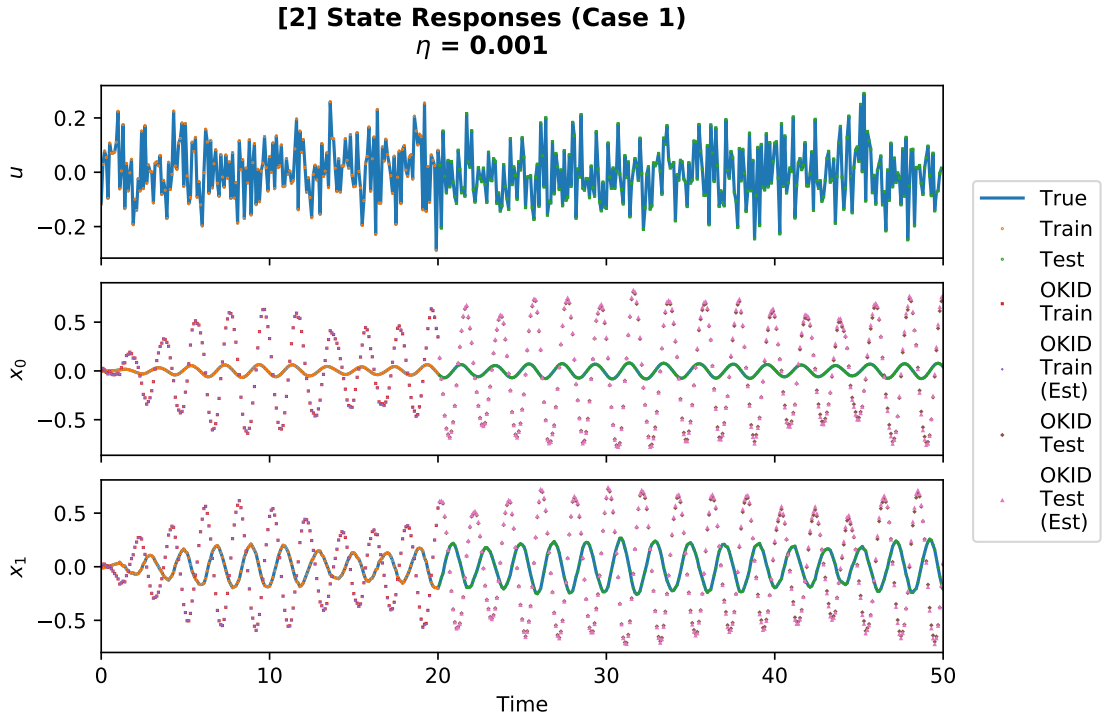


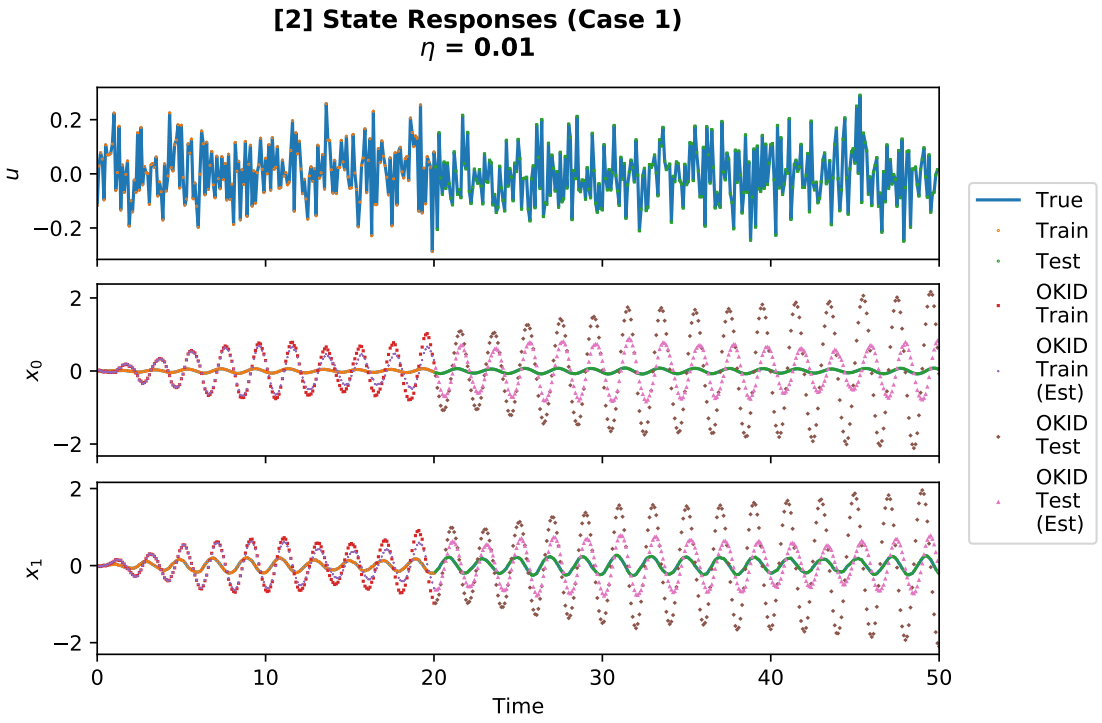



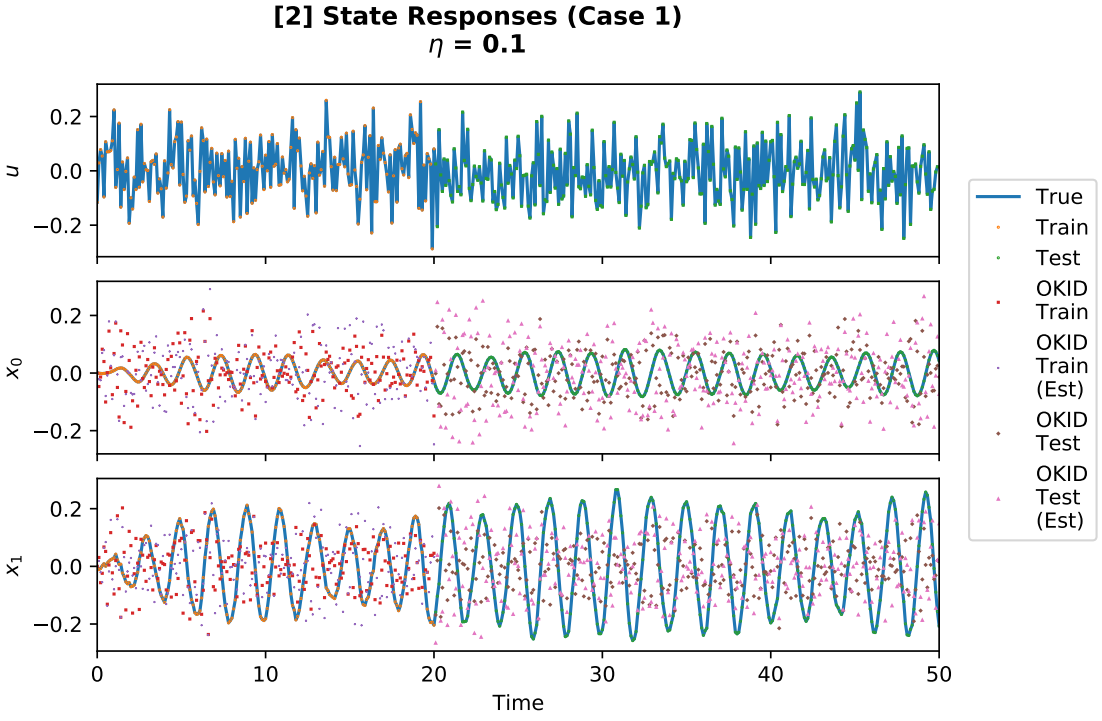

As shown by the singular value plot, the singular values remain somewhat disjoint for $\eta < 0.1$, but as $\eta$ increases, the singular values converge and the dropoff between the 2nd and 3rd singular values decreases sharply, making it harder to identify that the system has a true order $n$ of 2.

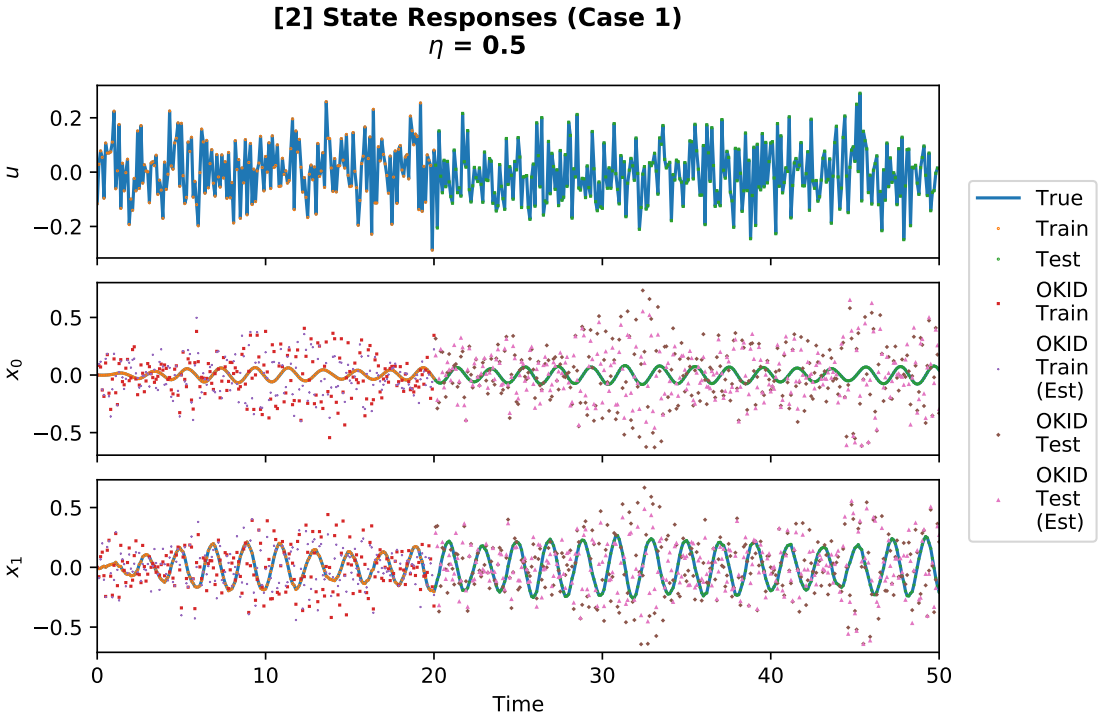
[8]:
```
# Response plots
ms = 0.5 # Marker size
for i, k in it.product(range(cases), range(len(noises))):
    fig, axs = plt.subplots(1 + n, 1,
                            sharex = "col",
                            constrained_layout = True) # type:figure.Figure
    fig.suptitle(f"[{prob}] State Responses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

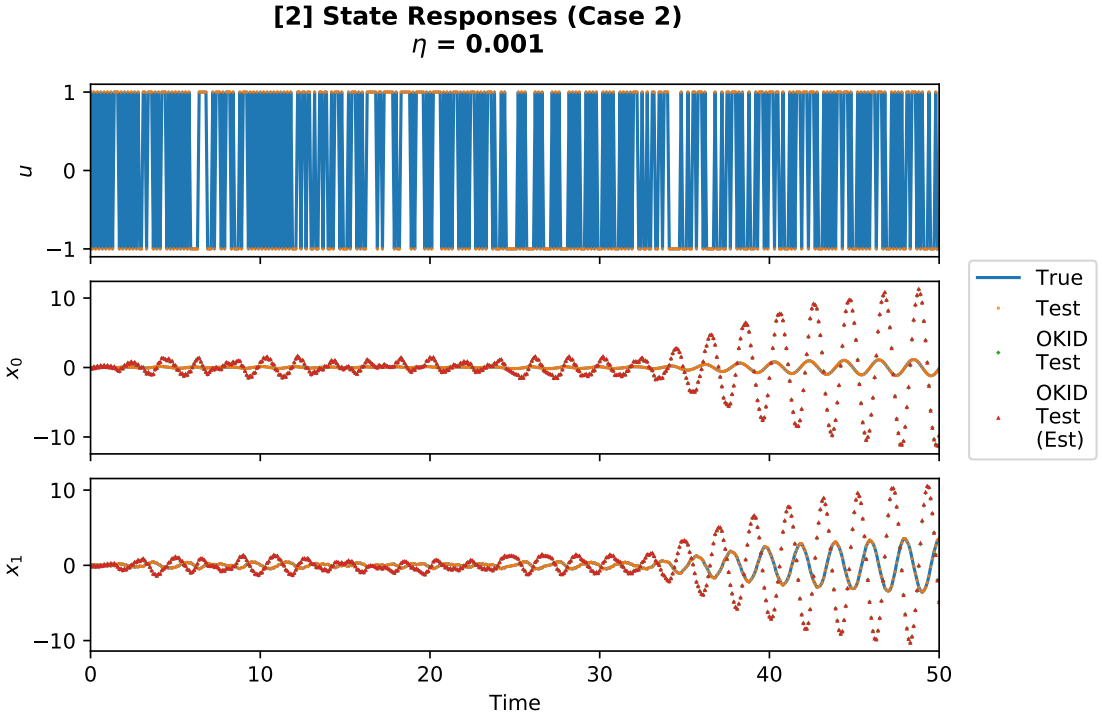
```
    for j in range(n):
        axs[j + 1].plot(t_sim, X_sim[i, j])
        axs[j + 1].plot(t_test, X_test[k, i, j],
                        "o", ms = ms, mfc = "None")
        axs[j + 1].plot(t_test, X_okid_test[k, i, j],
                        "D", ms = ms, mfc = "None")
        axs[j + 1].plot(t_test, 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")
```

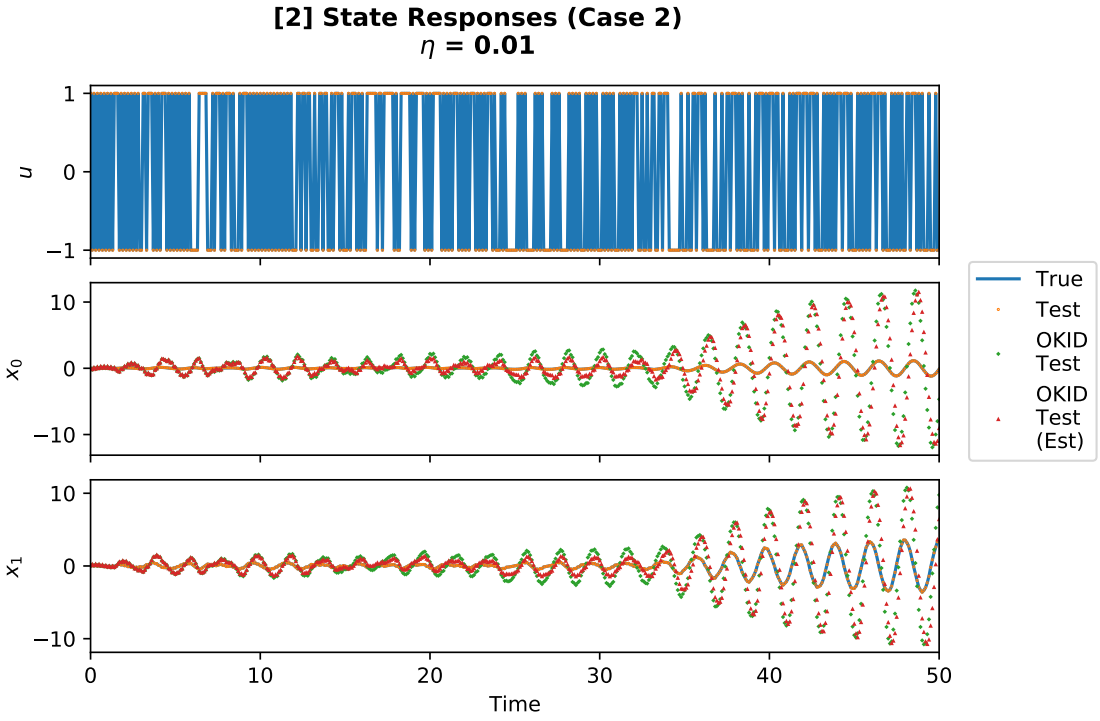


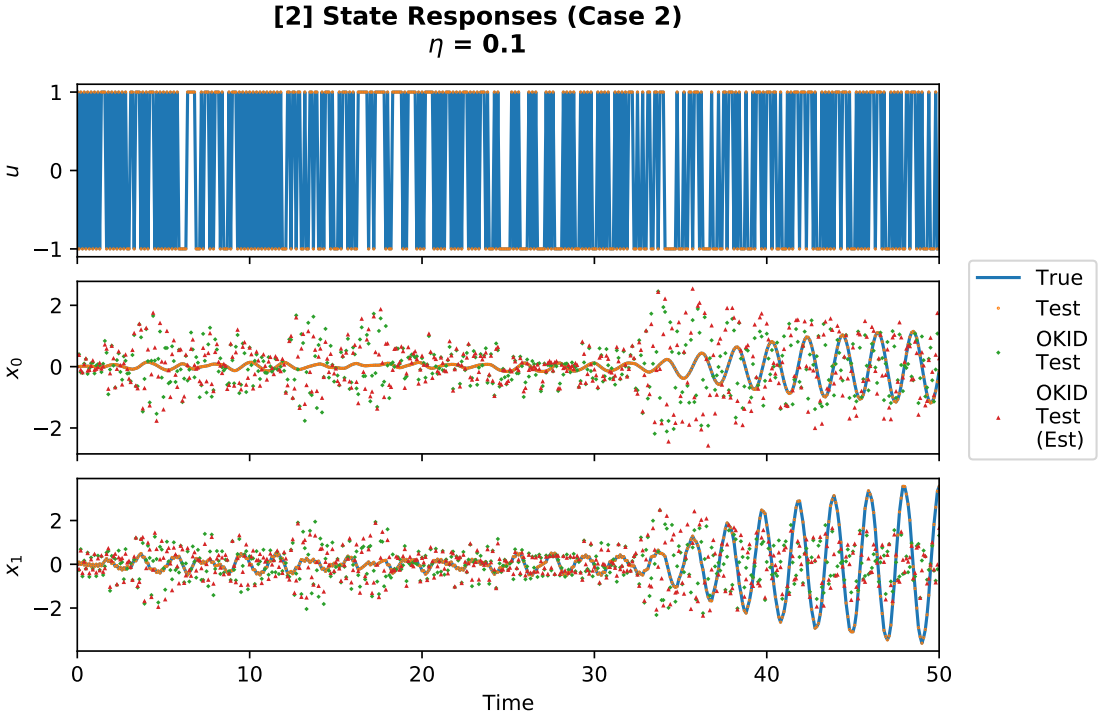

[2] State Responses (Case 1)
$\eta$ = 0.01
$u$
$x_0$
$x_1$
Time
True
Train
Test
OKID Train
OKID Train (Est)
OKID Test
OKID Test (Est)

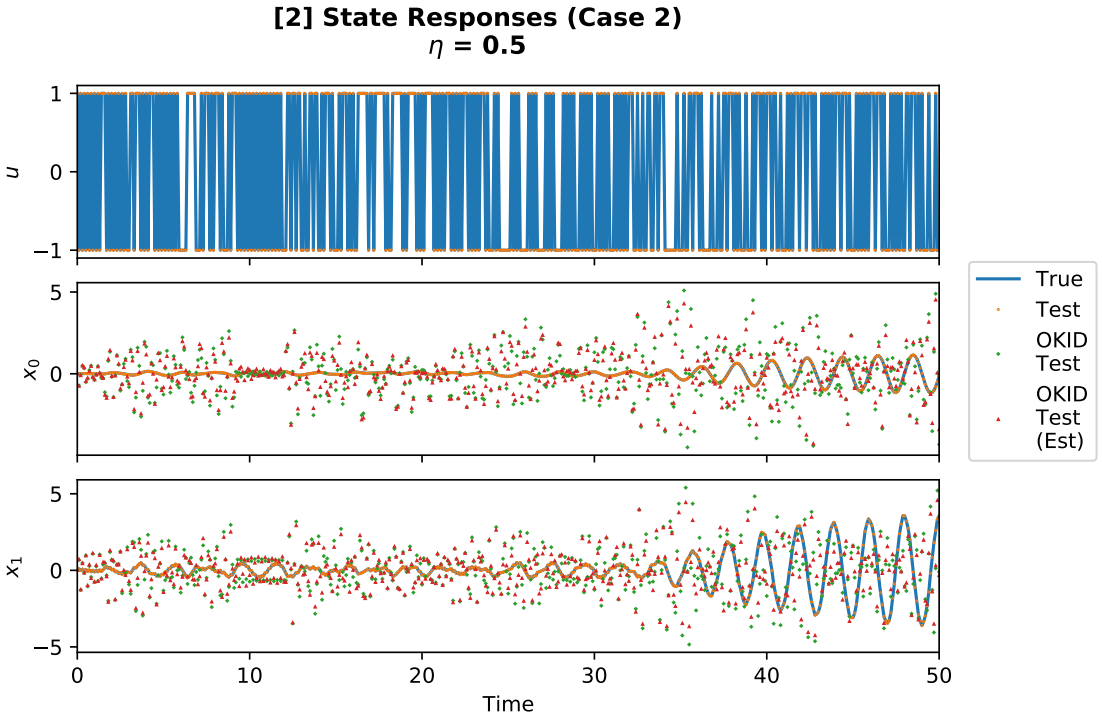

[2] State Responses (Case 1)
$\eta$ = 0.1
$u$
$x_0$
$x_1$
Time
True
Train
Test
OKID Train
OKID Train (Est)
OKID Test
OKID Test (Est)

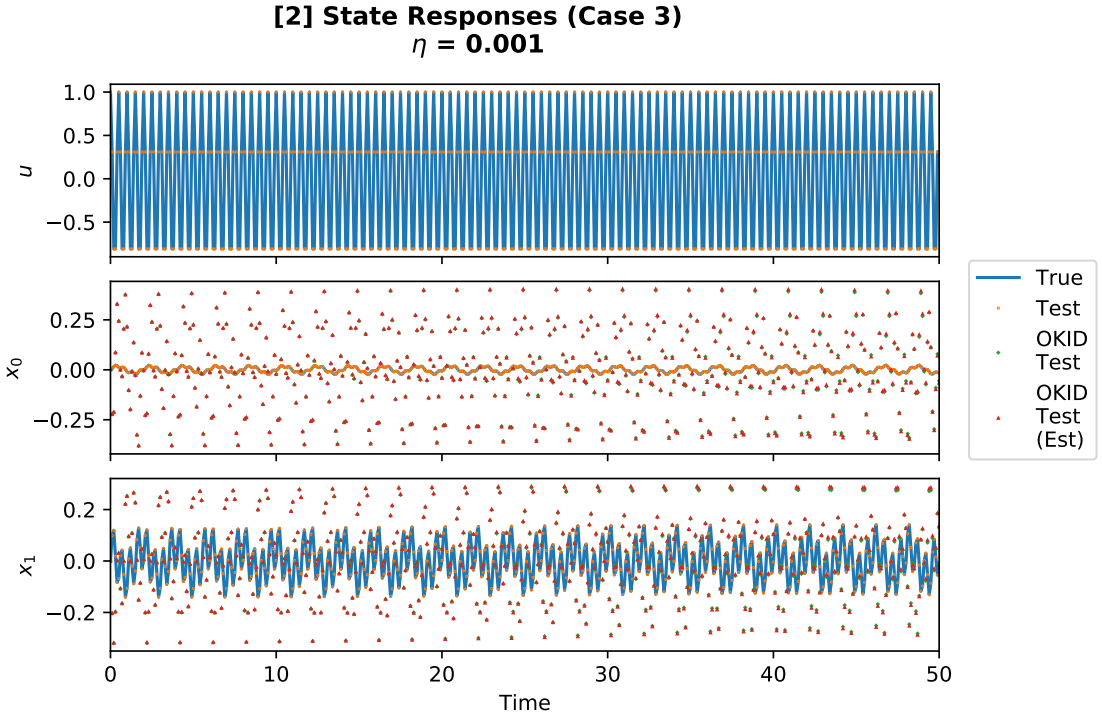

[2] State Responses (Case 1)
$\eta$ = 0.5
$u$
$x_0$
$x_1$
Time
True
Train
Test
OKID Train
OKID Train (Est)
OKID Test
OKID Test (Est)

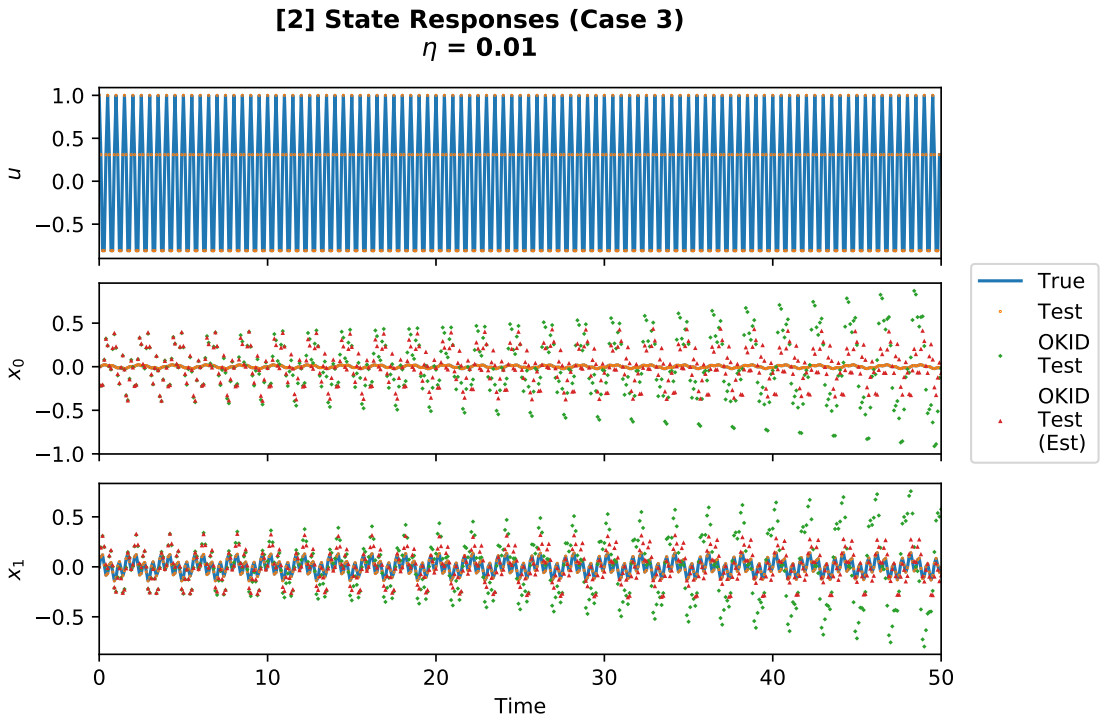

[2] State Responses (Case 2)
$\eta$ = 0.001
$u$
$x_0$
$x_1$
Time
True
Test
OKID Test
OKID Test (Est)

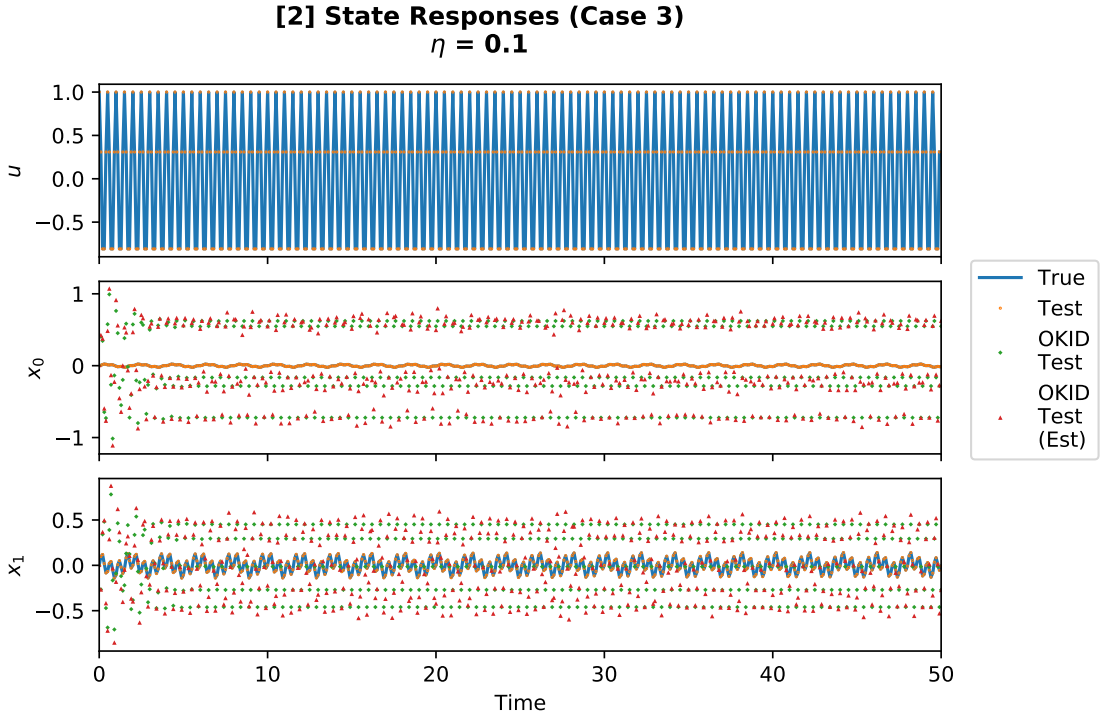

[2] State Responses (Case 2)
η = 0.01
u
x0
x1
Time
True
Test
OKID Test
OKID Test (Est)

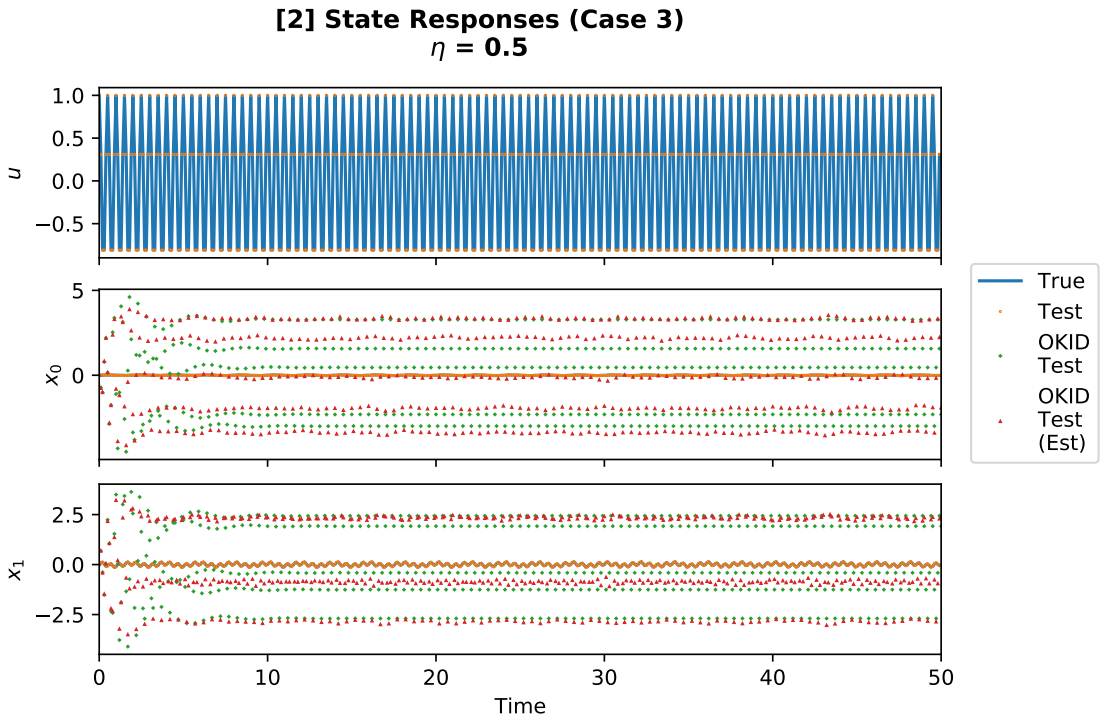

[2] State Responses (Case 2)
η = 0.1
u
x0
x1
Time
True
Test
OKID Test
OKID Test (Est)


[2] State Responses (Case 2)
$\eta$ = 0.5
u
$x_0$
$x_1$
Time
True
Test
OKID Test
OKID Test (Est)


[2] State Responses (Case 3)
$\eta$ = 0.001
u
$x_0$
$x_1$
Time
True
Test
OKID Test
OKID Test (Est)


[2] State Responses (Case 3)
η = 0.01
u
x0
x1
Time
True
Test
OKID Test
OKID Test (Est)


[2] State Responses (Case 3)
η = 0.1
u
x0
x1
Time
True
Test
OKID Test
OKID Test (Est)

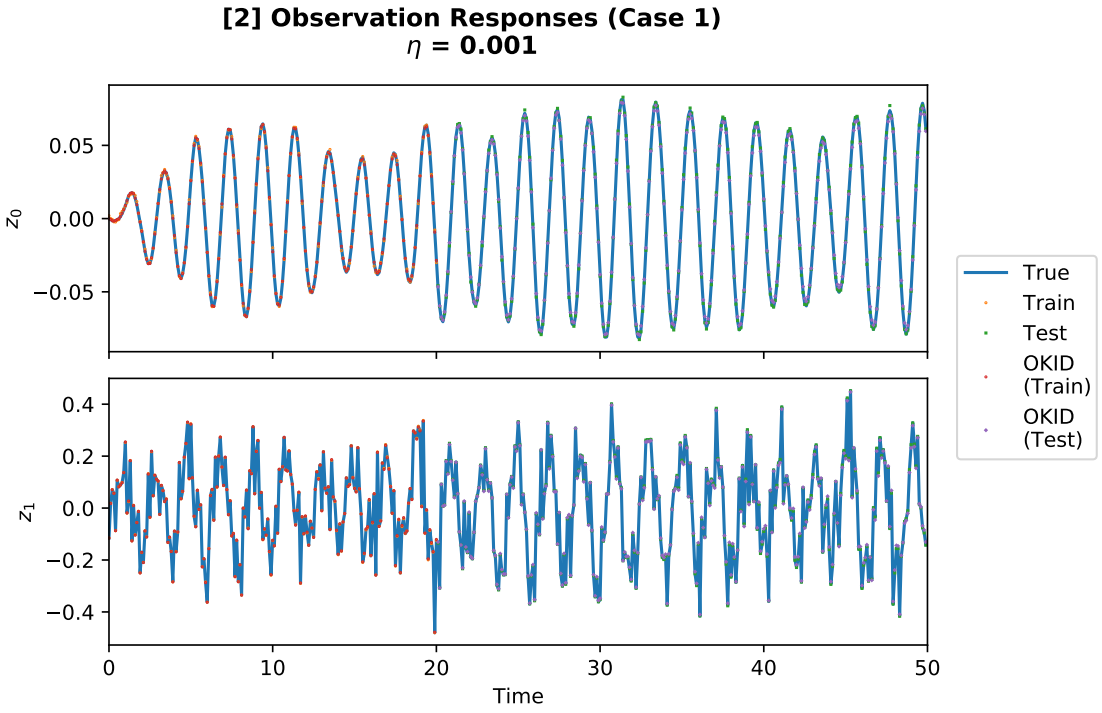


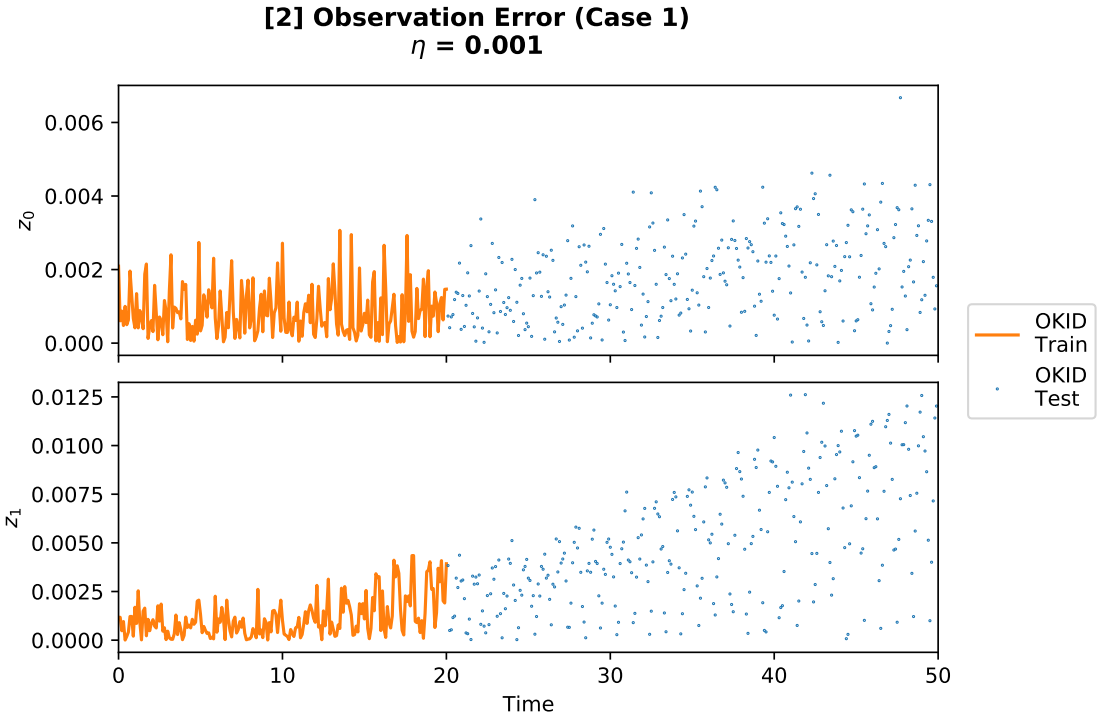

The observer does help the stability of the estimated state when $\eta = 0.01$. However, for $\eta > 0.01$, the observer does not help the estimation in any case examined, as the identified systems are very far from the true systems to begin with.

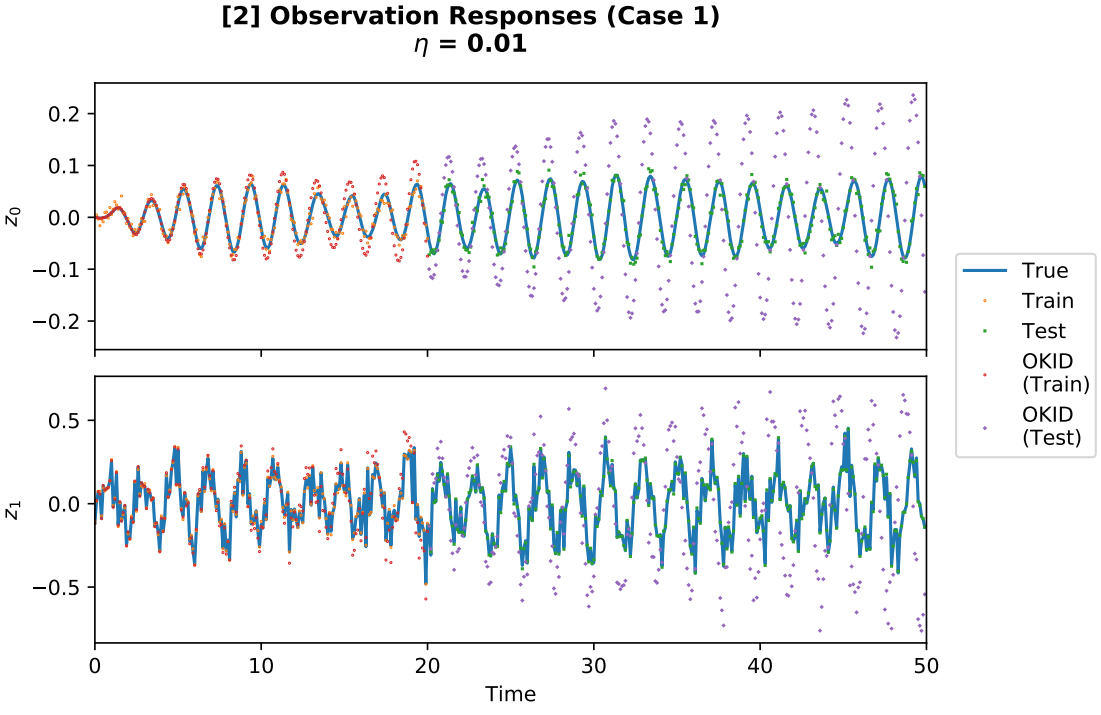
[9]:
```
# Observation plots
for i, k in it.product(range(cases), range(len(noises))):
    # Raw observations
    raw_fig, axs = plt.subplots(m, 1,
                                sharex = "col", constrained_layout = True) #␣
↪type:figure.Figure
    raw_fig.suptitle(f"[{prob}] Observation Responses (Case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

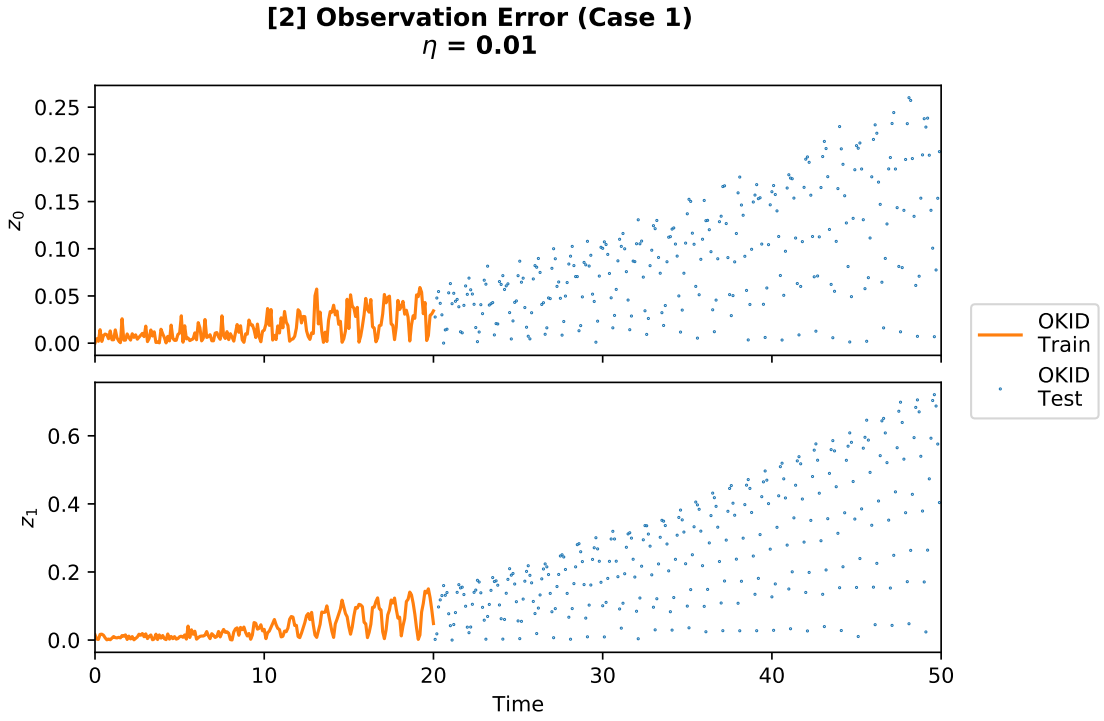
```
                        "D", ms = ms, mfc = "None")
            plt.setp(axs[j], ylabel = f"$z_{j}$",
                     xlim = [0, t_max])
            if j == (m - 1):
                plt.setp(axs[j], xlabel = f"Time")
        raw_fig.legend(labels = ["True", "Train", "Test",
                                 "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):
```

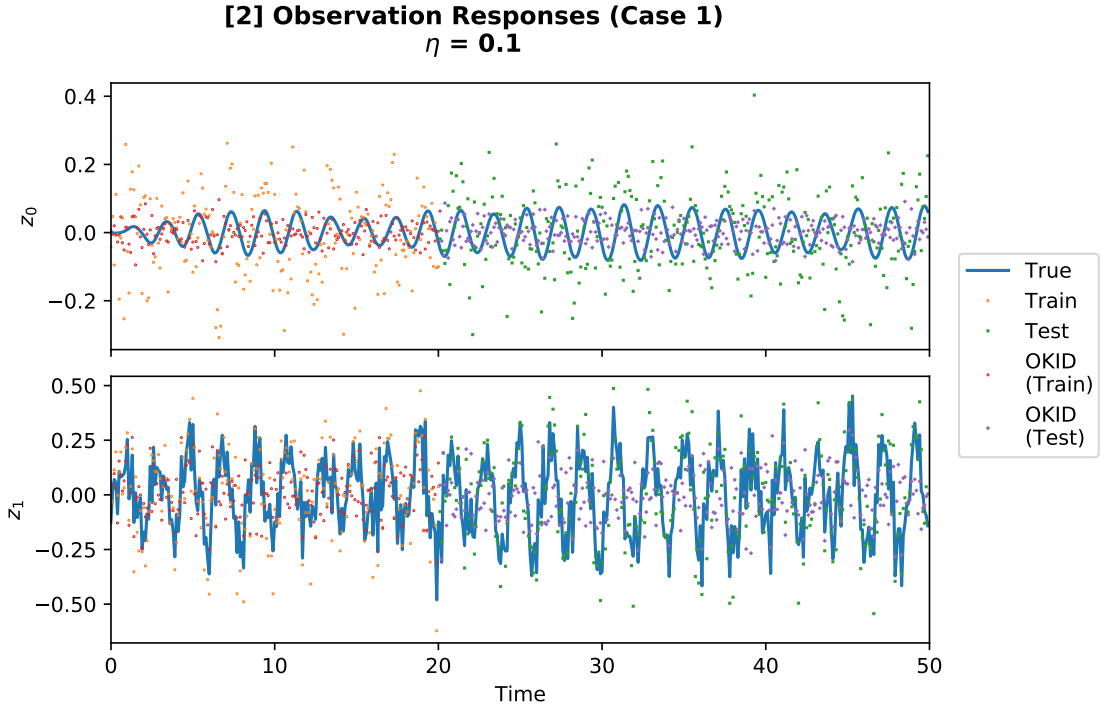
```
            axs[j].plot(t_test[:-1], np.abs(Z_okid_test[k, i, j] - Z_test[k, i,␣
↪j]),
                        "o", ms = ms, mfc = "None")
            plt.setp(axs[j], ylabel = f"$z_{j}$",
                     xlim = [0, t_max])
            if j == (m - 1):
                plt.setp(axs[j], xlabel = 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")
```

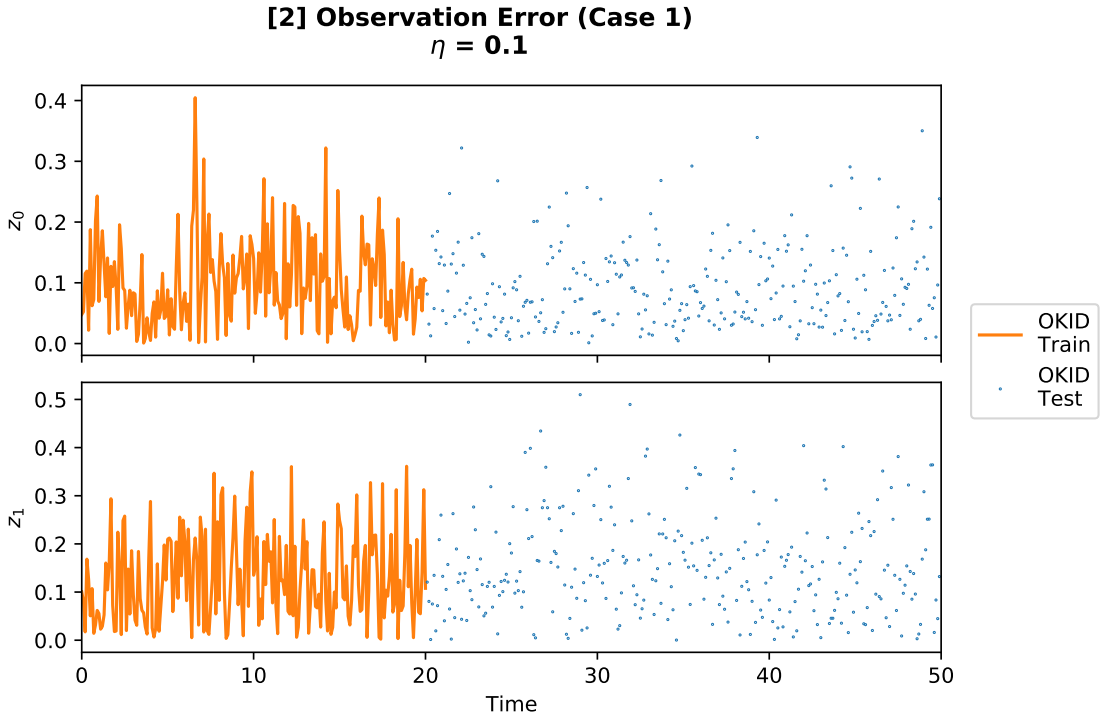


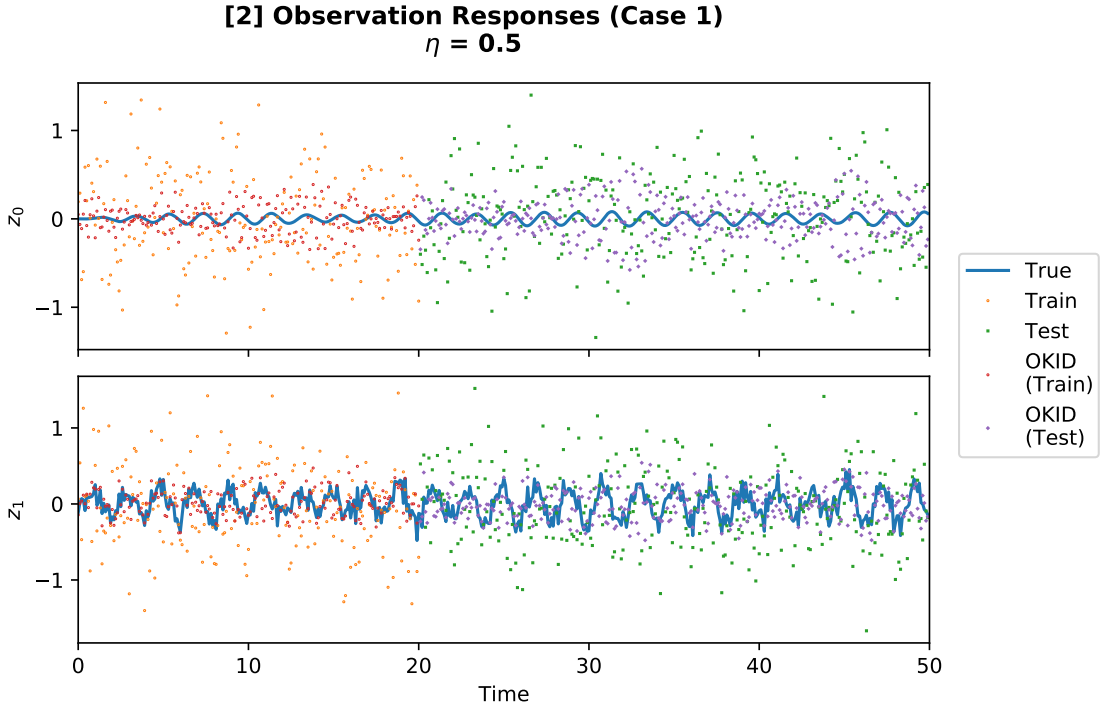

[2] Observation Error (Case 1)
$\eta$ = 0.001
$z_0$
$z_1$
Time
OKID Train
OKID Test

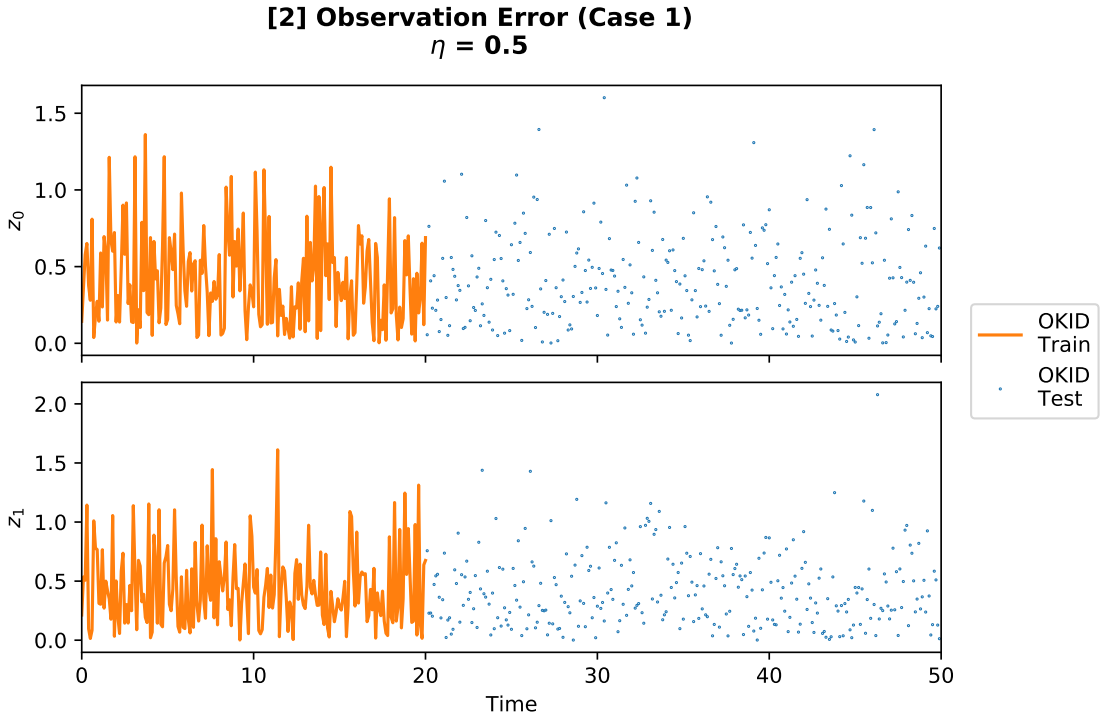

[2] Observation Responses (Case 1)
$\eta$ = 0.01
$z_0$
$z_1$
Time
True
Train
Test
OKID (Train)
OKID (Test)

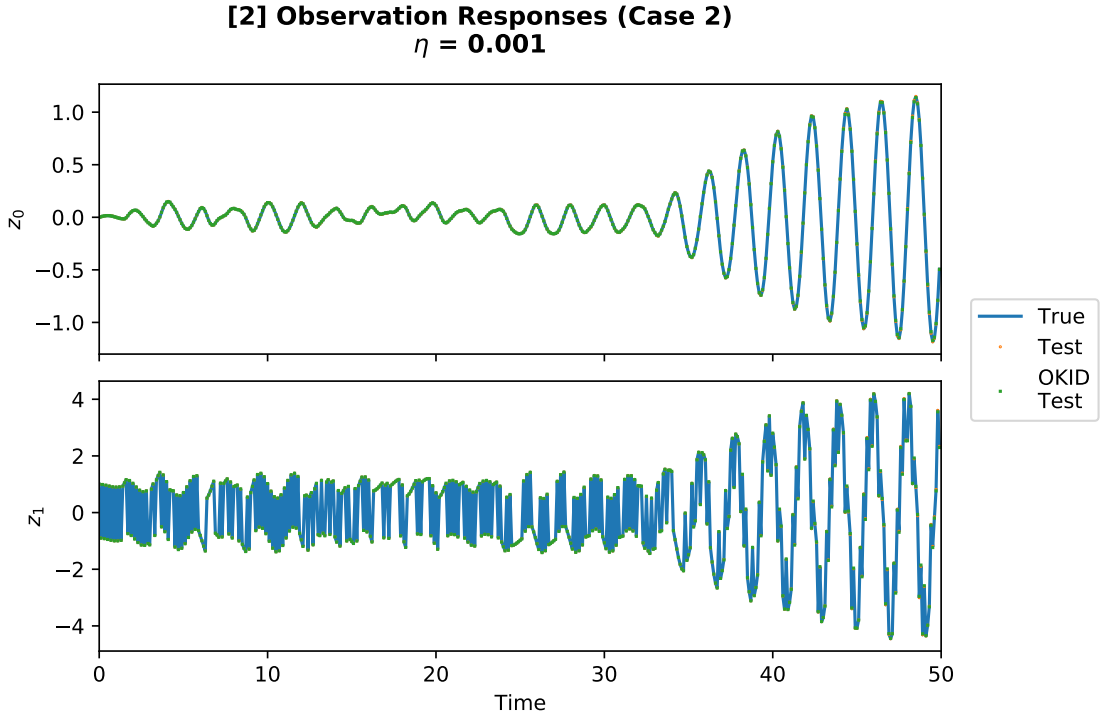

[2] Observation Error (Case 1)
$\eta$ = 0.01
$z_0$
$z_1$
Time
OKID Train
OKID Test

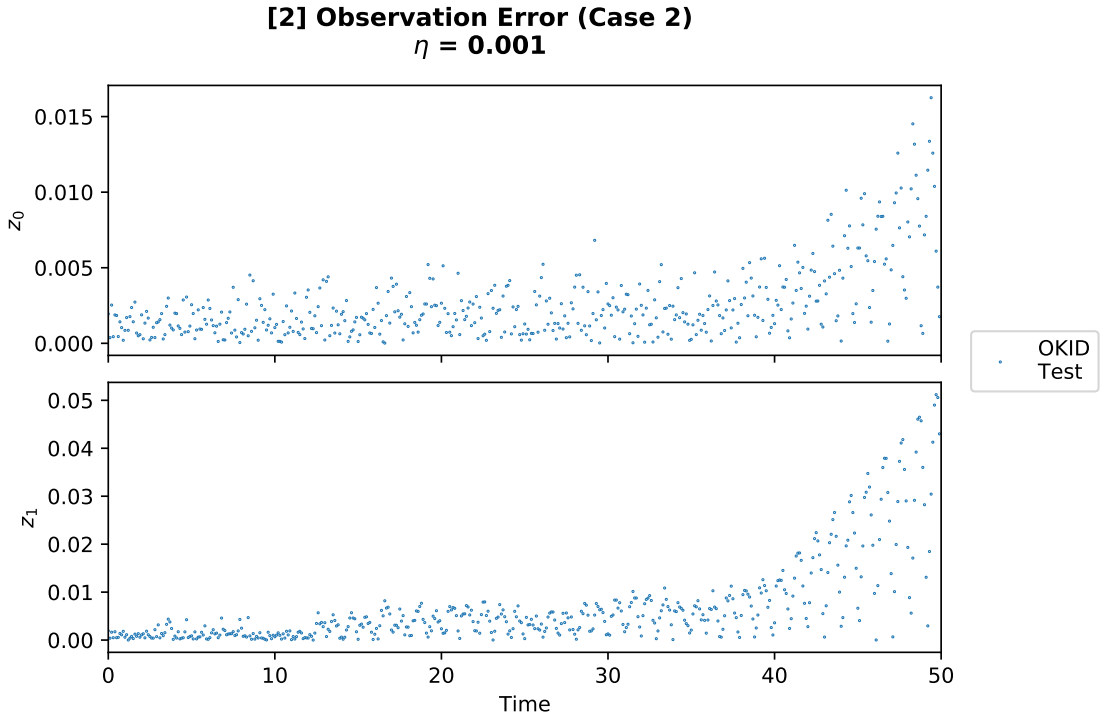

[2] Observation Responses (Case 1)
$\eta$ = 0.1
$z_0$
$z_1$
Time
True
Train
Test
OKID (Train)
OKID (Test)

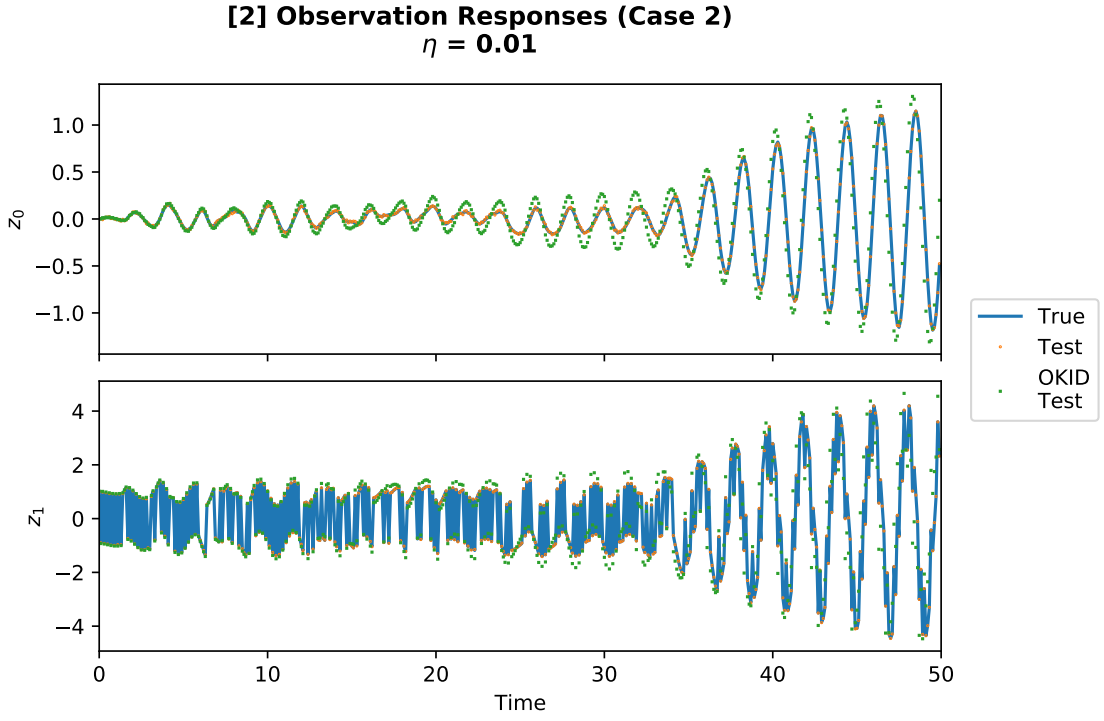

[2] Observation Error (Case 1)
$\eta$ = 0.1
$z_0$
$z_1$
Time
OKID Train
OKID Test

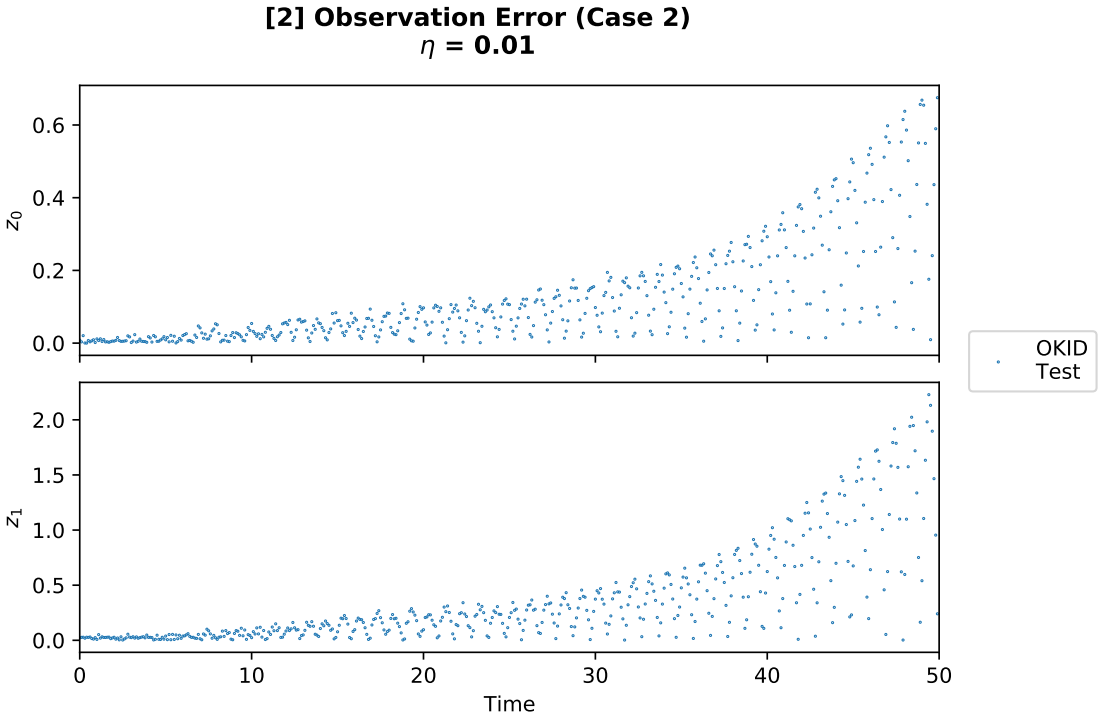

[2] Observation Responses (Case 1)
$\eta$ = 0.5
$z_0$
$z_1$
Time
True
Train
Test
OKID (Train)
OKID (Test)

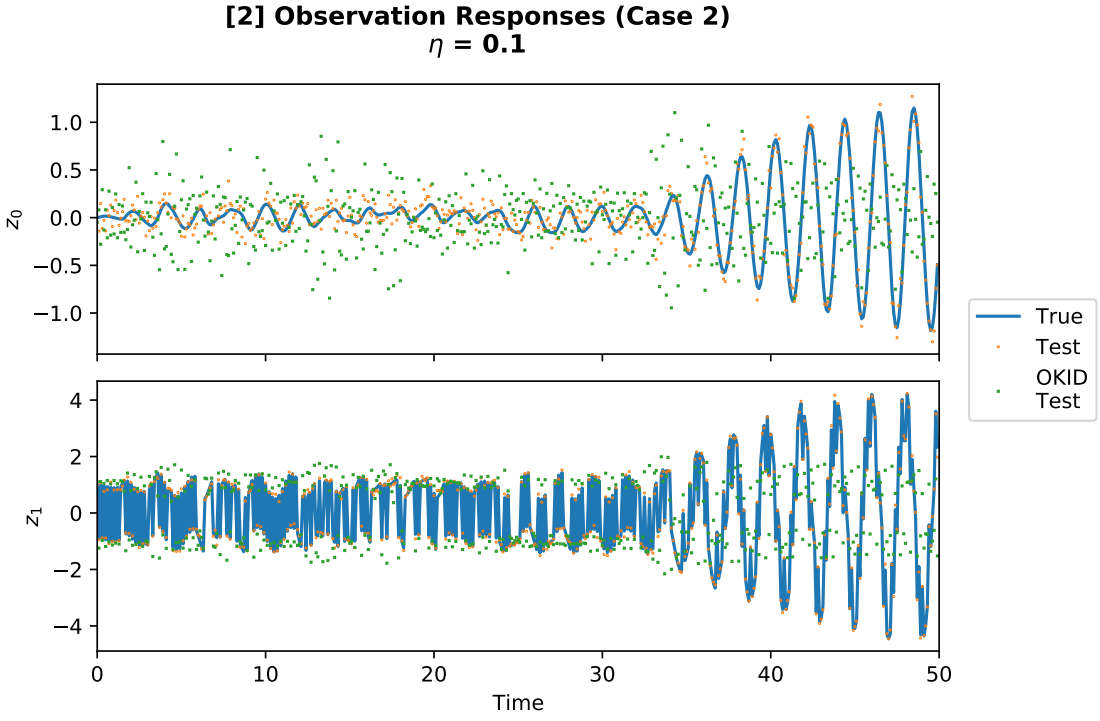

[2] Observation Error (Case 1)
$\eta$ = 0.5
$z_0$
$z_1$
Time
OKID Train
OKID Test

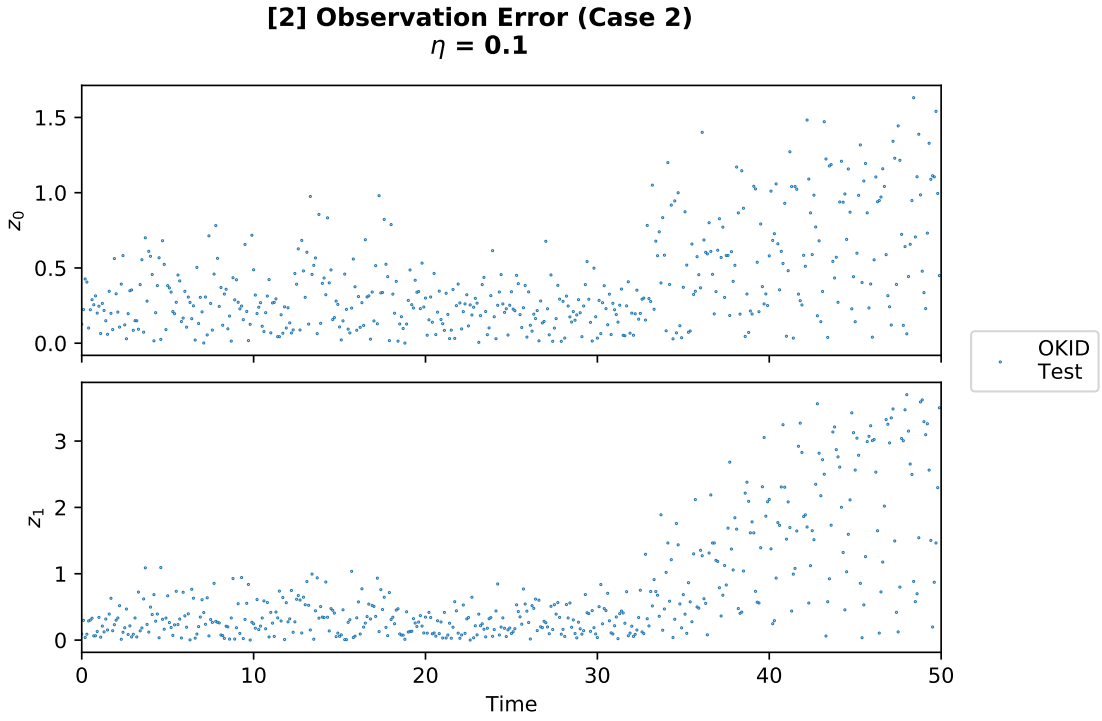

[2] Observation Responses (Case 2)
$\eta$ = 0.001
$z_0$
$z_1$
Time
True
Test
OKID Test

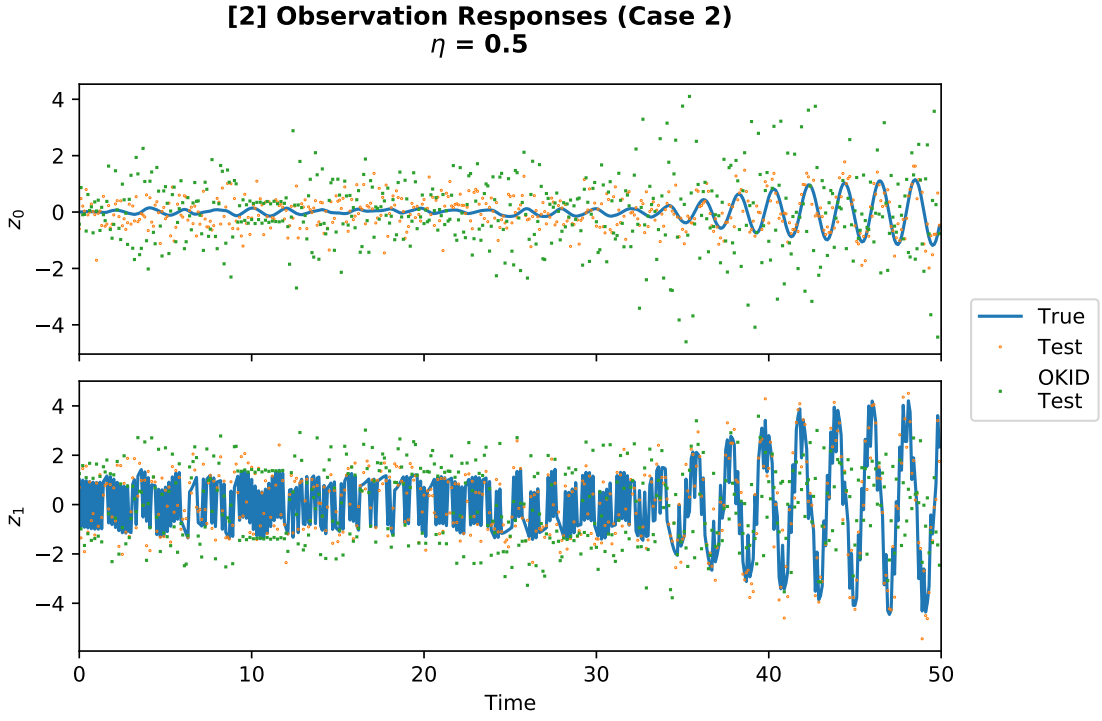

[2] Observation Error (Case 2)
$\eta$ = 0.001
$z_0$
$z_1$
Time
OKID
Test

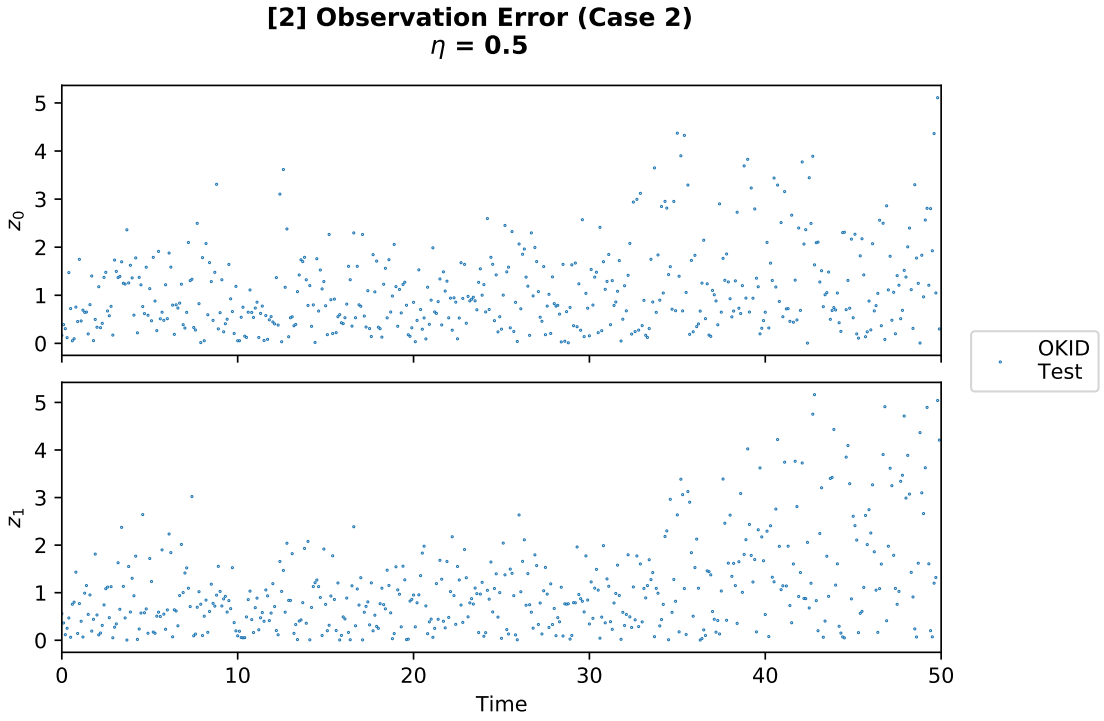

[2] Observation Responses (Case 2)
$\eta$ = 0.01
$z_0$
$z_1$
Time
True
Test
OKID
Test

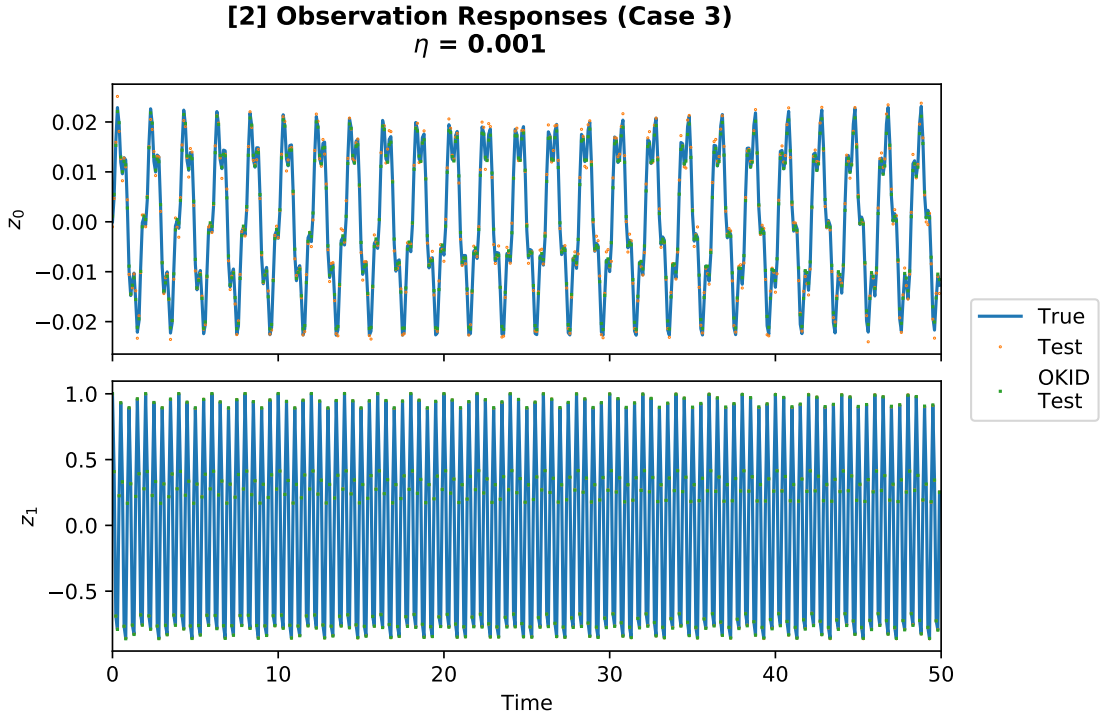

[2] Observation Error (Case 2)
$\eta$ = 0.01
$z_0$
$z_1$
Time
OKID
Test

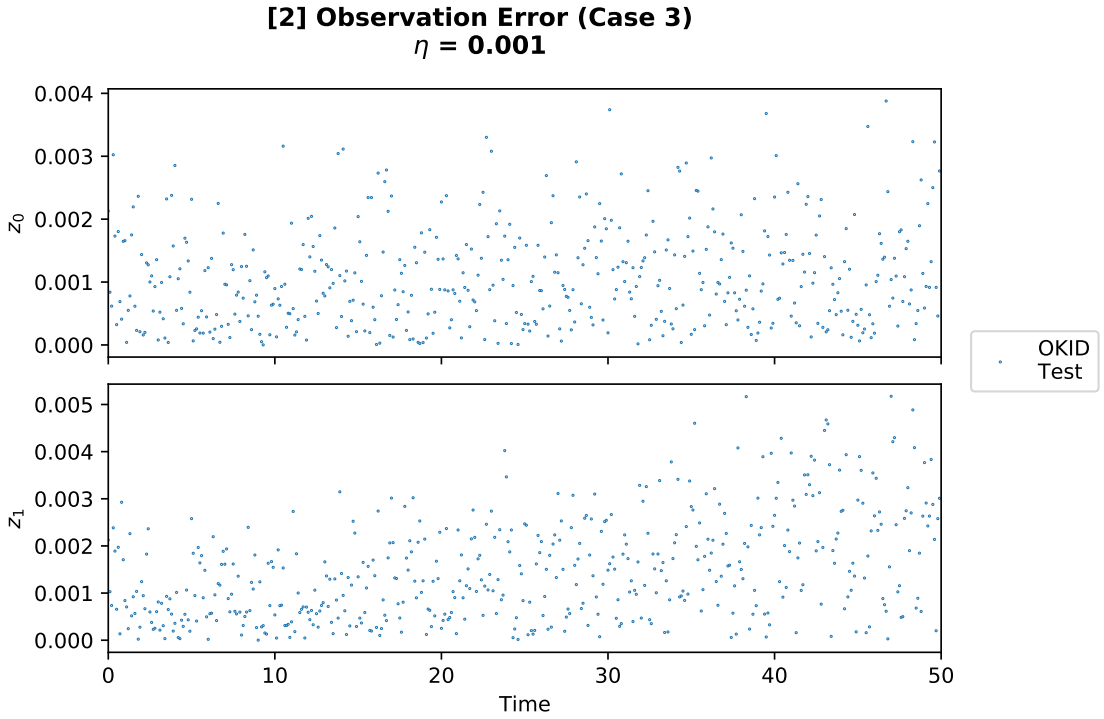

[2] Observation Responses (Case 2)
$\eta$ = 0.1
$z_0$
$z_1$
Time
True
Test
OKID
Test

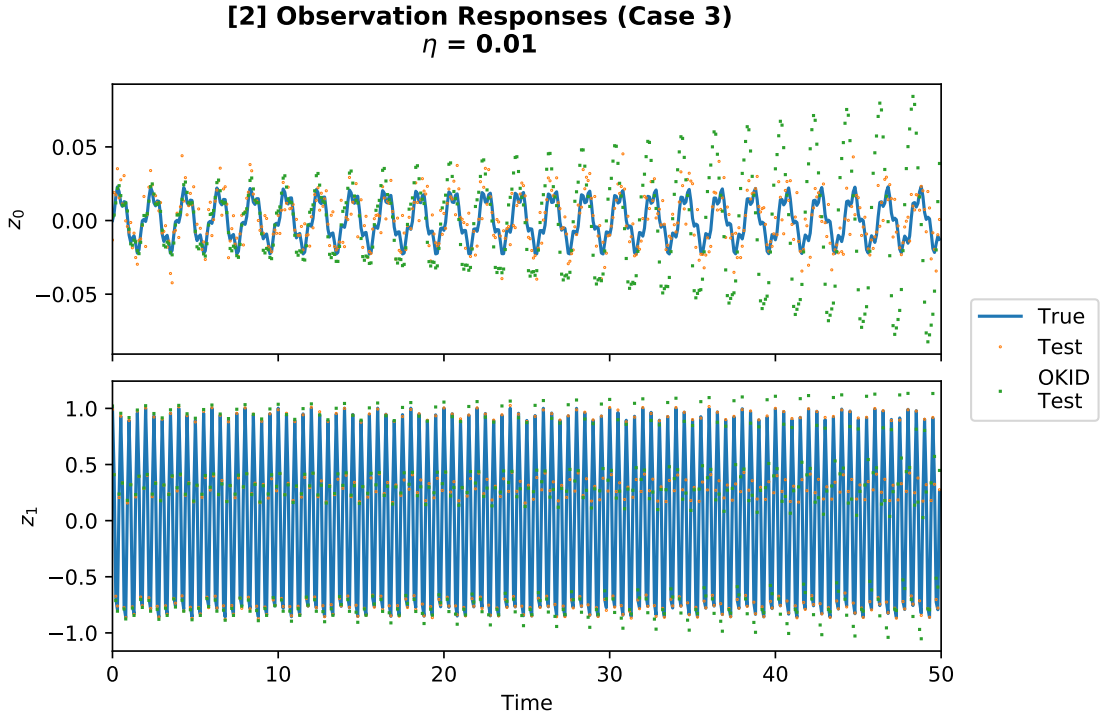

[2] Observation Error (Case 2)
$\eta$ = 0.1
$z_0$
$z_1$
Time
OKID
Test

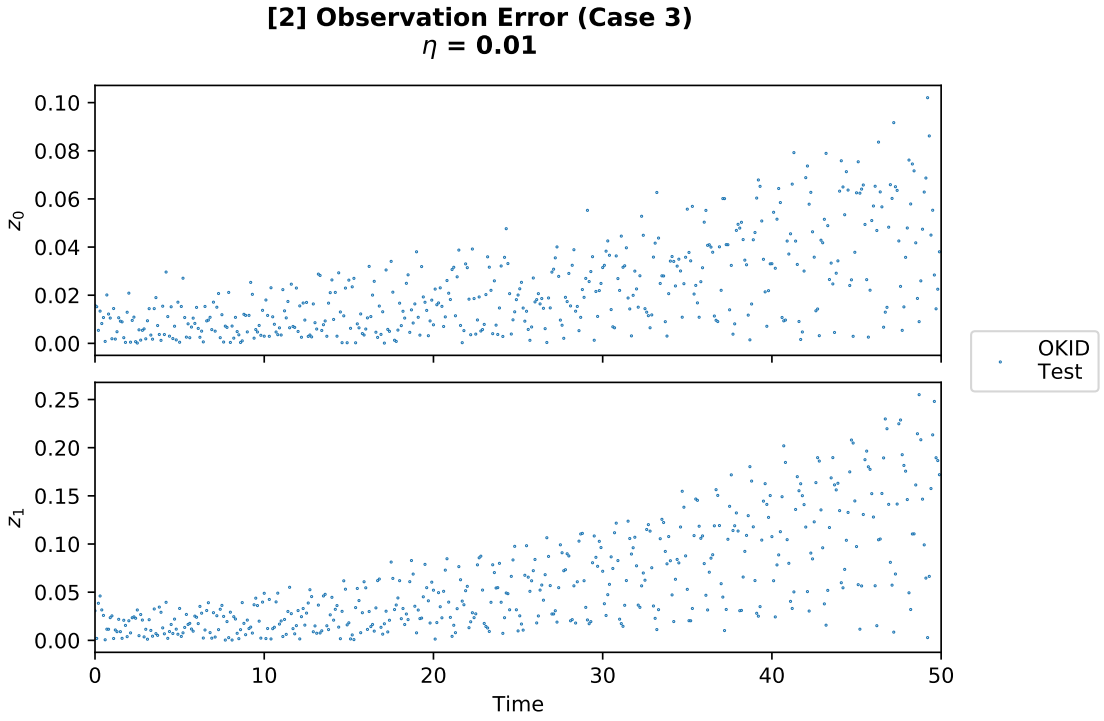

[2] Observation Responses (Case 2)
$\eta$ = 0.5
$z_0$
$z_1$
Time
True
Test
OKID
Test

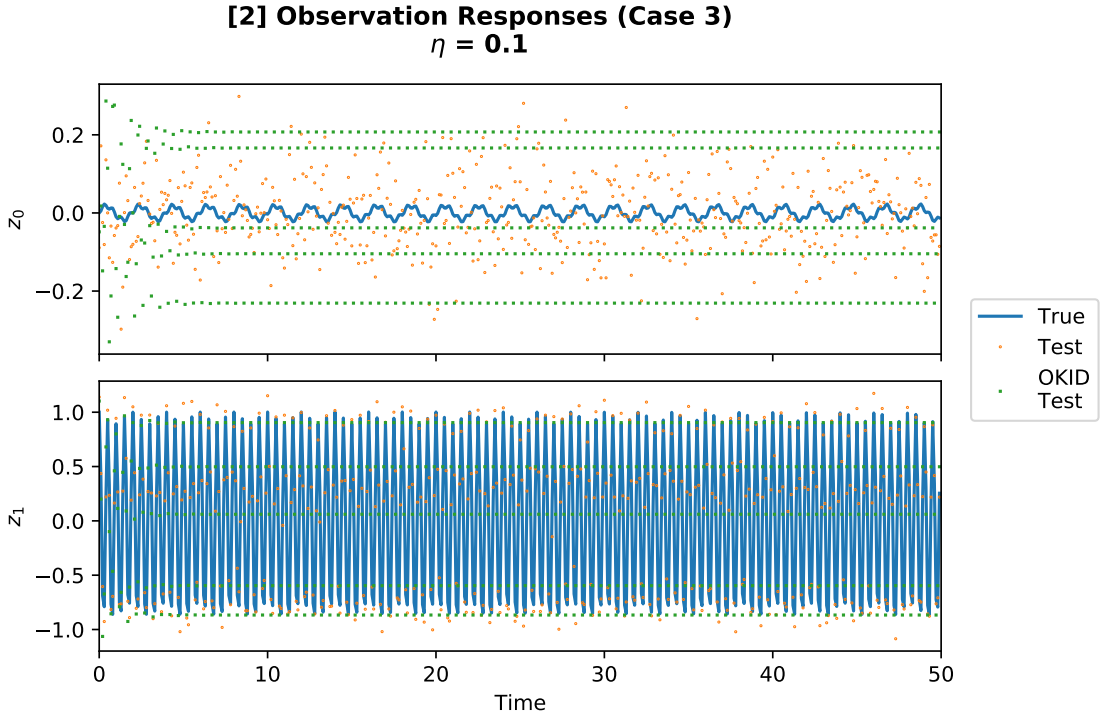

[2] Observation Error (Case 2)
$\eta$ = 0.5
$z_0$
$z_1$
Time
OKID
Test

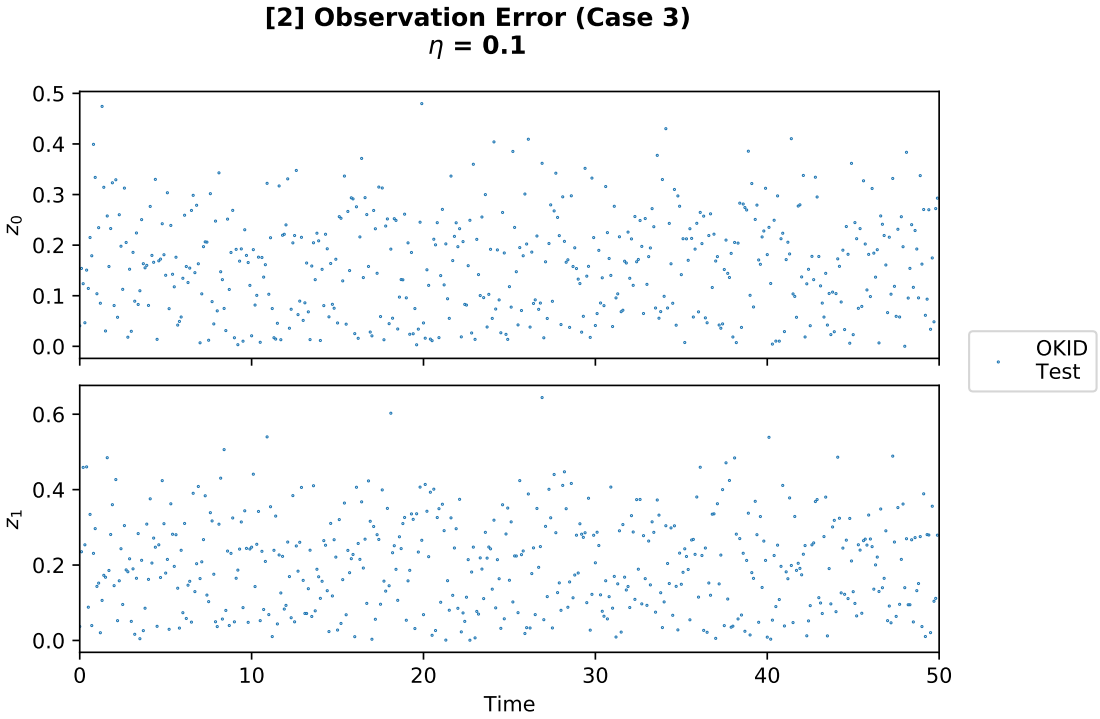

[2] Observation Responses (Case 3)
$\eta$ = 0.001
$z_0$
$z_1$
Time
True
Test
OKID
Test

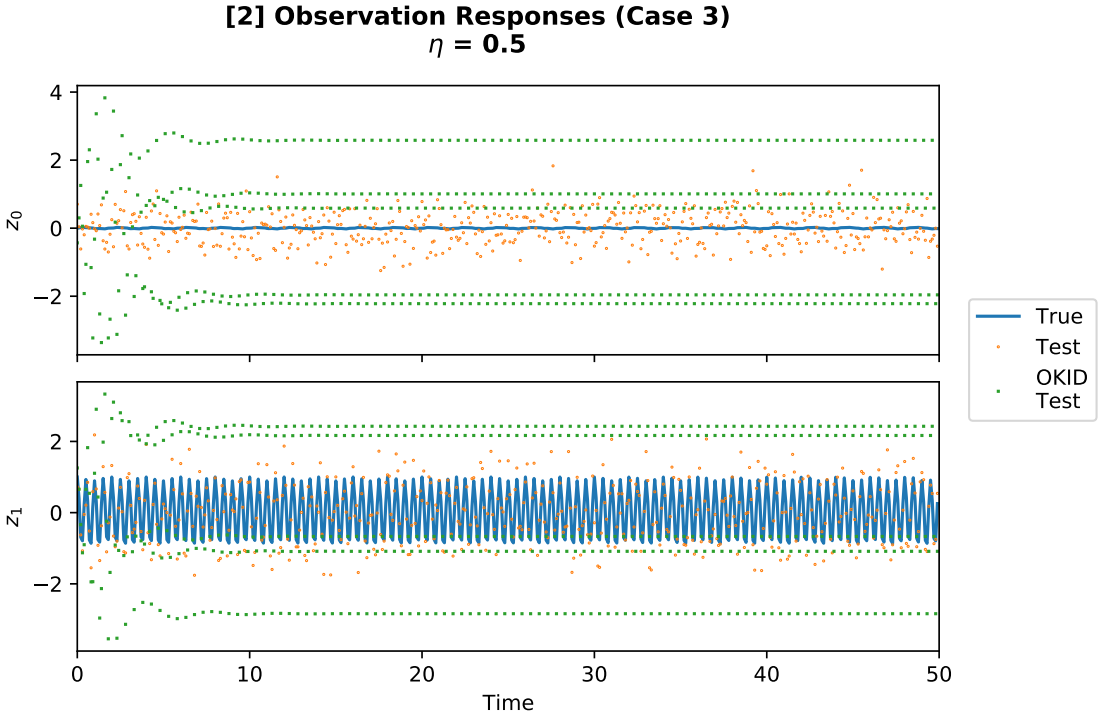

[2] Observation Error (Case 3)
$\eta$ = 0.001
$z_0$
$z_1$
Time
OKID
Test

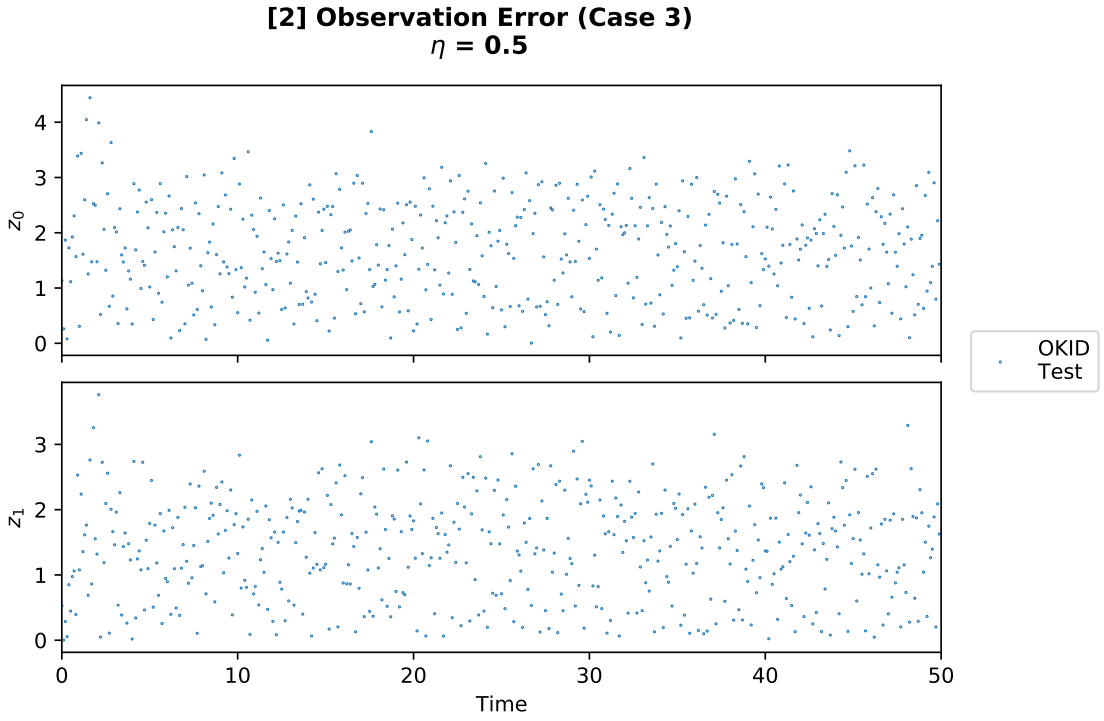

[2] Observation Responses (Case 3)
$\eta$ = 0.01
$z_0$
$z_1$
Time
True
Test
OKID
Test


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


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


[2] Observation Error (Case 3)
$\eta$ = 0.1
$z_0$
$z_1$
Time
OKID
Test


[2] Observation Responses (Case 3)
$\eta$ = 0.5
$z_0$
$z_1$
Time
True
Test
OKID
Test




- $\eta = 0.001$: 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.

# AERSP597 Midterm

Ani Perumalla

April 1, 2021

## 1 Q. #3

[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 = 3
```

[2]:
```
def okid_ic(Z: np.ndarray, U: np.ndarray,
            l_0: int,
            alpha: int, beta: int,
            n: int):
    """Observer Kalman Identification Algorithm (OKID) with nonzero initial␣
↪conditions.

    :param np.ndarray Z: Observation vector array over duration
    :param np.ndarray U: Continual inputs
    :param int l_0: Order of OKID to execute (i.e., number of Markov parameters␣
↪to generate via OKID)
    :param int alpha: Num. of rows of Markov parameters in Hankel matrix
    :param int beta: Num. of columns of Markov parameters in Hankel matrix
    :param int n: Number of proposed states to use for ERA
    :return: (Y) Markov parameters, (Y_og) Observer Gain Markov parameters
    :rtype: typing.Tuple[np.ndarray, np.ndarray]
    """
    r, l_u = U.shape
    m, l = Z.shape
    assert l == l_u
    V = np.concatenate([U, Z], 0)
    assert (max([alpha + beta, (n/m) + (n/r)]) <= l_0) and (l_0 <= (l - r)/(r +␣
↪m)) # Boundary conditions

    # Form observer
```

```
    Y_2_Z = np.zeros([r + (r + m)*l_0, l])
    Y_2_Z[:r, :] = U
    for i in range(1, l_0 + 1):
        Y_2_Z[((i*r) + ((i - 1)*m)):(((i + 1)*r) + (i*m)), :] = np.
↪concatenate([np.zeros([r + m, i]), V[:, 0:(-i)]], 1)
    Y_2_Z_ic = Y_2_Z[:, l_0:]
    # Find Observer Markov parameters via least-squares
    Y_obs = Z[:, l_0:] @ spla.pinv2(Y_2_Z_ic)
    Y_bar_1 = np.array(list(it.chain.from_iterable([Y_obs[:, i:(i + r)]
                                                    for i in range(r, r + (r +␣
↪m)*l_0, r + m)]))).reshape([l_0, m, r])
    Y_bar_2 = -np.array(list(it.chain.from_iterable([Y_obs[:, i:(i + m)]
                                                     for i in range(2*r, r + (r␣
↪+ m)*l_0, r + m)]))).reshape([l_0, m, m])

    # Obtain Markov parameters from Observer Markov parameters
    Y = np.zeros([l_0 + 1, m, r])
    Y[0] = Y_obs[:, :r]
    for k in range(1, l_0 + 1):
        Y[k] = Y_bar_1[k - 1] - \
               np.array([Y_bar_2[i] @ Y[k - (i + 1)]
                         for i in range(k)]).sum(axis = 0)
    # Obtain Observer Gain Markov parameters from Observer Markov parameters
    Y_og = np.zeros([l_0, m, m])
    Y_og[0] = Y_bar_2[0]
    for k in range(1, l_0):
        Y_og[k] = Y_bar_2[k] - \
                  np.array([Y_bar_2[i] @ Y_og[k - (i + 1)]
                            for i in range(k - 1)]).sum(axis = 0)
    return Y, Y_og
```

[3]:
```
# Set seed for consistent results
rng = np.random.default_rng(seed = 100)

# Simulation dimensions
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)))
# Note that damping is always positive even when it is displayed as negative.

# True simulation values
# X_0_sim = np.zeros([n, 1]) # Zero initial condition
X_0_sim = rng.uniform(-1, 1, [n, 1]) # Uniformly random 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

# 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([n, nt_train + 1]) # Train state vector
X_test = np.zeros([cases, n, nt_test + 1]) # Test state vectors
Z_train = np.zeros([m, nt_train]) # Train observation vector
Z_test = np.zeros([cases, m, nt_test]) # Test observation vectors
V_train = np.zeros([r + m, nt_train]) # Train observation input vectors
V_test = np.zeros([cases, r + m, nt_test]) # Test observation input vectors
```

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

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

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

[4]:
```
# OKID logistics
order = 8 # Order of OKID algorithm, number of Markov parameters to identify␣
 ↪after the zeroeth
alpha, beta = 4, 3 # 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

print(f"Min. OKID Order: {max([alpha + beta, (n_era/m) + (n_era/r)]):n}")
print(f"Max. OKID Order: {(nt_train - r)/(r + m):n}")
print(f"Proposed OKID Order: {order:n}")
```

```
Min. OKID Order: 7
Max. OKID Order: 66.6667
Proposed OKID Order: 8
```

To fit the new purpose of OKID with initial condition, we propose to use an OKID order $l_0$ of 8, Hankel height $\alpha = 4$, Hankel width $\beta = 3$, and assumed system order $n_{ERA} = 2$. All of these are in line with what was discussed in the first problem, i.e., the boundary conditions for the order size and the singular values as displayed below.

[5]:
```
# OKID System Markov parameters
Y_okid = np.zeros([order + 1, m, r])
# OKID Observer Markov Gain parameters
Y_og_okid = np.zeros([order, m, m])
# OKID state vector, drawn from state space model derived from OKID/ERA
X_okid_train = np.zeros([n_era, nt_train + 1])
X_okid_test = np.zeros([cases, n_era, nt_test + 1])
X_okid_train_obs = np.zeros([n_era, nt_train + 1])
X_okid_test_obs = np.zeros([cases, n_era, nt_test + 1])
# OKID observations, drawn from state space model derived from OKID/ERA
Z_okid_train = np.zeros([n_era, nt_train])
Z_okid_test = np.zeros([cases, n_era, nt_test])
Z_okid_train_obs = np.zeros([n_era, nt_train])
Z_okid_test_obs = np.zeros([cases, n_era, nt_test])
# Singular values of the Hankel matrix constructed through OKID Markov␣
 ↪parameters
S_okid = np.zeros([min(alpha*m, beta*r)])
eig_A_okid = np.zeros([n_era], dtype = complex)

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

```
C_okid_obs = np.zeros([m, n_era])
D_okid_obs = np.zeros([m, r + m])
```

[6]:
```
# Simulation
for i in range(cases):
    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, Z_train = 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_ic(Z_train, U_train,
                                    l_0 = order, alpha = alpha, beta = beta, n␣
↪= n_era)
        # Identify state space model using System Markov parameters for ERA
        A_okid, B_okid, C_okid, D_okid, S_okid = \
            era(Y_okid, alpha = alpha, beta = beta, n = n_era)
        # Construct observability matrix
        O_p_okid = np.array([C_okid @ np.linalg.matrix_power(A_okid, i)
                             for i in range(order)])
        # Identify initial state
        CABu = np.concatenate([np.zeros([1, n_era, r]),
                               np.array([np.array([C_okid @ np.linalg.
↪matrix_power(A_okid, i) @ B_okid * U_train[0, k - (i + 1)]
                                                   for i in range(k)]).sum(axis␣
↪= 0)
                                         for k in range(1, order)])],
                              axis = 0)
        Du = np.array([D_okid * U_train[0, i]
                       for i in range(order)])
        X_0_id = spla.pinv2(O_p_okid.reshape([order*m, n_era])) @ \
                 (Z_train[:, :order].reshape([order*m, r]) - (CABu + Du).
↪reshape([order*m, r]))
        # Find observer gain matrix
        G_okid = 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 = A_okid + G_okid @ C_okid
        B_okid_obs = np.concatenate([B_okid + G_okid @ D_okid, -G_okid], 1)
        C_okid_obs = C_okid
        D_okid_obs = np.concatenate([D_okid, np.zeros([m, m])], 1)
        V_train = np.concatenate([U_train, Z_train], 0)
        # Simulate OKID realization with "raw" state and OKID realization with␣
↪estimated state
        X_okid_train, Z_okid_train = \
```

```
            sim_ss(A_okid, B_okid, C_okid, D_okid,
                   X_0 = X_0_id, U = U_train, nt = nt_train)
        X_okid_train_obs, Z_okid_train_obs = \
            sim_ss(A_okid_obs, B_okid_obs, C_okid_obs, D_okid_obs,
                   X_0 = X_0_id, U = V_train, nt = nt_train)
        # Display outputs
        etch(f"A_{{OKID}}", A_okid)
        etch(f"B_{{OKID}}", B_okid)
        etch(f"C_{{OKID}}", C_okid)
        etch(f"D_{{OKID}}", D_okid)
        etch(f"G_{{OKID}}", G_okid)
        etch(f"\hat{{x}}_0", X_0_id)
        # Calculate and display eigenvalues
        eig_A_okid = spla.eig(d2c(A_okid, B_okid, dt)[0])[0] # Eigenvalues of␣
↪identified system
        etch(f"\hat{{\lambda}}", eig_A_okid)
        etch(f"\hat{{\omega}}_{{n}}", np.abs(eig_A_okid))
        etch(f"\hat{{\zeta}}", -np.cos(np.angle(eig_A_okid)))
    X_test[i], Z_test[i] = X_sim[i], Z_sim[i]
    X_okid_test[i], Z_okid_test[i] = \
        sim_ss(A_okid, B_okid, C_okid, D_okid,
               X_0 = X_0_id, U = U_test[i], nt = nt_test)
    V_test[i] = np.concatenate([U_test[i], Z_test[i]], 0)
    X_okid_test_obs[i], Z_okid_test_obs[i] = \
        sim_ss(A_okid_obs, B_okid_obs, C_okid_obs, D_okid_obs,
               X_0 = X_0_id, U = V_test[i], nt = nt_test)
```

```
Rank of H(0): 3
Rank of H(1): 3
```

$$A_{OKID} = \begin{bmatrix} 0.73574 & 0.25274 \\ -0.55785 & 1.16755 \end{bmatrix}$$

$$B_{OKID} = \begin{bmatrix} -0.33273 \\ -0.13524 \end{bmatrix}$$

$$C_{OKID} = \begin{bmatrix} -0.0444 & 0.07257 \\ -0.31405 & 0.0452 \end{bmatrix}$$

$$D_{OKID} = \begin{bmatrix} 0.0 \\ 1.0 \end{bmatrix}$$

$$G_{OKID} = \begin{bmatrix} -0.24474 & 0.92035 \\ -1.22873 & 0.34024 \end{bmatrix}$$

$$\hat{x}_0 = \begin{bmatrix} -0.839 \\ 5.77474 \end{bmatrix}$$

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

$$\hat{\omega}_n = \begin{bmatrix} 3.1225 \\ 3.1225 \end{bmatrix}$$

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

The eigenvalues of the system, as well as the corresponding natural frequencies and damping ratios, are accurately identified via the new OKID formulation for the system with nonzero initial condition. This makes sense, since changing the initial condition of the system does not change the underlying dynamics of the system, allowing OKID to identify the modes well.

[7]:
```
RMS_train = np.sqrt(np.mean((Z_okid_train - Z_train)**2, axis = 1))
print(f"RMS Error of sim. for system found via OKID for train data:␣
 ↪{RMS_train}")
RMS_test = np.zeros([cases, m])
for i in range(cases):
    RMS_test[i] = np.sqrt(np.mean((Z_okid_test[i] - Z_test[i])**2, axis = 1))
    print(f"RMS Error of sim. for system found via OKID for test data, case {i}:
 ↪ {RMS_test[i]}")
```

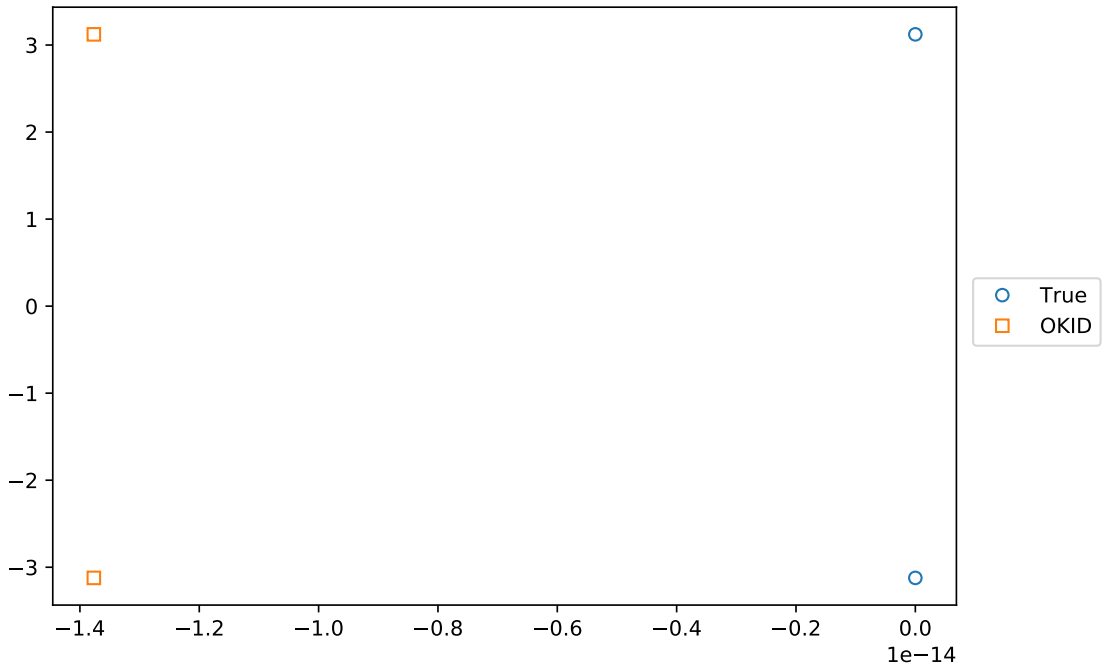
```
RMS Error of sim. for system found via OKID for train data: [0.1685842
0.52691283]
RMS Error of sim. for system found via OKID for test data, case 0: [0.16824344
0.52797367]
RMS Error of sim. for system found via OKID for test data, case 1: [0.16824344
0.52797367]
RMS Error of sim. for system found via OKID for test data, case 2: [0.16824344
0.52797367]
```

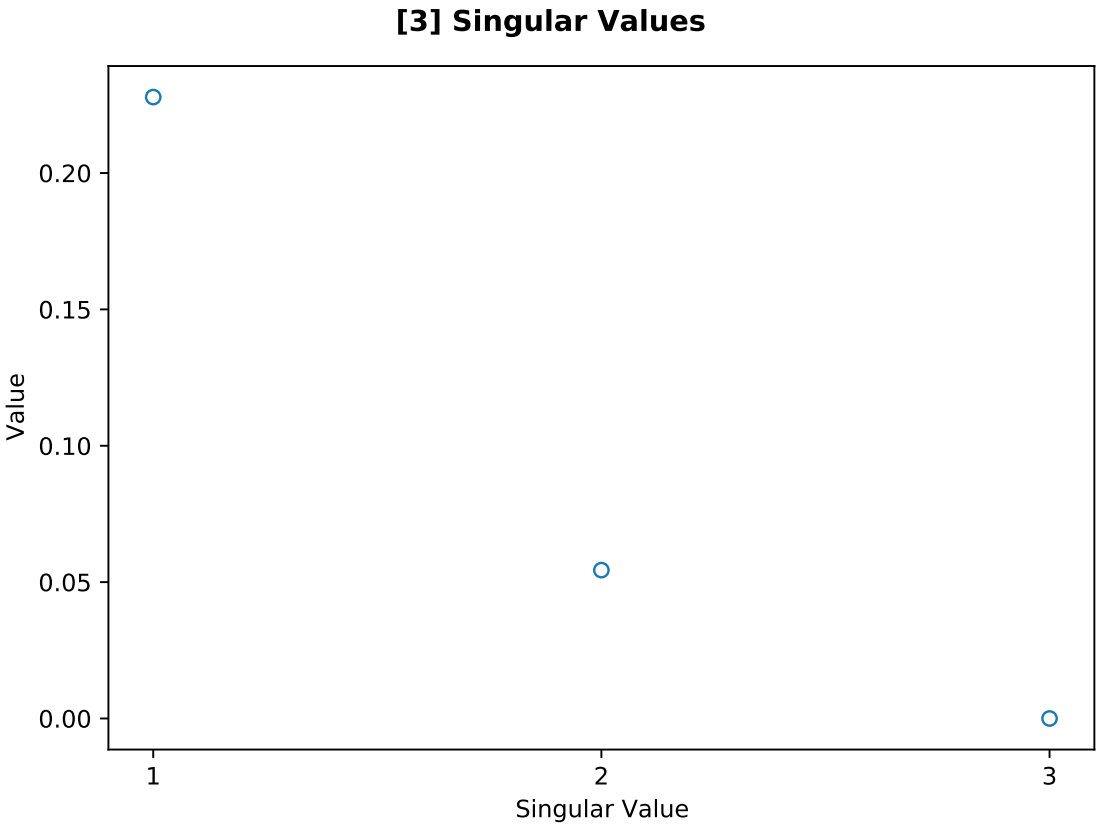
[8]:
```
# Eigenvalue plots
fig, ax = plt.subplots(constrained_layout = True) # type:figure.Figure
fig.suptitle(f"[{prob}] Eigenvalues", fontweight = "bold")

ax.plot(np.real(eig_A), np.imag(eig_A),
         "o", mfc = "None")
ax.plot(np.real(eig_A_okid), np.imag(eig_A_okid),
         "s", mfc = "None")

fig.legend(labels = ("True", "OKID"),
           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")

ax.plot(np.linspace(1, len(S_okid), len(S_okid)), S_okid,
         "o", mfc = "None")
```

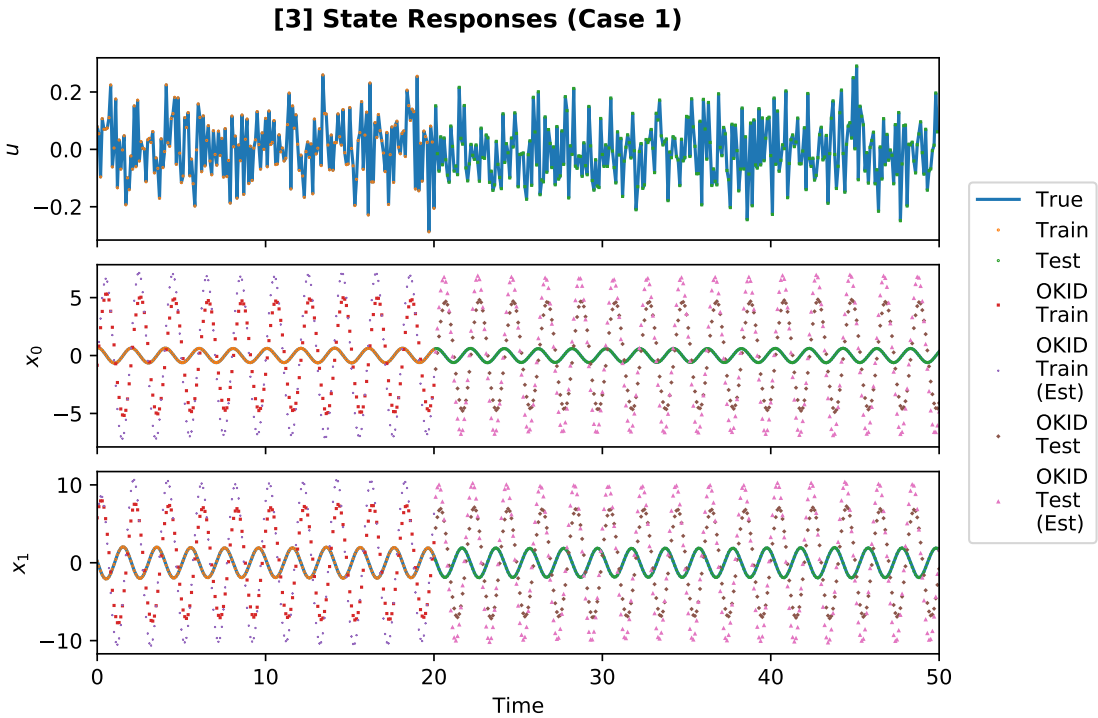
```
plt.setp(ax, xlabel = f"Singular Value", ylabel = f"Value",
         xticks = np.arange(1, len(S_okid) + 1))

fig.savefig(figs_dir / f"midterm_{prob}_singval.pdf",
            bbox_inches = "tight")

# Response plots
ms = 0.5 # Marker size
for i in range(cases):
    fig, axs = plt.subplots(1 + n, 1,
                            sharex = "col",
                            constrained_layout = True) # type:figure.Figure
    fig.suptitle(f"[{prob}] State Responses (Case {i + 1})",
                 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[j],
                            "o", ms = ms, mfc = "None")
            axs[j + 1].plot(t_test[train_cutoff:], X_test[i, j, train_cutoff:],
                            "o", ms = ms, mfc = "None")
            axs[j + 1].plot(t_train, X_okid_train[j, :-1],
                            "s", ms = ms, mfc = "None")
            axs[j + 1].plot(t_train, X_okid_train_obs[j, :-1],
                            "*", ms = ms, mfc = "None")
            axs[j + 1].plot(t_test[train_cutoff:], X_okid_test[i, j,␣
↪train_cutoff:],
                            "D", ms = ms, mfc = "None")
            axs[j + 1].plot(t_test[train_cutoff:], X_okid_test_obs[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:
```

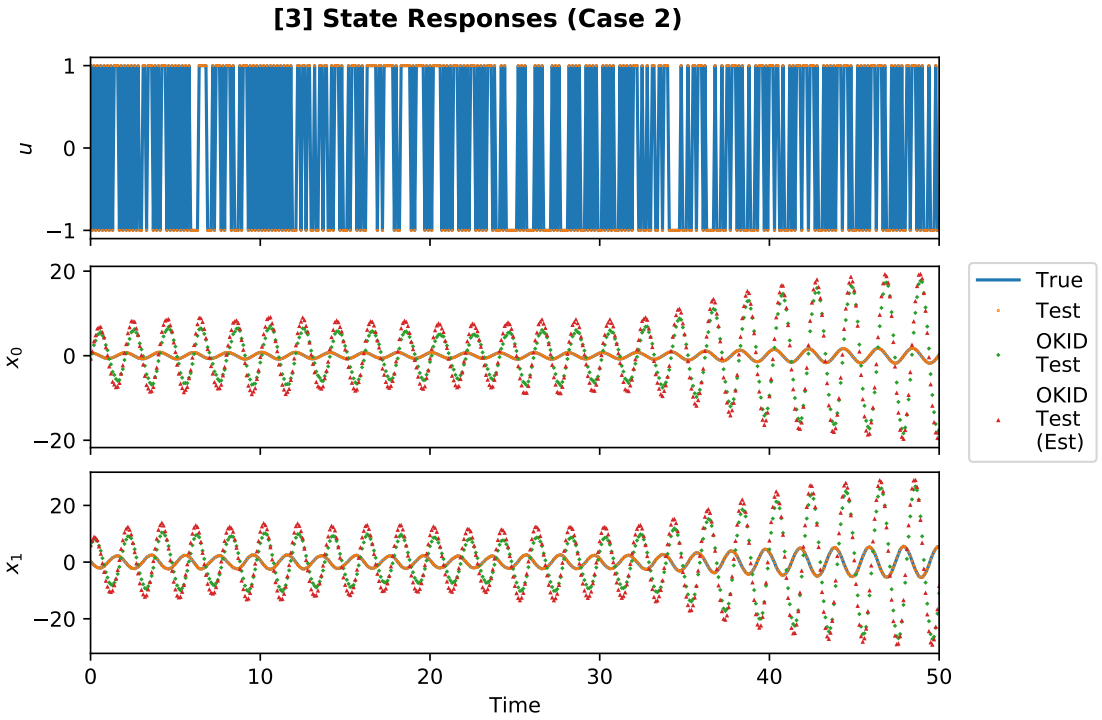
```
        axs[0].plot(t_sim[:-1], U_sim[i, 0])
        axs[0].plot(t_test[:-1], U_test[i, 0],
                    "o", ms = ms, mfc = "None")
        plt.setp(axs[0], ylabel = f"$u$", xlim = [0, t_max])

        for j in range(n):
            axs[j + 1].plot(t_sim, X_sim[i, j])
            axs[j + 1].plot(t_test, X_test[i, j],
                            "o", ms = ms, mfc = "None")
            axs[j + 1].plot(t_test, X_okid_test[i, j],
                            "D", ms = ms, mfc = "None")
            axs[j + 1].plot(t_test, X_okid_test_obs[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}.pdf",
                bbox_inches = "tight")
```

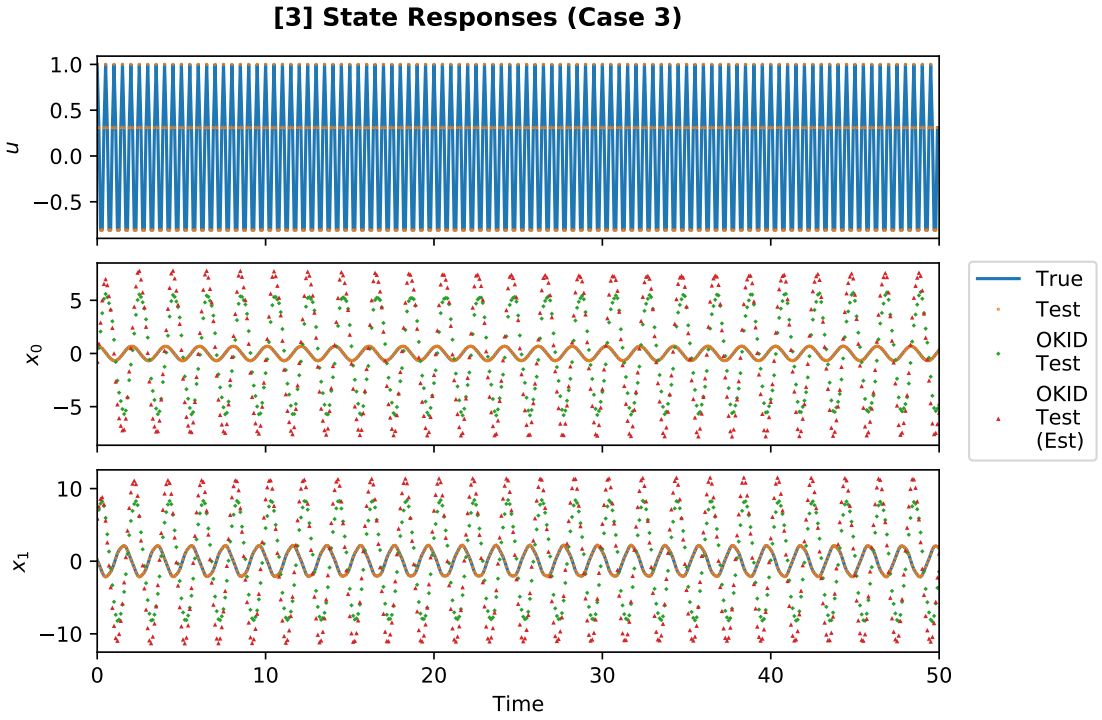
**[3] Eigenvalues**




[3] Singular Values
Value
0.20
0.15
0.10
0.05
0.00
1
2
3
Singular Value


[3] State Responses (Case 1)
u
x0
x1
Time
True
Train
Test
OKID Train
OKID Train (Est)
OKID Test
OKID Test (Est)

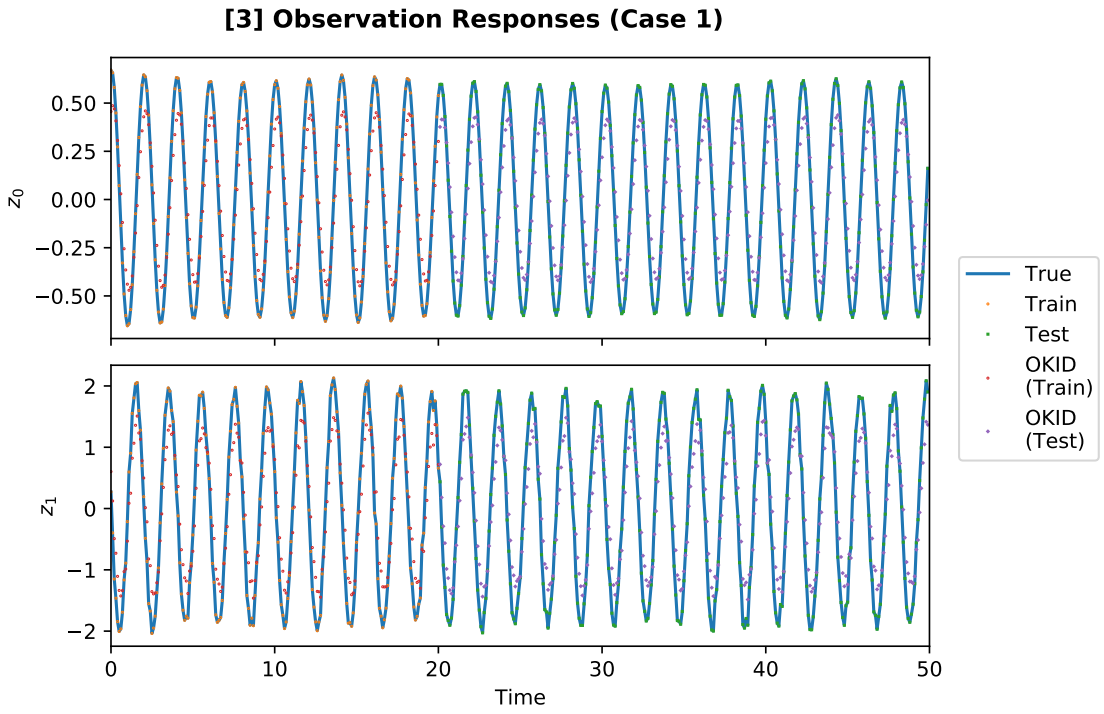

[3] State Responses (Case 2)
u
x0
x1
Time
True
Test
OKID Test
OKID Test (Est)

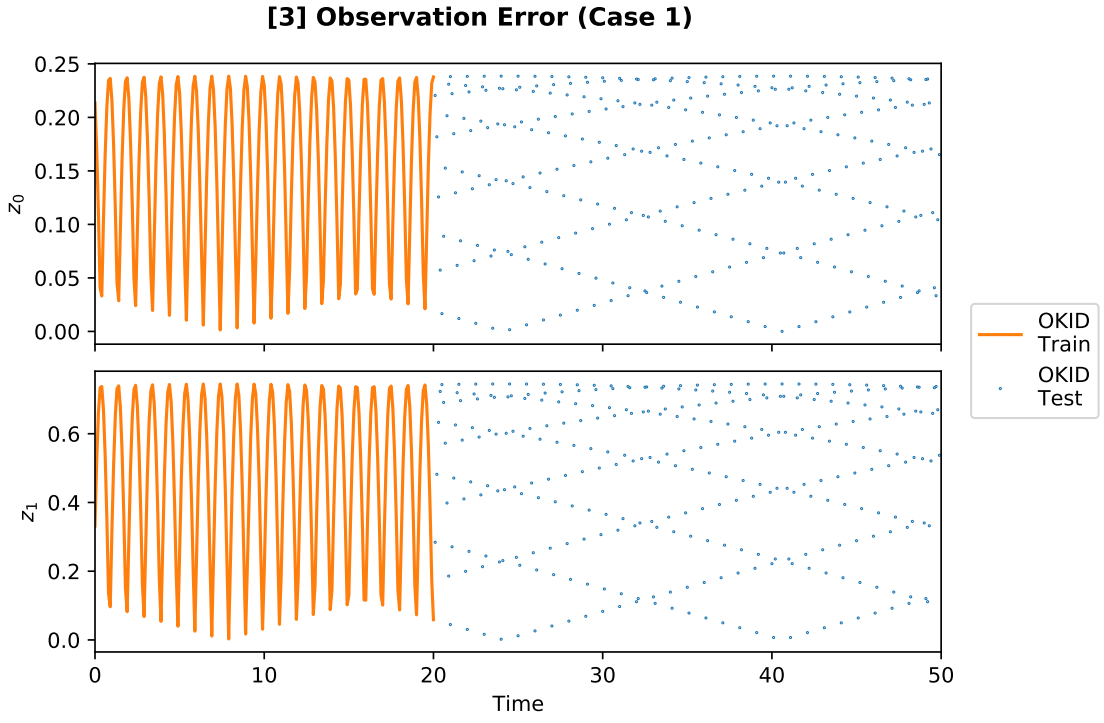

[3] State Responses (Case 3)
u
x0
x1
Time
True
Test
OKID Test
OKID Test (Est)

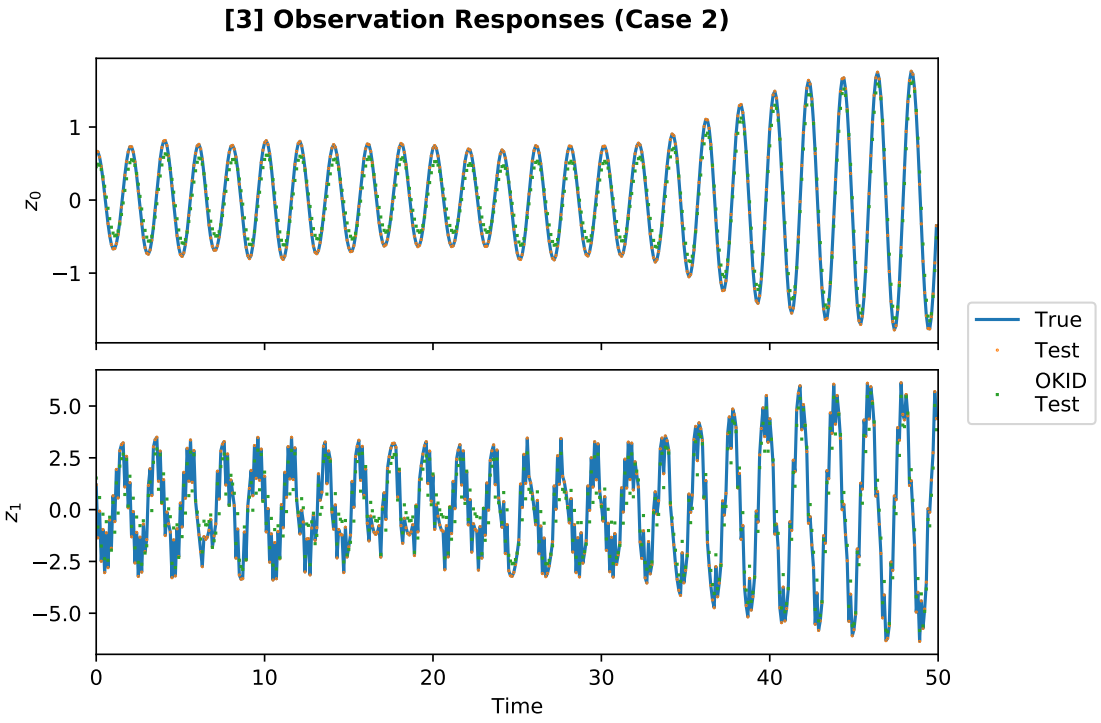
The identified states appear to have a larger magnitude in general than the assumed true states, as depicted in the plots above. It is difficult to judge the impact the new formulation had on the state estimate and observer, since we have not found a transformation between the assumed true states of $x_0(t)$ and $x_1(t)$ and the identified states. If we had calculated a transformation matrix, it would be possible to verify whether the estimated initial condition was close to the true initial condition.

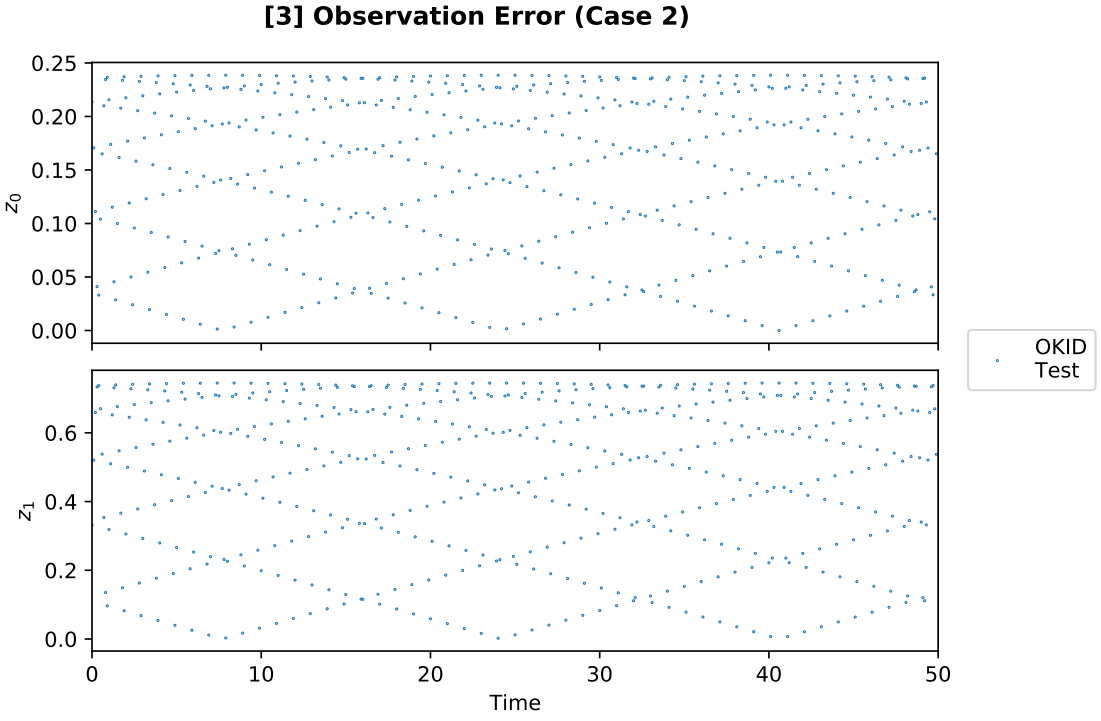
[9]:
```
# Observation plots
for i in range(cases):
    # Raw observations
    fig, axs = plt.subplots(m, 1,
                            sharex = "col",
                            constrained_layout = True) # type:figure.Figure
    fig.suptitle(f"[{prob}] Observation Responses (Case {i + 1})",
                 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[j],
                        "o", ms = ms, mfc = "None")
            axs[j].plot(t_test[train_cutoff:-1], Z_test[i, j, train_cutoff:],
                        "s", ms = ms, mfc = "None")
            axs[j].plot(t_train, Z_okid_train[j],
                        "o", ms = ms, mfc = "None")
            axs[j].plot(t_test[train_cutoff:-1], Z_okid_test[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")
        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[i, j],
                        "o", ms = ms, mfc = "None")
            axs[j].plot(t_test[:-1], Z_okid_test[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")
```

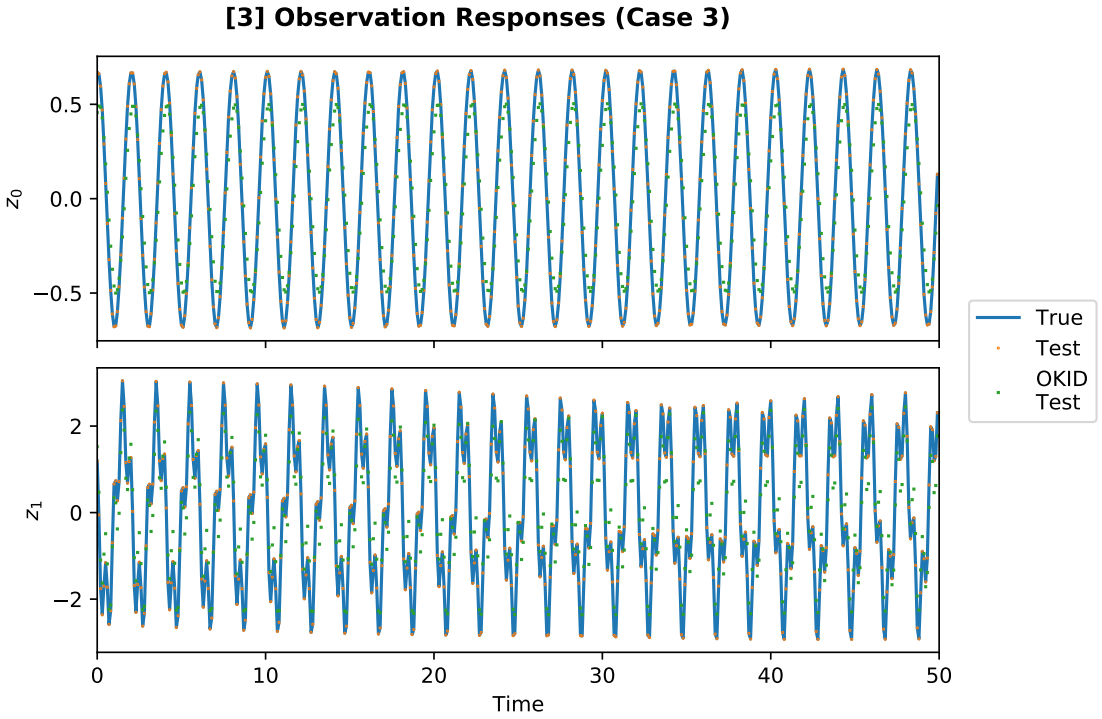
```
        fig.legend(labels = ["True", "Test", "OKID\nTest"],
                   bbox_to_anchor = (1, 0.5), loc = 6)
    fig.savefig(figs_dir / f"midterm_{prob}_obs_case{i + 1}.pdf",
                bbox_inches = "tight")

    # Observation error
    fig, axs = plt.subplots(m, 1,
                            sharex = "col",
                            constrained_layout = True) # type:figure.Figure
    fig.suptitle(f"[{prob}] Observation Error (Case {i + 1})",
                 fontweight = "bold")
    if i == 0:
        for j in range(m):
            axs[j].plot(t_train, np.abs(Z_okid_train[j] - Z_train[j]),
                        c = "C1")
            axs[j].plot(t_test[train_cutoff:-1], np.abs(Z_okid_test[i, j,␣
↪train_cutoff:] - Z_test[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")
        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[i, j] - Z_test[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")
        fig.legend(labels = ["OKID\nTest"],
                   bbox_to_anchor = (1, 0.5), loc = 6)
    fig.savefig(figs_dir / f"midterm_{prob}_obs-error_case{i + 1}.pdf",
                bbox_inches = "tight")
```

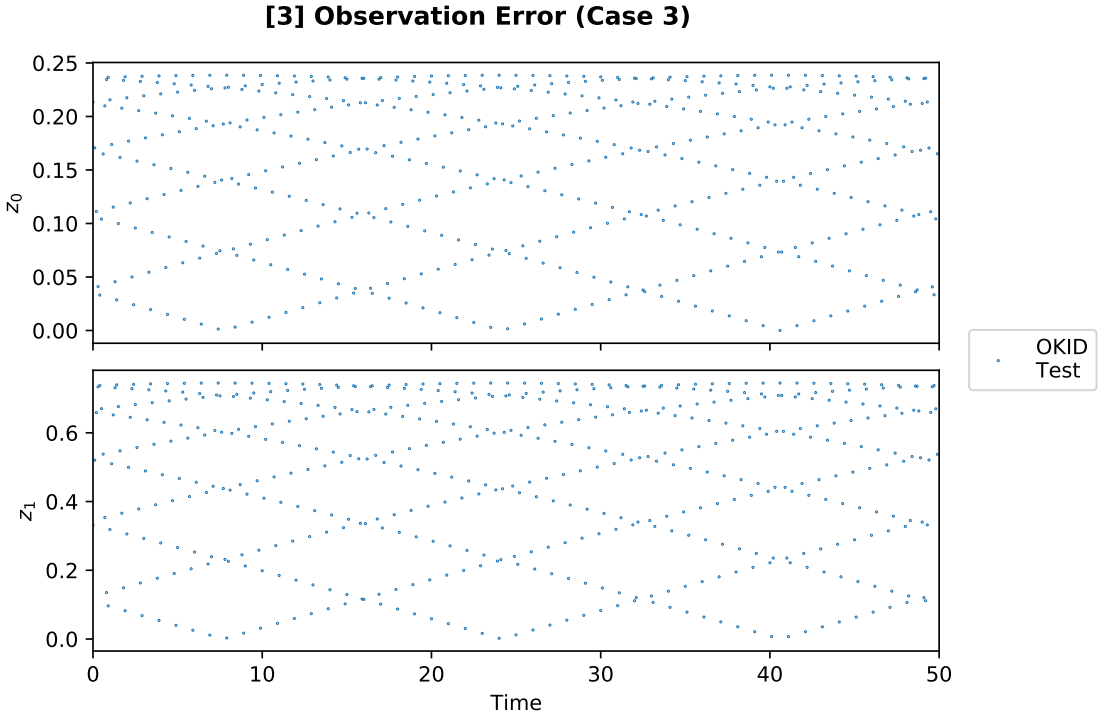

[3] Observation Responses (Case 1)
$z_0$
$z_1$
Time
True
Train
Test
OKID (Train)
OKID (Test)


[3] Observation Error (Case 1)
$z_0$
$z_1$
Time
OKID Train
OKID Test


[3] Observation Responses (Case 2)
$z_0$
$z_1$
Time
True
Test
OKID
Test


[3] Observation Error (Case 2)
$z_0$
$z_1$
Time
OKID
Test


[3] Observation Responses (Case 3)
$z_0$
$z_1$
Time
True
Test
OKID
Test


[3] Observation Error (Case 3)
$z_0$
$z_1$
Time
OKID
Test

The observation sequence is not identified as well as it was in problem 1, especially the second observation $z_1$. This could be because the new formulation meant that we trimmed the $V$ matrix from the classical OKID formulation to only $l - l_0$ columns, meaning we had less data for the least-squares estimation of the Markov parameters.

# 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 = "4-1"
```

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

# Simulation dimensions
orders = (15, 30, 60) # Order of OKID algorithm, number of Markov parameters to␣
 ↪identify after the zeroeth
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(orders), 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(orders), n, nt_train]) # Train state vector
X_test = np.zeros([len(orders), cases, n, nt_test + 1]) # Test state vectors
Z_train = np.zeros([len(orders), m, nt_train]) # Train observation vector
Z_test = np.zeros([len(orders), cases, m, nt_test]) # Test observation vectors
V_train = np.zeros([len(orders), r + m, nt_train]) # Train observation input␣
↪vectors
V_test = np.zeros([len(orders), 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
alpha, beta = 3, 3 # 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
```

Note that we have set $\alpha = 3$ and $\beta = 3$ for this simulation. We choose $l_0 = \{15, 30, 60\}$ to determine the effect of observer order on the results of the system identification.

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

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

[5]:
```
# Simulation
for i, j in it.product(range(cases), range(len(orders))):
    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 = orders[j], 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, 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(orders[j])])
        # Find observer gain matrix
        G_okid[j] = spla.pinv2(O_p_okid.reshape([orders[j]*m, n_era])) @␣
↪Y_og_okid.reshape([orders[j]*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}}(l_0 = {orders[j]})", A_okid[j])
        etch(f"B_{{OKID}}(l_0 = {orders[j]})", B_okid[j])
        etch(f"C_{{OKID}}(l_0 = {orders[j]})", C_okid[j])
        etch(f"D_{{OKID}}(l_0 = {orders[j]})", D_okid[j])
        etch(f"G_{{OKID}}(l_0 = {orders[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}}(l_0 = {orders[j]})", eig_A_okid[j])
        etch(f"\hat{{\omega}}_{{n}}(l_0 = {orders[j]})", np.abs(eig_A_okid[j]))
        etch(f"\hat{{\zeta}}(l_0 = {orders[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): 3
Rank of H(1): 3
Rank of H(0): 3
Rank of H(1): 3
Rank of H(0): 3
```

```
Rank of H(1): 3
```

$$A_{OKID}(l_0 = 15) = \begin{bmatrix} 0.74973 & 0.28172 \\ -0.4797 & 1.15356 \end{bmatrix}$$

$$B_{OKID}(l_0 = 15) = \begin{bmatrix} -0.3226 \\ -0.11535 \end{bmatrix}$$

$$C_{OKID}(l_0 = 15) = \begin{bmatrix} -0.04576 & 0.08499 \\ -0.32048 & 0.04339 \end{bmatrix}$$

$$D_{OKID}(l_0 = 15) = \begin{bmatrix} 0.0 \\ 1.0 \end{bmatrix}$$

$$G_{OKID}(l_0 = 15) = \begin{bmatrix} -0.20329 & 0.53199 \\ -0.45948 & 0.08979 \end{bmatrix}$$

$$\hat{\lambda}(l_0 = 15) = \begin{bmatrix} 3.1225i \\ -3.1225i \end{bmatrix}$$

$$\hat{\omega}_n(l_0 = 15) = \begin{bmatrix} 3.1225 \\ 3.1225 \end{bmatrix}$$

$$\hat{\zeta}(l_0 = 15) = \begin{bmatrix} 0.0 \\ 0.0 \end{bmatrix}$$

$$A_{OKID}(l_0 = 30) = \begin{bmatrix} 0.74973 & 0.28172 \\ -0.4797 & 1.15356 \end{bmatrix}$$

$$B_{OKID}(l_0 = 30) = \begin{bmatrix} -0.3226 \\ -0.11535 \end{bmatrix}$$

$$C_{OKID}(l_0 = 30) = \begin{bmatrix} -0.04576 & 0.08499 \\ -0.32048 & 0.04339 \end{bmatrix}$$

$$D_{OKID}(l_0 = 30) = \begin{bmatrix} 0.0 \\ 1.0 \end{bmatrix}$$

$$G_{OKID}(l_0 = 30) = \begin{bmatrix} -0.05438 & 0.32373 \\ -0.1717 & 0.07943 \end{bmatrix}$$

$$\hat{\lambda}(l_0 = 30) = \begin{bmatrix} 3.1225i \\ -3.1225i \end{bmatrix}$$

$$\hat{\omega}_n(l_0 = 30) = \begin{bmatrix} 3.1225 \\ 3.1225 \end{bmatrix}$$

$$\hat{\zeta}(l_0 = 30) = \begin{bmatrix} 0.0 \\ 0.0 \end{bmatrix}$$

$$A_{OKID}(l_0 = 60) = \begin{bmatrix} 0.74973 & 0.28172 \\ -0.4797 & 1.15356 \end{bmatrix}$$

$$B_{OKID}(l_0 = 60) = \begin{bmatrix} -0.3226 \\ -0.11535 \end{bmatrix}$$

$$C_{OKID}(l_0 = 60) = \begin{bmatrix} -0.04576 & 0.08499 \\ -0.32048 & 0.04339 \end{bmatrix}$$

$$D_{OKID}(l_0 = 60) = \begin{bmatrix} 0.0 \\ 1.0 \end{bmatrix}$$

$$G_{OKID}(l_0 = 60) = \begin{bmatrix} -0.02721 & 0.22937 \\ -0.08045 & 0.05929 \end{bmatrix}$$

$$\hat{\lambda}(l_0 = 60) = \begin{bmatrix} 3.1225i \\ -3.1225i \end{bmatrix}$$

$$\hat{\omega}_n(l_0 = 60) = \begin{bmatrix} 3.1225 \\ 3.1225 \end{bmatrix}$$

$$\hat{\zeta}(l_0 = 60) = \begin{bmatrix} 0.0 \\ 0.0 \end{bmatrix}$$

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.

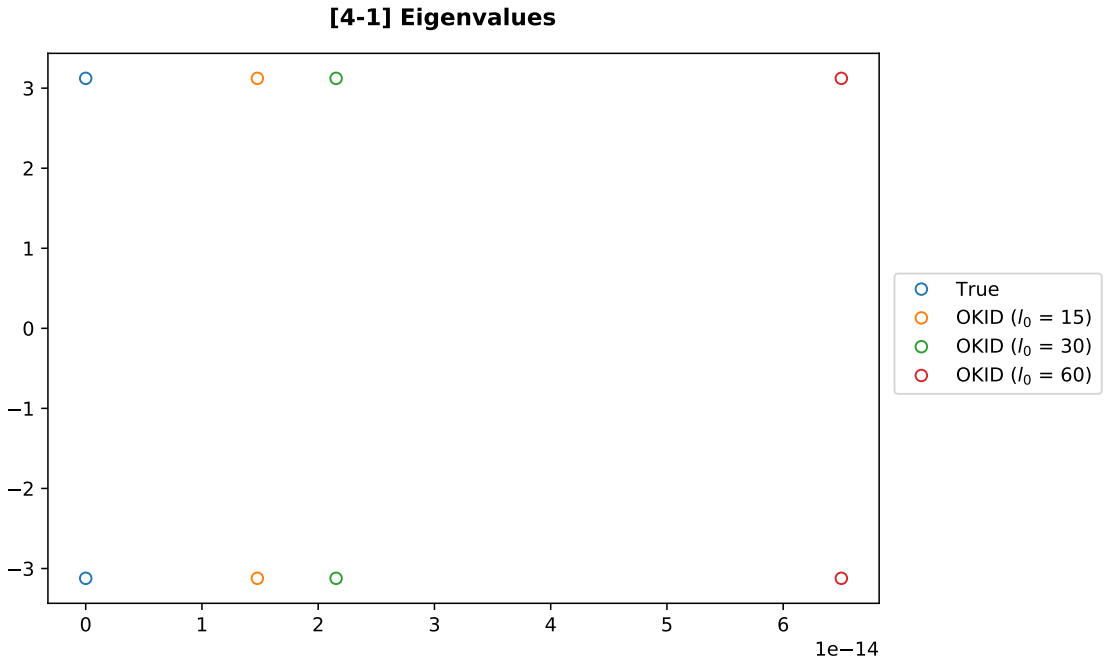
[6]:
```
RMS_train = np.zeros([len(orders), m])
RMS_test = np.zeros([len(orders), cases, m])
for j in range(len(orders)):
    RMS_train[j] = np.sqrt(np.mean((Z_okid_train[j] - Z_train[j])**2, axis = 1))
    print(f"RMS Error of sim. for system found via OKID for train data, order =␣
↪{orders[j]}: {RMS_train[j]}")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

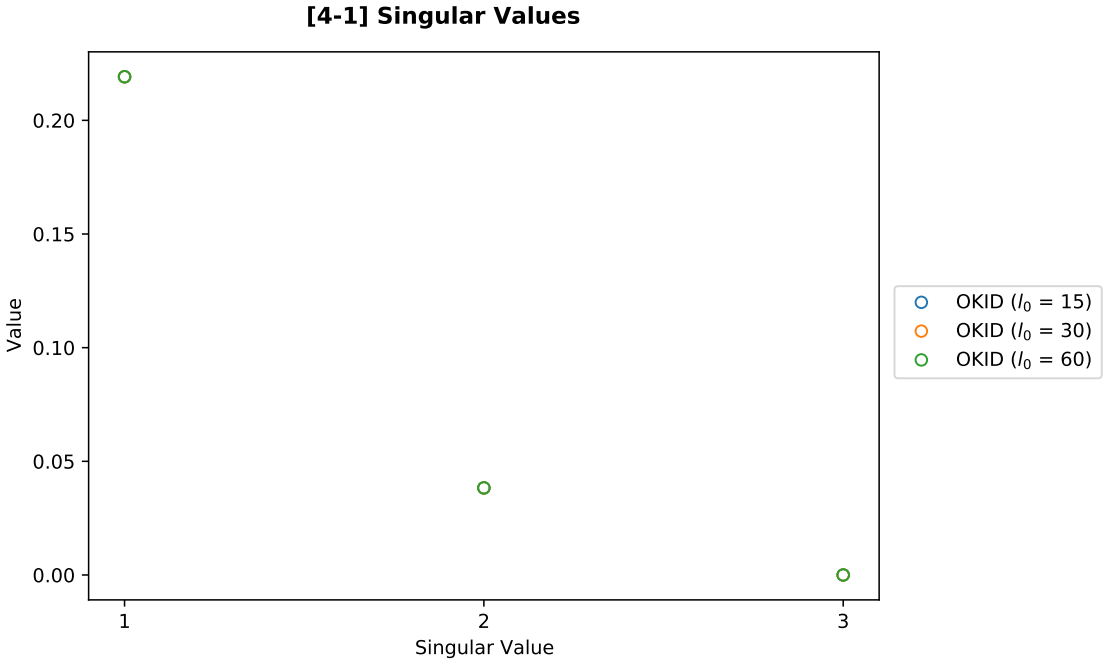
```
RMS Error of sim. for system found via OKID for train data, order = 15:
[3.11624537e-14 9.14513908e-14]
RMS Error of sim. for system found via OKID for test data, order = 15, case 0:
[9.60278756e-14 2.96279925e-13]
RMS Error of sim. for system found via OKID for test data, order = 15, case 1:
[3.11070952e-13 9.08136191e-13]
RMS Error of sim. for system found via OKID for test data, order = 15, case 2:
[3.04070728e-14 9.44537477e-14]
RMS Error of sim. for system found via OKID for train data, order = 30:
[1.66274629e-14 5.11408759e-14]
RMS Error of sim. for system found via OKID for test data, order = 30, case 0:
[5.22832852e-14 1.63558192e-13]
RMS Error of sim. for system found via OKID for test data, order = 30, case 1:
[1.61807204e-13 5.05204184e-13]
RMS Error of sim. for system found via OKID for test data, order = 30, case 2:
[1.66186607e-14 5.17150678e-14]
RMS Error of sim. for system found via OKID for train data, order = 60:
```

```
[6.57020079e-14 1.93138958e-13]
RMS Error of sim. for system found via OKID for test data, order = 60, case 0:
[2.02711891e-13 6.24666885e-13]
RMS Error of sim. for system found via OKID for test data, order = 60, case 1:
[6.53925687e-13 1.92000609e-12]
RMS Error of sim. for system found via OKID for test data, order = 60, case 2:
[6.41334551e-14 1.99187584e-13]
```

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

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

fig.legend(labels = ("True", *[f"OKID ($l_0$ = {q})" for q in orders]),
           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(orders)):
    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 ($l_0$ = {q})" for q in orders],
           bbox_to_anchor = (1, 0.5), loc = 6)
fig.savefig(figs_dir / f"midterm_{prob}_singval.pdf",
            bbox_inches = "tight")
```

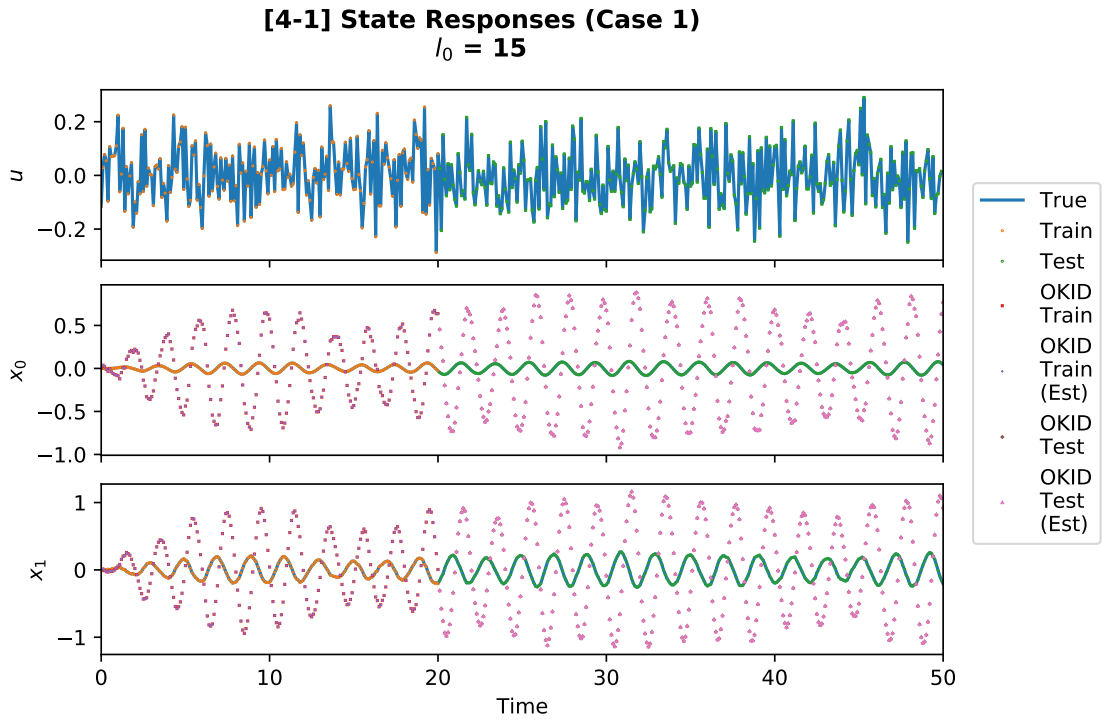


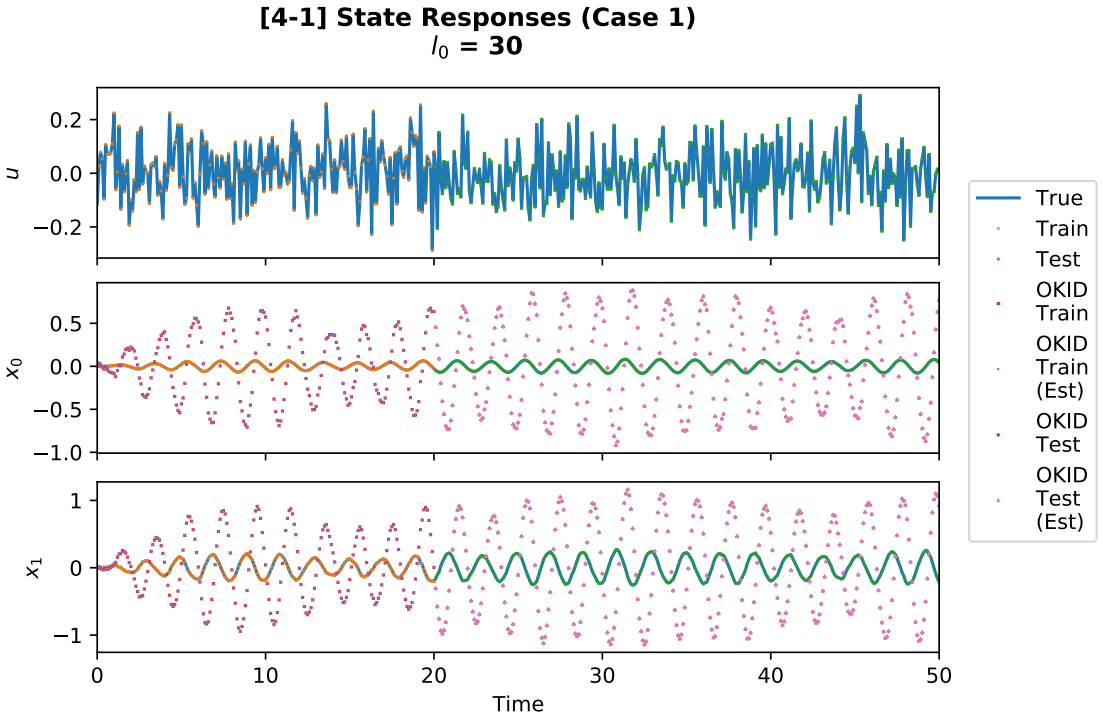



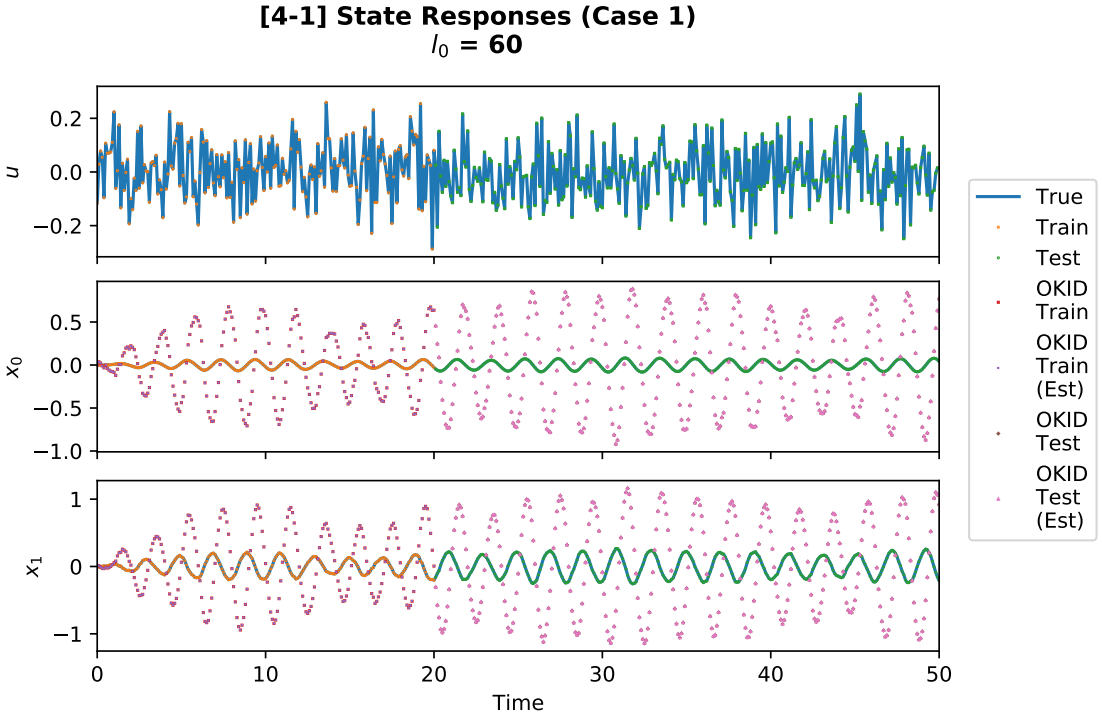

The singular values do not change as the observer order is varied.

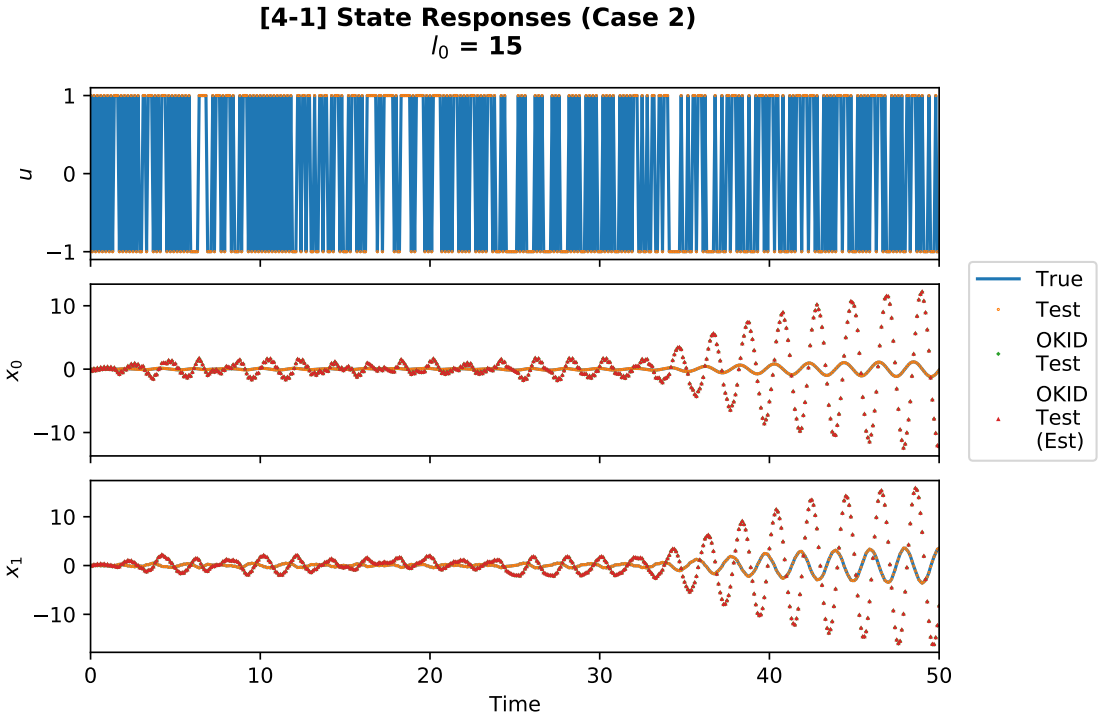
[8]:
```
# Response plots
ms = 0.5 # Marker size
for i, k in it.product(range(cases), range(len(orders))):
    fig, axs = plt.subplots(1 + n, 1,
                            sharex = "col",
                            constrained_layout = True) # type:figure.Figure
    fig.suptitle(f"[{prob}] State Responses (Case {i + 1})\n$l_0$ =␣
↪{orders[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])
```

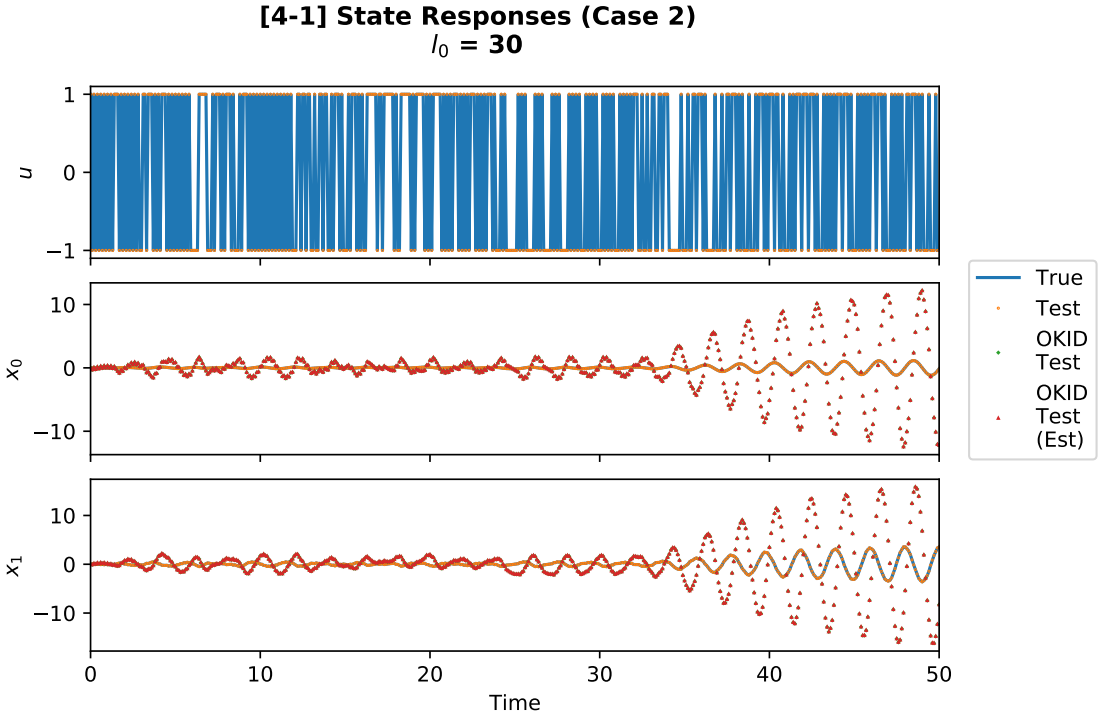
```
    for j in range(n):
        axs[j + 1].plot(t_sim, X_sim[i, j])
        axs[j + 1].plot(t_test, X_test[k, i, j],
                        "o", ms = ms, mfc = "None")
        axs[j + 1].plot(t_test, X_okid_test[k, i, j],
                        "D", ms = ms, mfc = "None")
        axs[j + 1].plot(t_test, 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}_order{k}.pdf",
            bbox_inches = "tight")
```

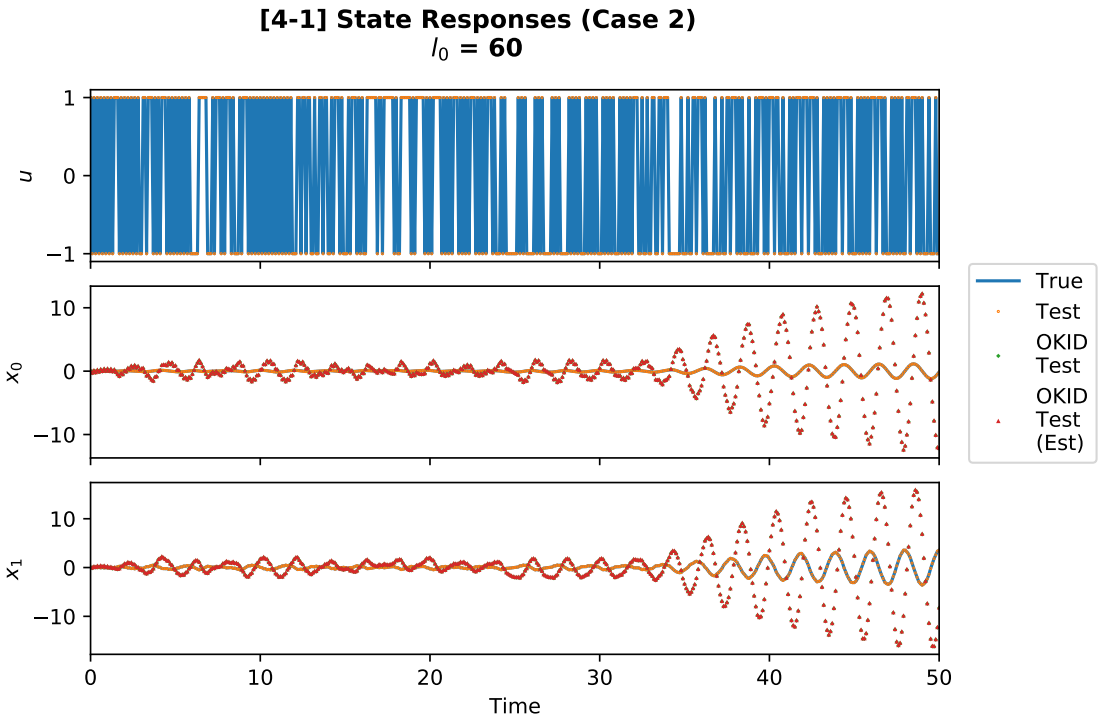


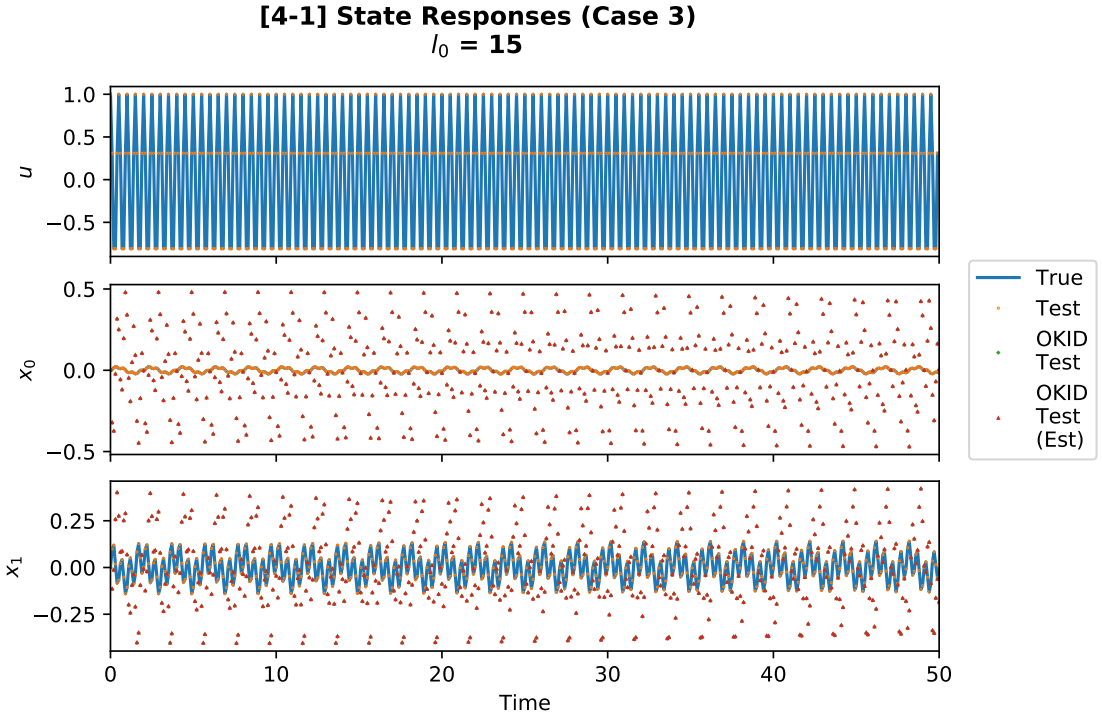

[4-1] State Responses (Case 1)
$l_0$ = 30
u
$x_0$
$x_1$
Time
True
Train
Test
OKID Train
OKID Train (Est)
OKID Test
OKID Test (Est)

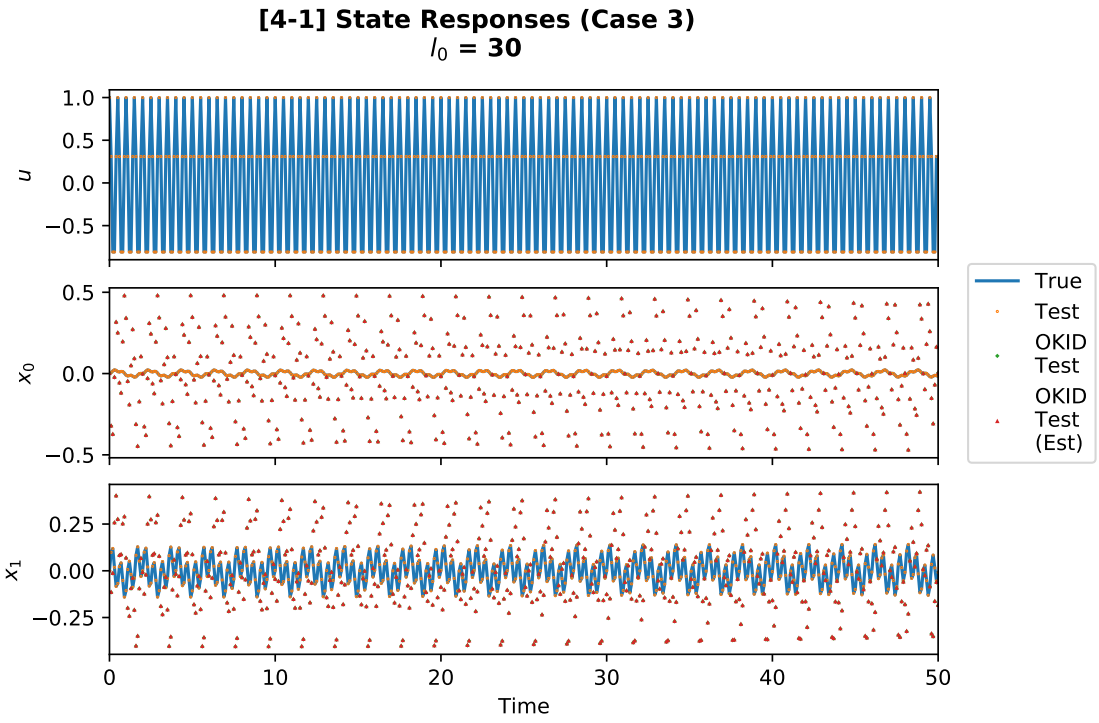

[4-1] State Responses (Case 1)
$l_0$ = 60
u
$x_0$
$x_1$
Time
True
Train
Test
OKID Train
OKID Train (Est)
OKID Test
OKID Test (Est)

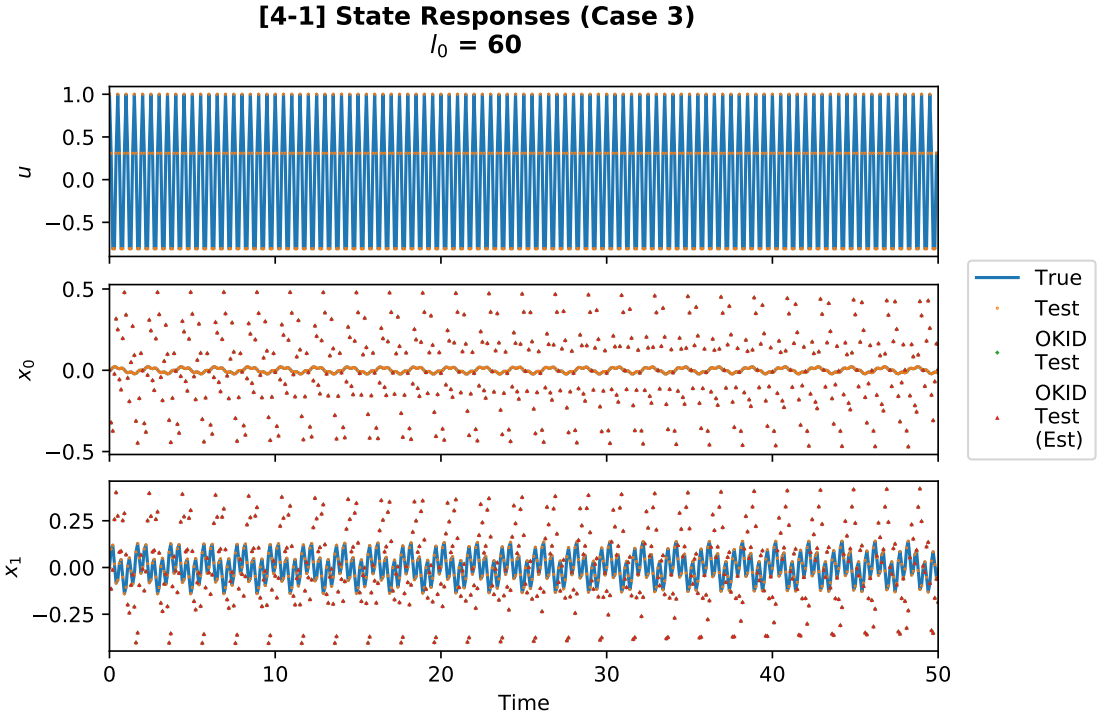

[4-1] State Responses (Case 2)
$l_0$ = 15
u
$x_0$
$x_1$
Time
True
Test
OKID Test
OKID Test (Est)


[4-1] State Responses (Case 2)
$l_0$ = 30
u
$x_0$
$x_1$
Time
True
Test
OKID Test
OKID Test (Est)


[4-1] State Responses (Case 2)
$l_0$ = 60
u
$x_0$
$x_1$
Time
True
Test
OKID Test
OKID Test (Est)


[4-1] State Responses (Case 3)
$l_0$ = 15
u
$x_0$
$x_1$
Time
True
Test
OKID Test
OKID Test (Est)

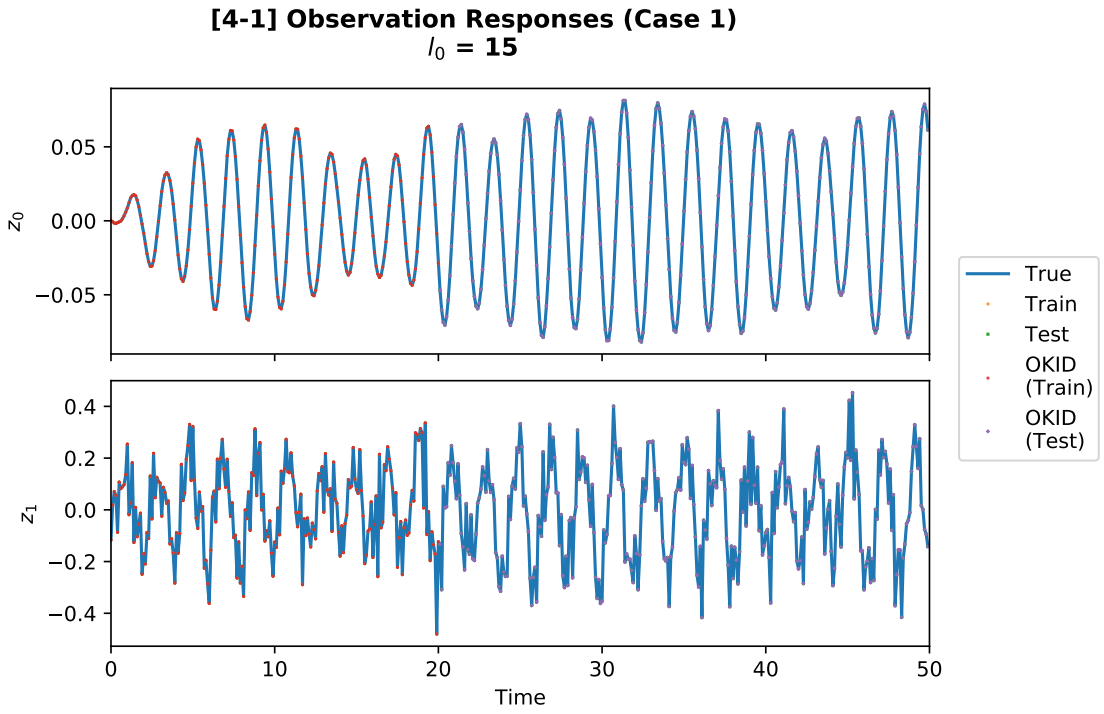

[4-1] State Responses (Case 3)
$l_0$ = 30
u
$x_0$
$x_1$
Time
True
Test
OKID Test
OKID Test (Est)

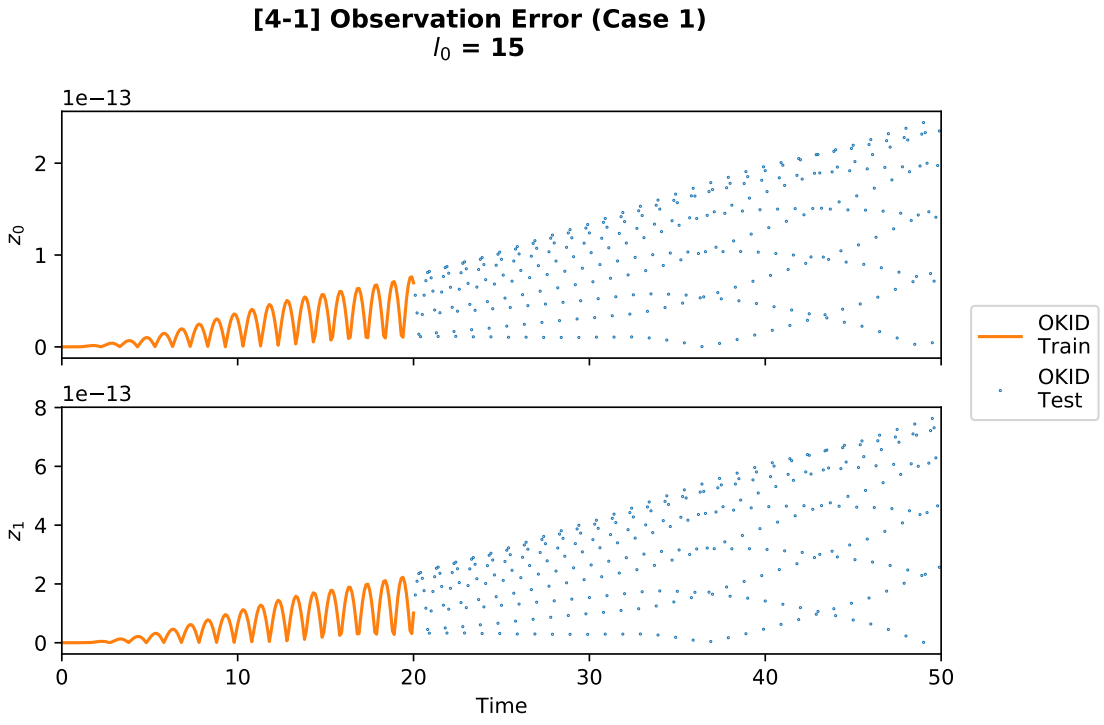

[4-1] State Responses (Case 3)
$l_0$ = 60
u
$x_0$
$x_1$
Time
True
Test
OKID Test
OKID Test (Est)

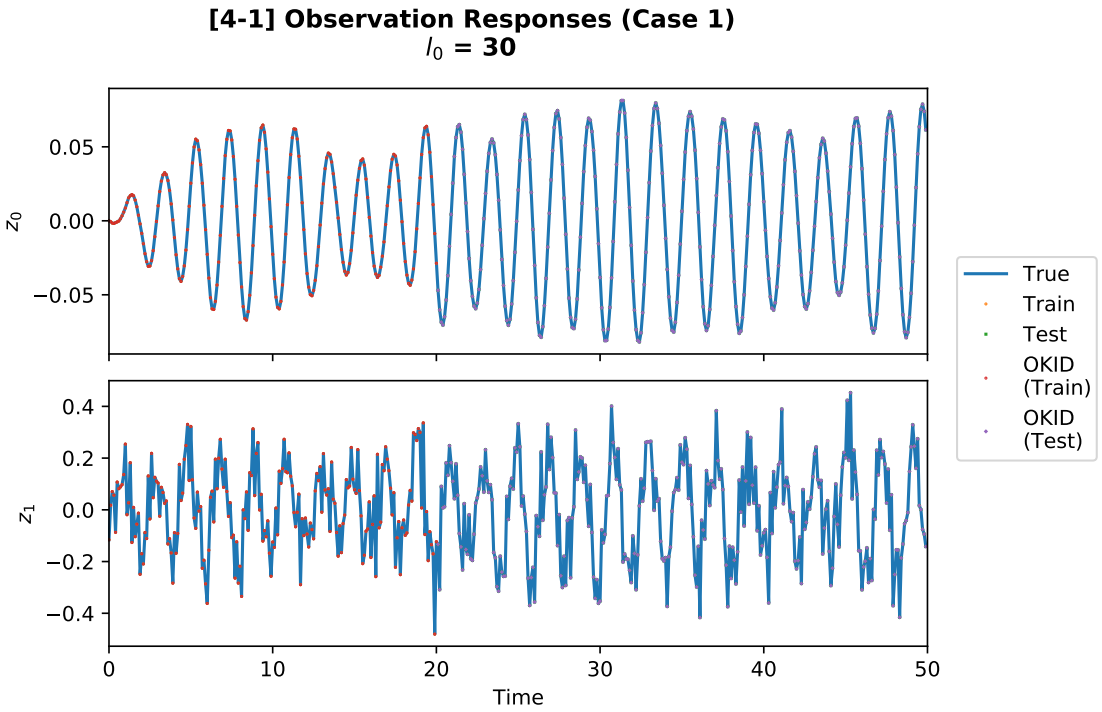
The improvement of the observer to the state estimate is negligible, as the "raw" state is already a perfect realization of the system state that reproduces the test outputs at each sample time.

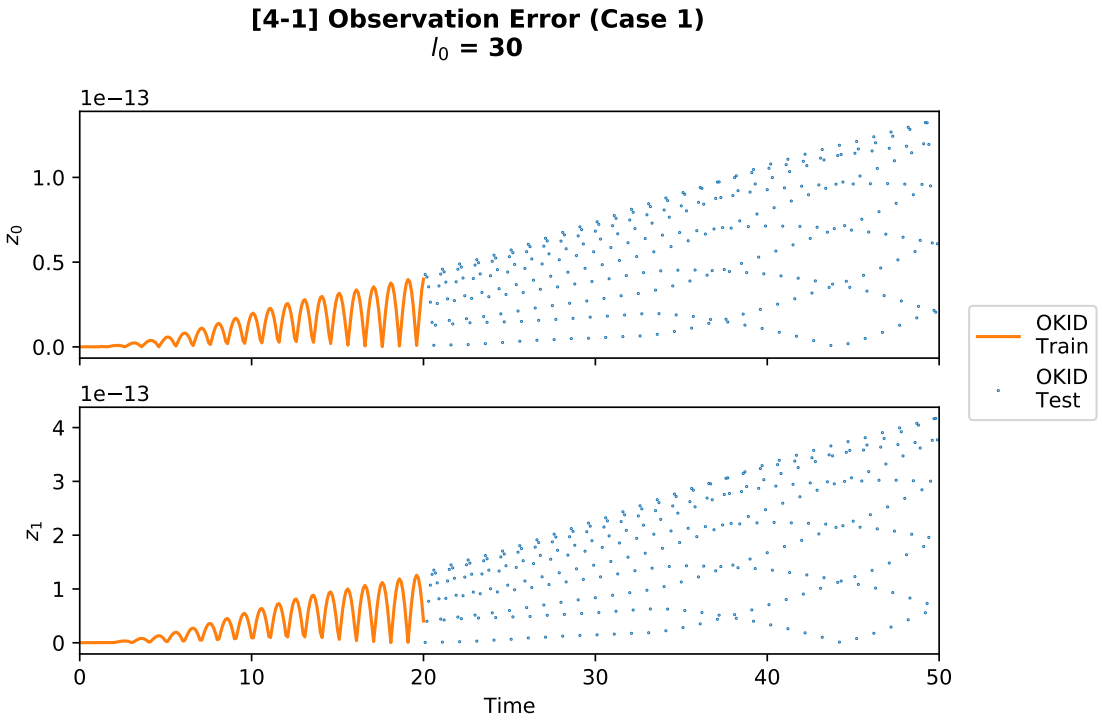
[9]:
```
# Observation plots
for i, k in it.product(range(cases), range(len(orders))):
    # Raw observations
    raw_fig, axs = plt.subplots(m, 1,
                                sharex = "col", constrained_layout = True) #␣
↪type:figure.Figure
    raw_fig.suptitle(f"[{prob}] Observation Responses (Case {i + 1})\n$l_0$ =␣
↪{orders[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}_order{k}.pdf",
                    bbox_inches = "tight")
```

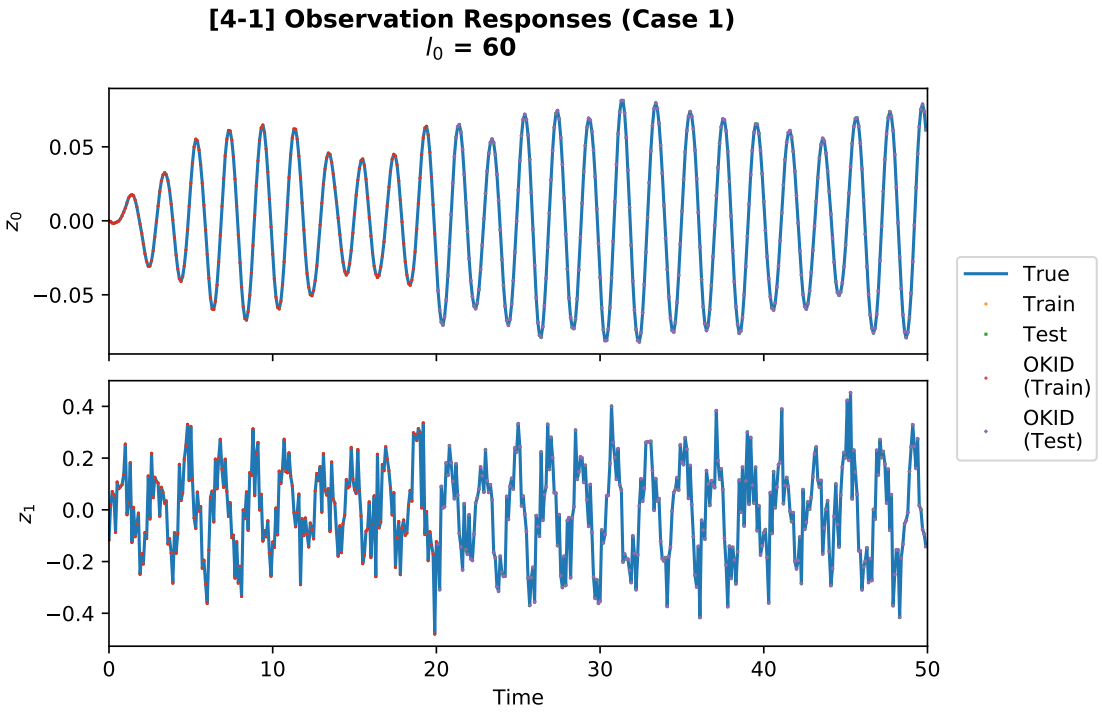
```
    # Observation error
    err_fig, axs = plt.subplots(m, 1,
                                sharex = "col", constrained_layout = True) #␣
↪type:figure.Figure
    err_fig.suptitle(f"[{prob}] Observation Error (Case {i + 1})\n$l_0$ =␣
↪{orders[k]}",
                     fontweight = "bold")
    if 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}_order{k}.
↪pdf",
                    bbox_inches = "tight")
```

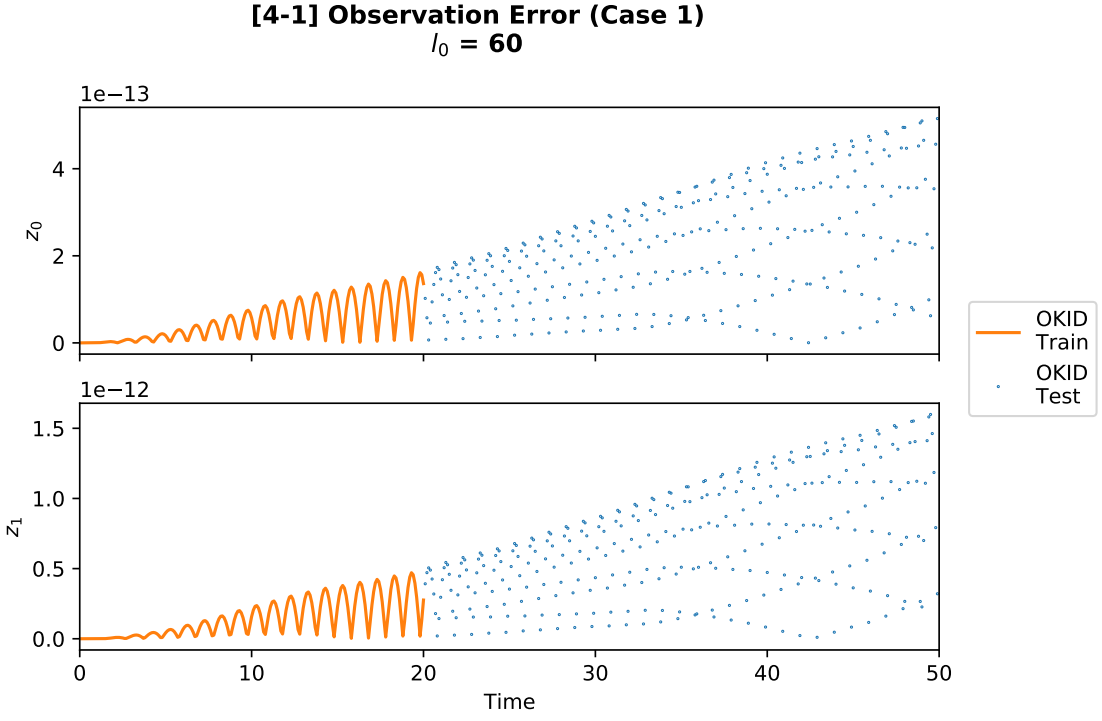

[4-1] Observation Responses (Case 1)
$l_0$ = 15
$z_0$
$z_1$
Time
True
Train
Test
OKID (Train)
OKID (Test)

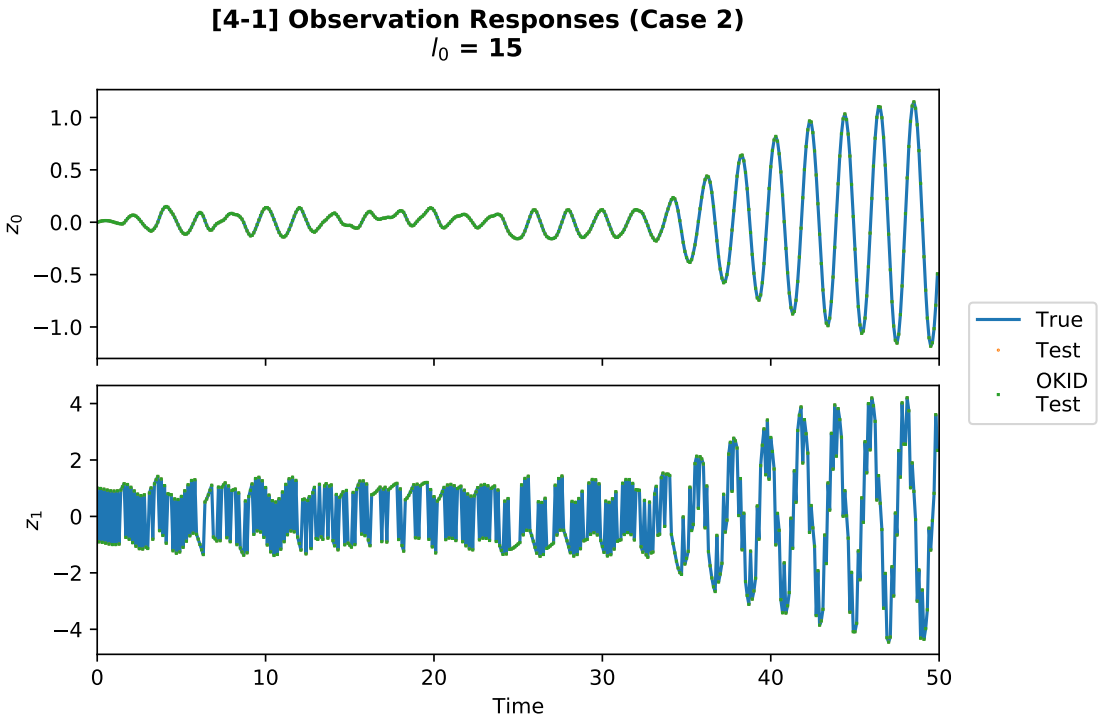

[4-1] Observation Error (Case 1)
$l_0$ = 15
1e−13
$z_0$
$z_1$
Time
OKID Train
OKID Test

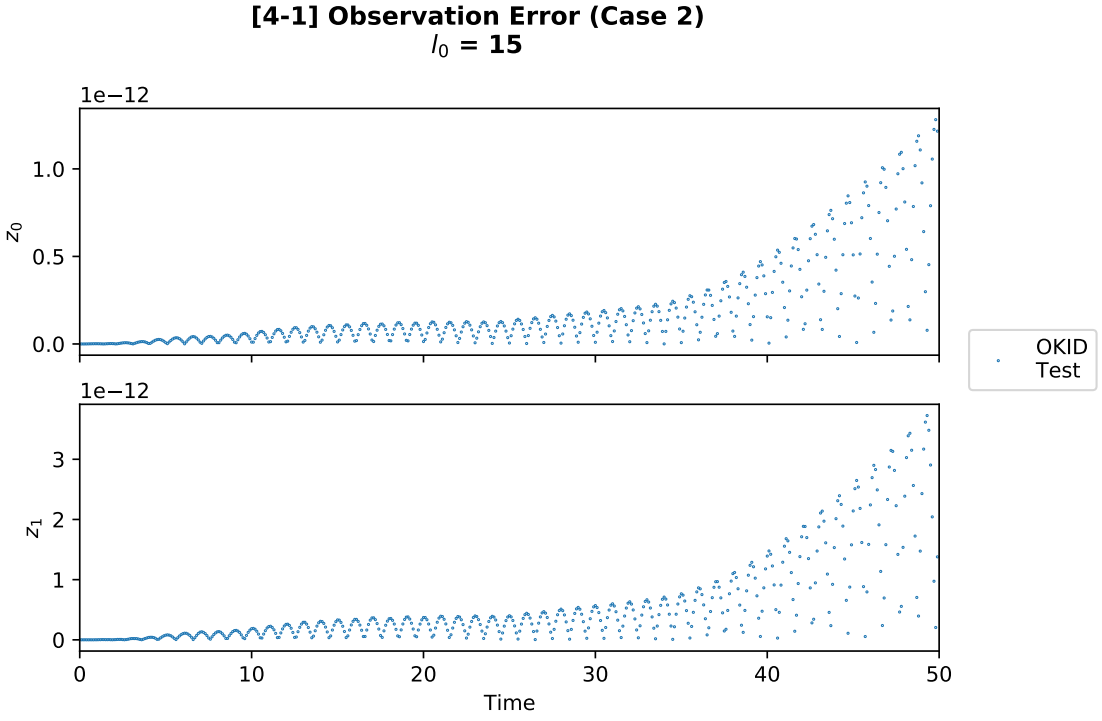

[4-1] Observation Responses (Case 1)
$l_0$ = 30
$z_0$
$z_1$
Time
True
Train
Test
OKID (Train)
OKID (Test)

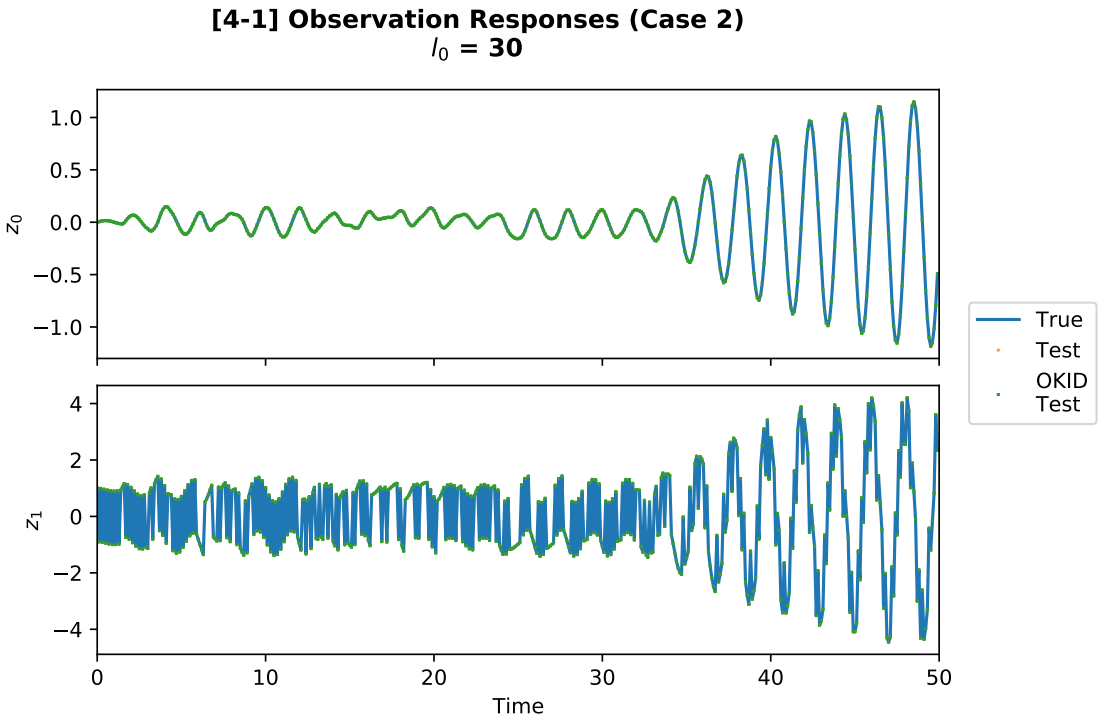

[4-1] Observation Error (Case 1)
$l_0$ = 30
1e−13
$z_0$
$z_1$
Time
OKID Train
OKID Test

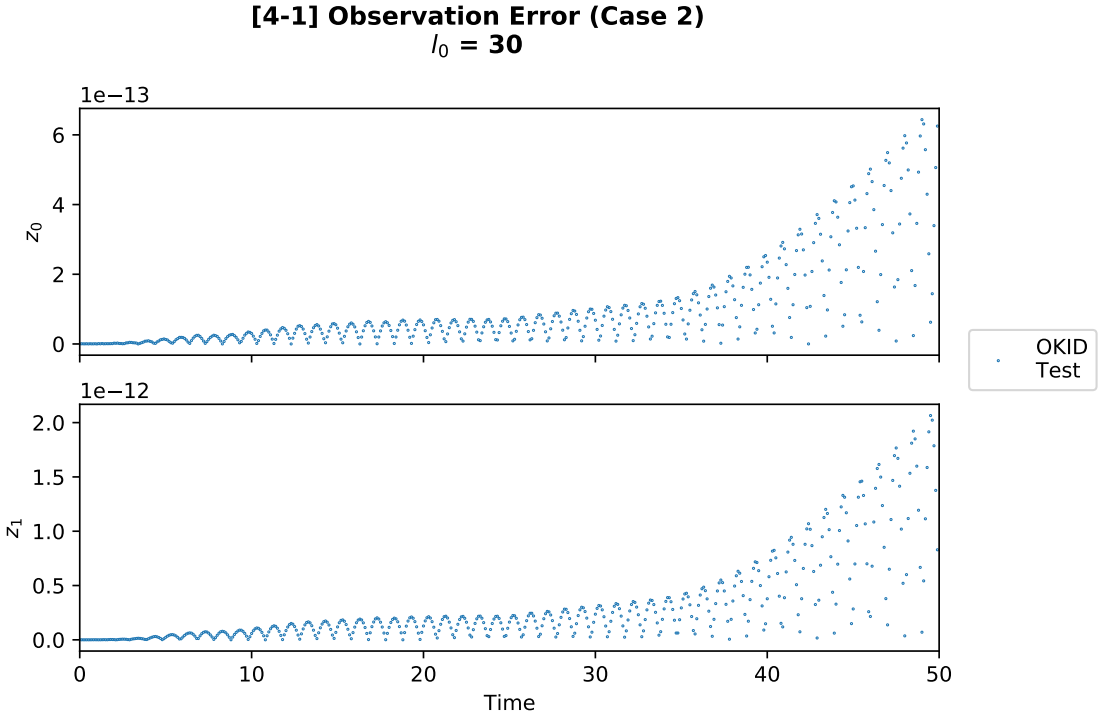

[4-1] Observation Responses (Case 1)
$l_0$ = 60
0.05
0.00
−0.05
$z_0$
0.4
0.2
0.0
−0.2
−0.4
$z_1$
0
10
20
30
40
50
Time
True
Train
Test
OKID (Train)
OKID (Test)

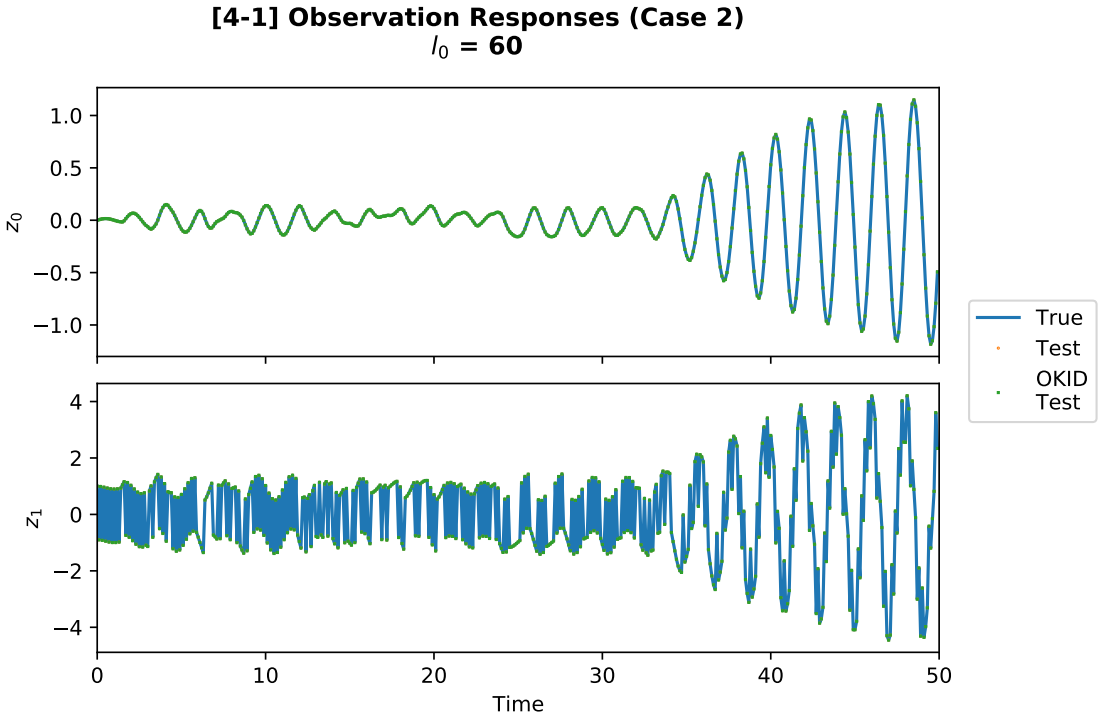

[4-1] Observation Error (Case 1)
$l_0$ = 60
1e−13
4
2
0
$z_0$
1e−12
1.5
1.0
0.5
0.0
$z_1$
0
10
20
30
40
50
Time
OKID Train
OKID Test

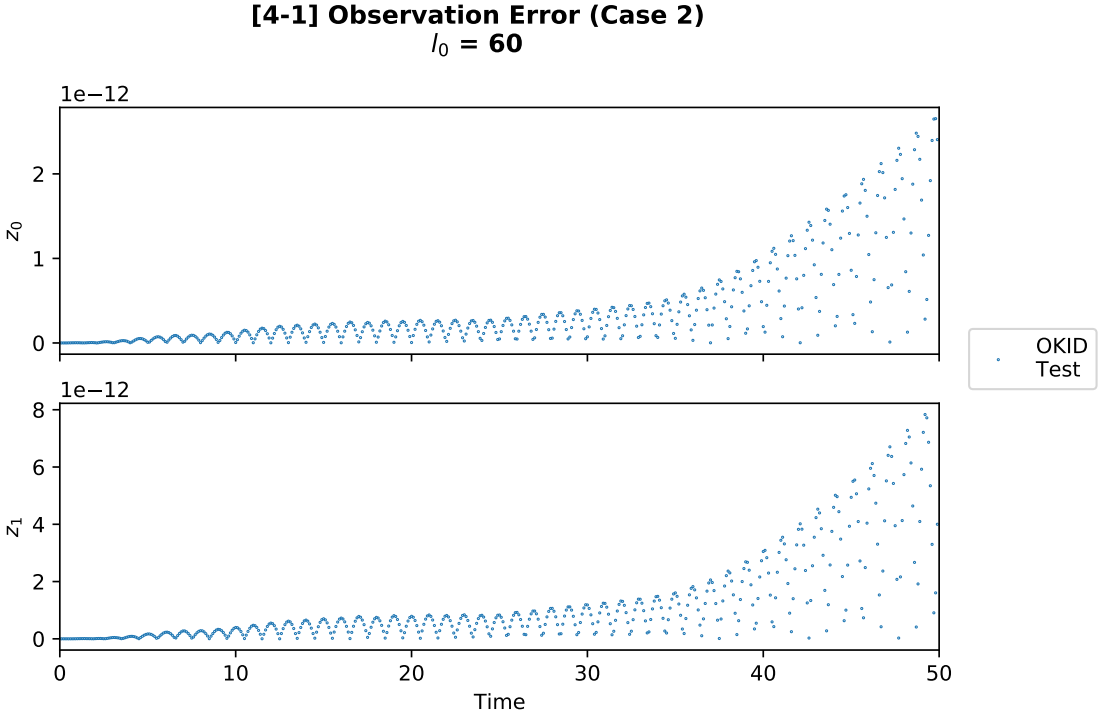

[4-1] Observation Responses (Case 2)
$l_0$ = 15
$z_0$
$z_1$
Time
True
Test
OKID
Test

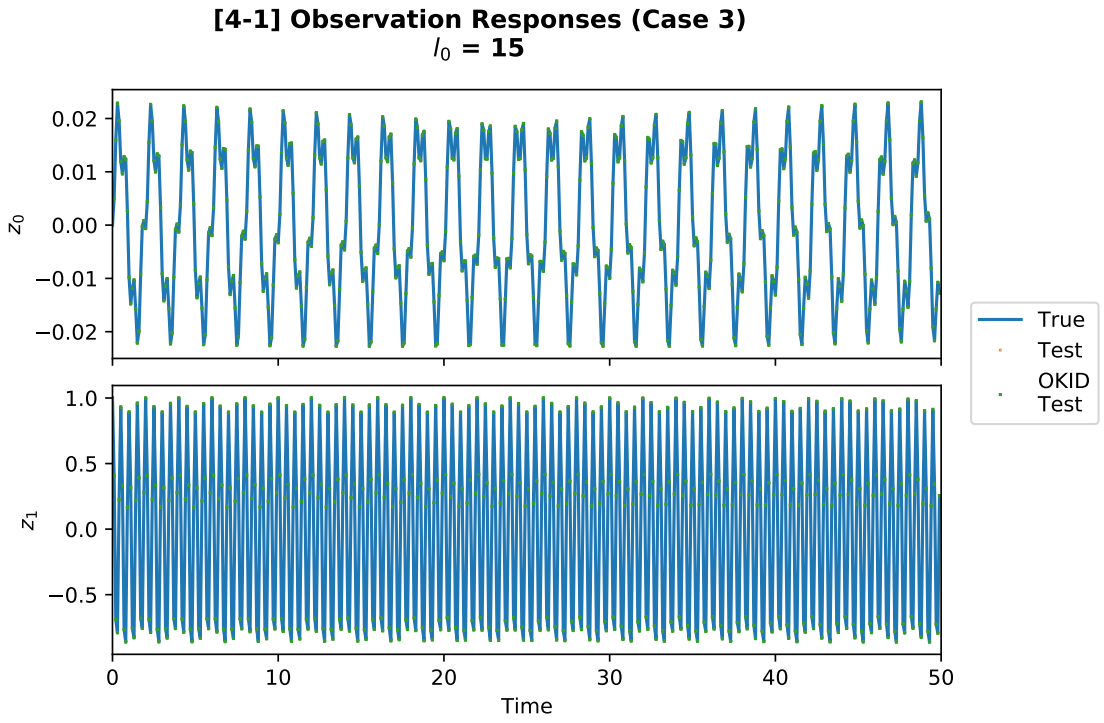

[4-1] Observation Error (Case 2)
$l_0$ = 15
1e−12
1e−12
$z_0$
$z_1$
Time
OKID
Test

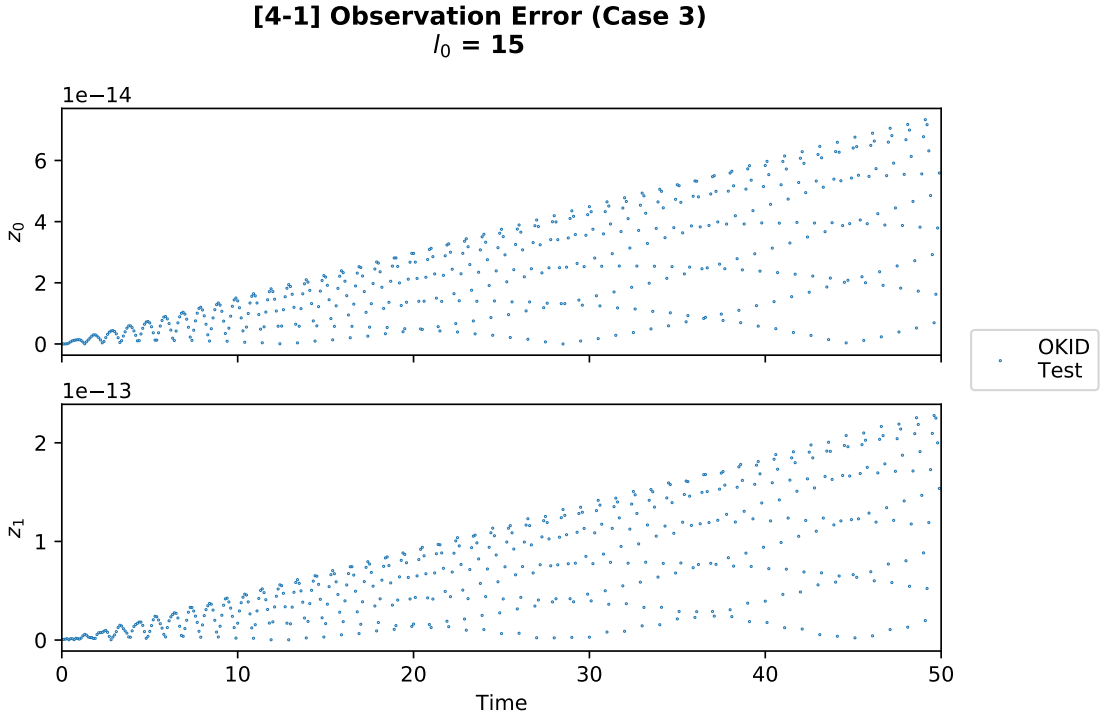

[4-1] Observation Responses (Case 2)
$l_0$ = 30
$z_0$
$z_1$
Time
True
Test
OKID
Test

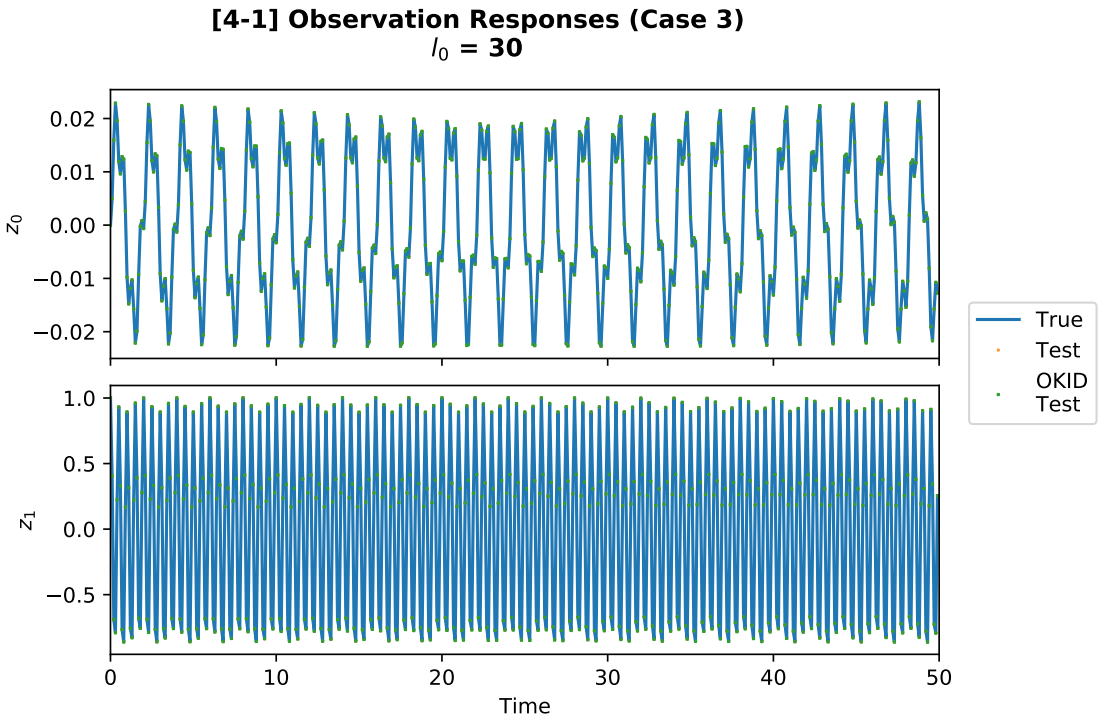

[4-1] Observation Error (Case 2)
$l_0$ = 30
1e−13
1e−12
$z_0$
$z_1$
Time
OKID
Test

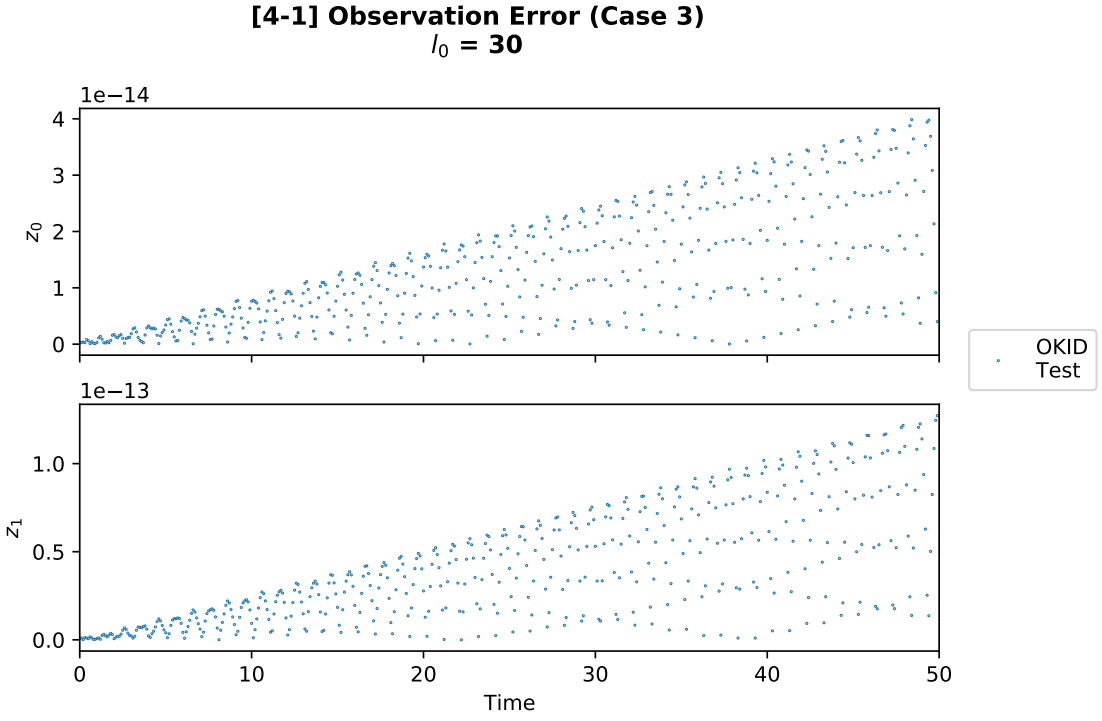

[4-1] Observation Responses (Case 2)
$l_0$ = 60
$z_0$
$z_1$
Time
True
Test
OKID
Test

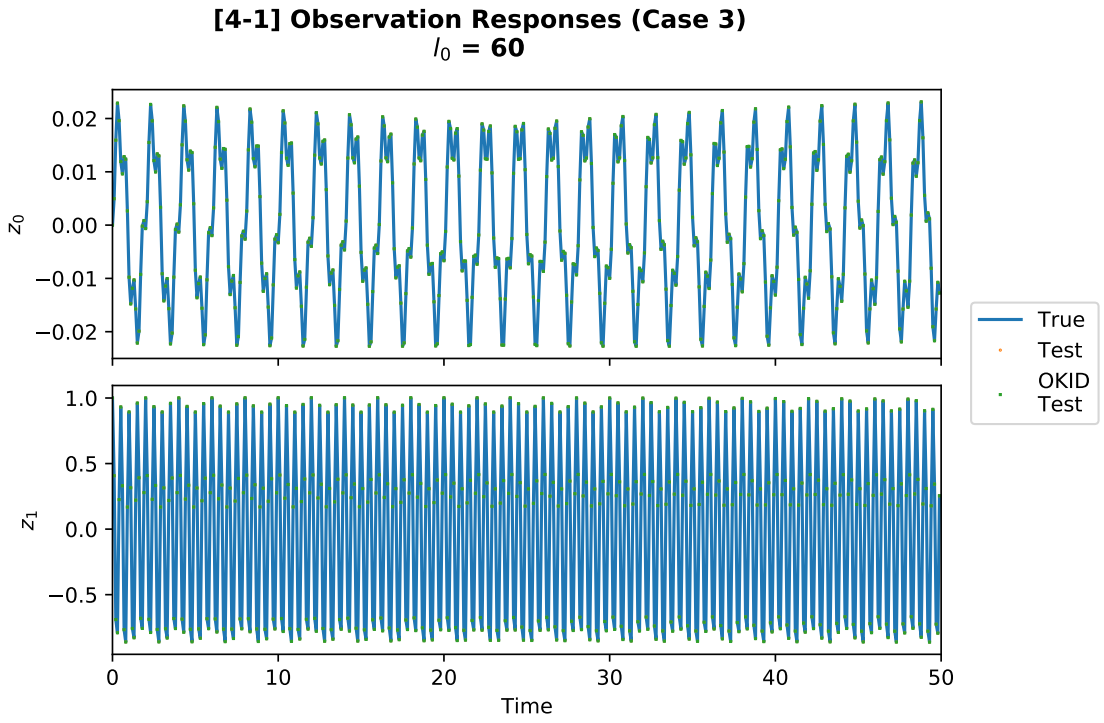

[4-1] Observation Error (Case 2)
$l_0$ = 60
1e−12
1e−12
$z_0$
$z_1$
Time
OKID
Test

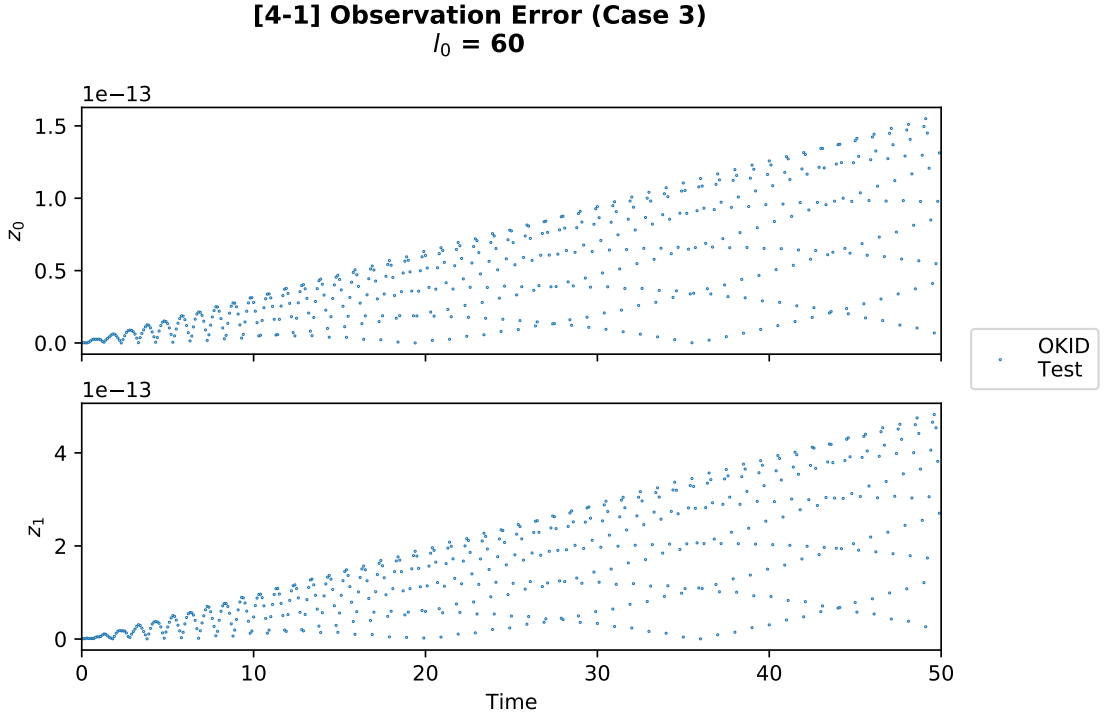

[4-1] Observation Responses (Case 3)
$l_0$ = 15
$z_0$
$z_1$
Time
True
Test
OKID
Test


[4-1] Observation Error (Case 3)
$l_0$ = 15
1e−14
1e−13
$z_0$
$z_1$
Time
OKID
Test


[4-1] Observation Responses (Case 3)
$l_0$ = 30
$z_0$
$z_1$
Time
True
Test
OKID
Test


[4-1] Observation Error (Case 3)
$l_0$ = 30
1e−14
1e−13
$z_0$
$z_1$
Time
OKID
Test


[4-1] Observation Responses (Case 3)
$l_0$ = 60
$z_0$
$z_1$
Time
True
Test
OKID
Test


[4-1] Observation Error (Case 3)
$l_0$ = 60
1e−13
1e−13
$z_0$
$z_1$
Time
OKID
Test

The estimation is flawless regardless of the observer order chosen. In this case where there is no noise, we conclude that decreasing or increasing the observer order, as long as it falls within the necessary boundaries, does not have any practical negative impact on the high accuracy of the identification of this simple system.

# 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
```

```
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]:
```
# 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]:
```
# 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.

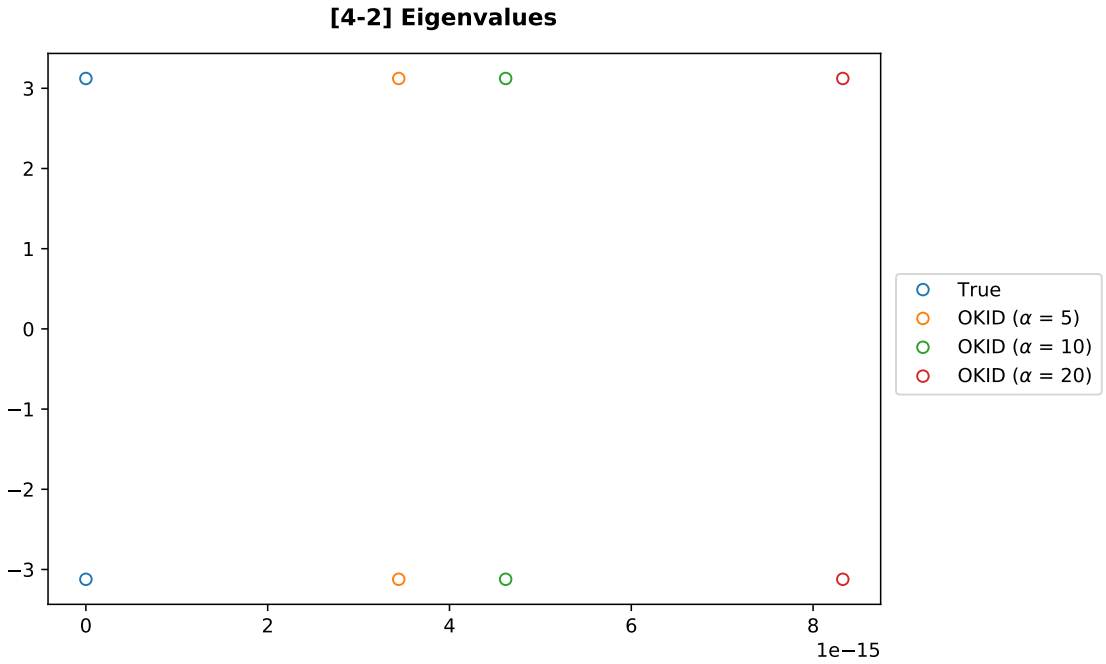
[6]:
```
RMS_train = np.zeros([len(alphas), m])
RMS_test = np.zeros([len(alphas), cases, m])
for j in range(len(alphas)):
    RMS_train[j] = np.sqrt(np.mean((Z_okid_train[j] - Z_train[j])**2, axis = 1))
    print(f"RMS Error of sim. for system found via OKID for train data, alpha =␣
↪{alphas[j]}: {RMS_train[j]}")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

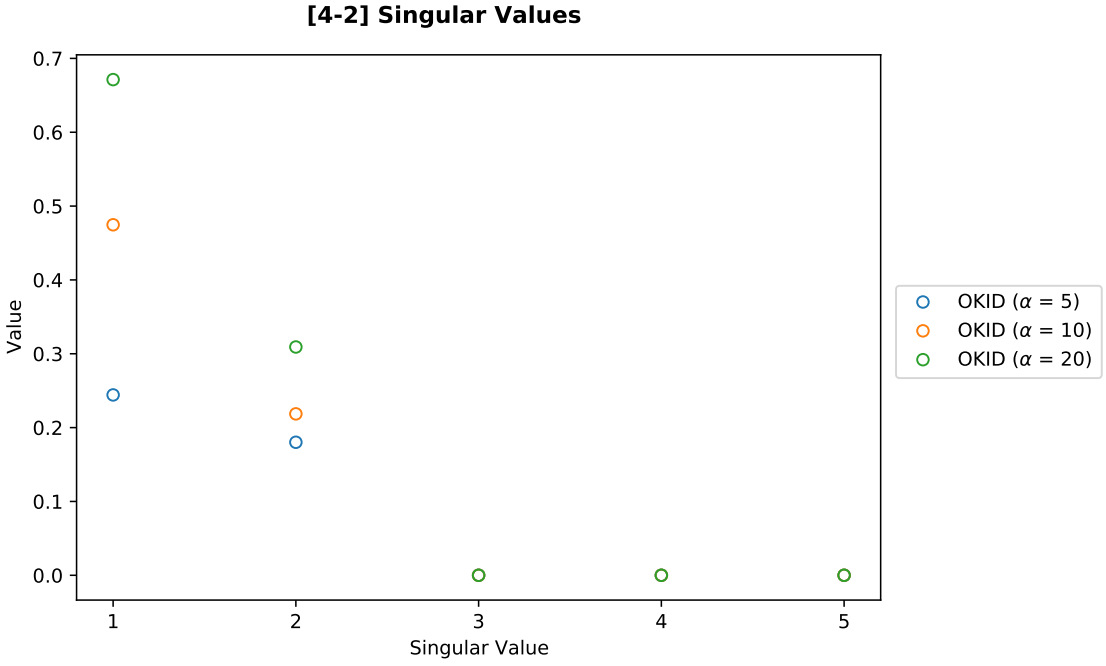
```
RMS Error of sim. for system found via OKID for train data, alpha = 5:
[1.40514174e-14 4.23549696e-14]
RMS Error of sim. for system found via OKID for test data, alpha = 5, case 0:
[4.39729475e-14 1.37597663e-13]
RMS Error of sim. for system found via OKID for test data, alpha = 5, case 1:
[1.3813405e-13 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")
```



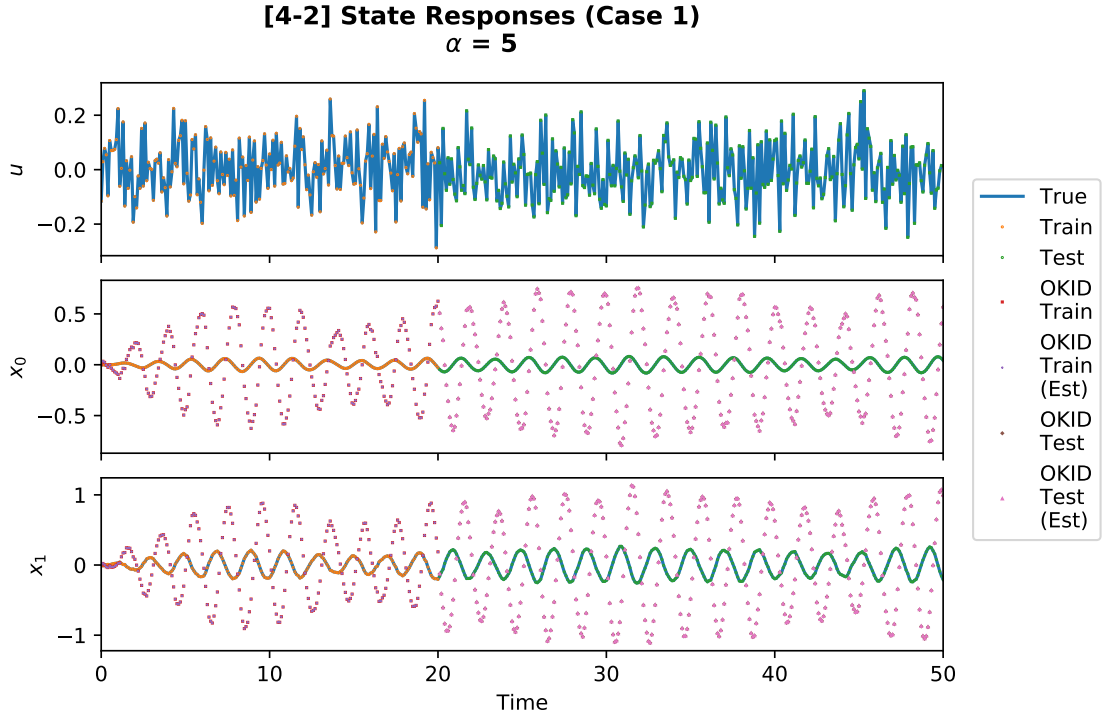



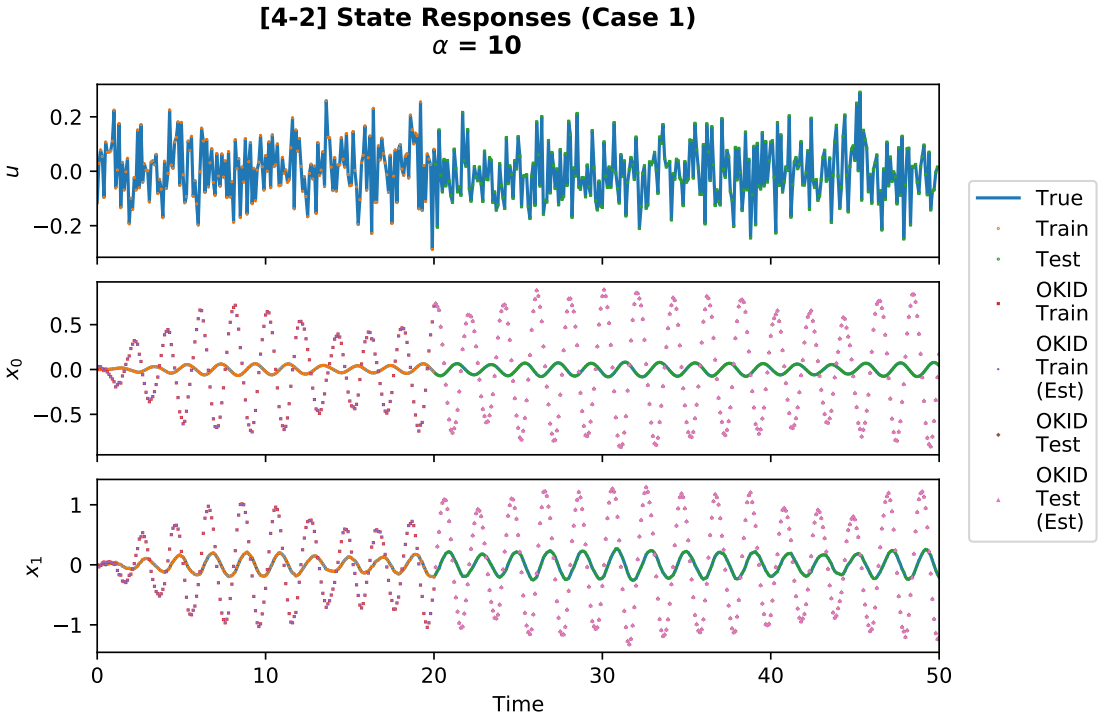

The main impact of the Hankel height is on the singular values. Although in this case the realization wasn't affected, increasing $\alpha$ led the singular values to become more disjointed, allowing us to more easily identify the order of the underlying system. As seen in the plot above, the difference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

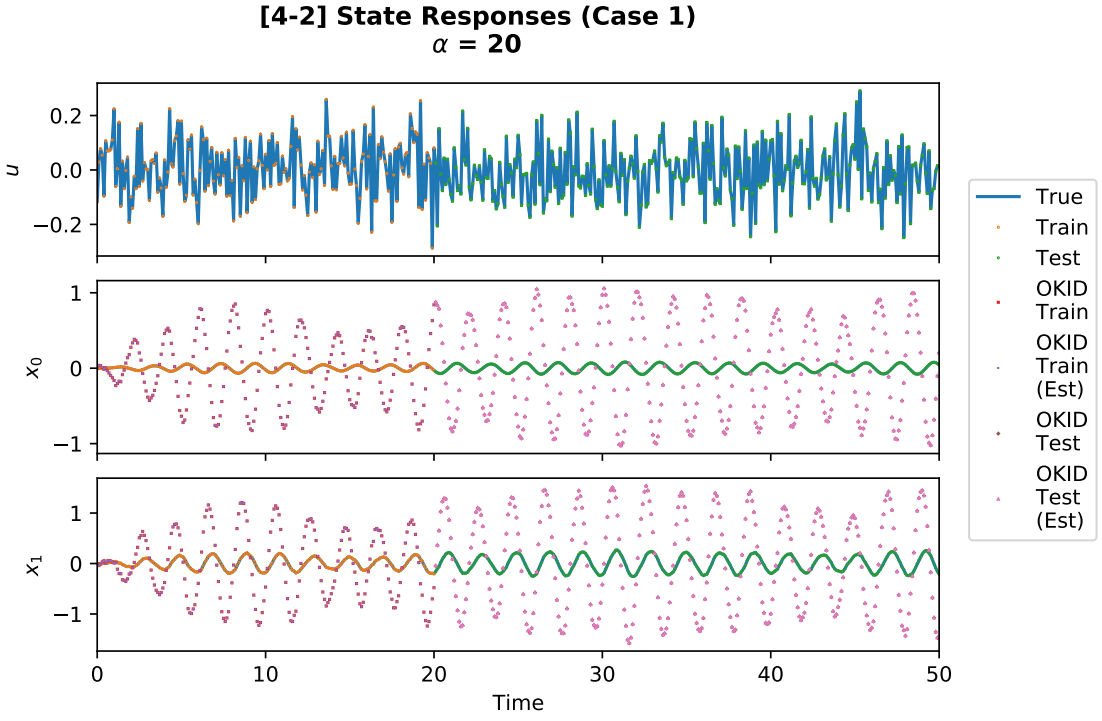
2nd and 3rd singular values slighly increased for higher $\alpha$.

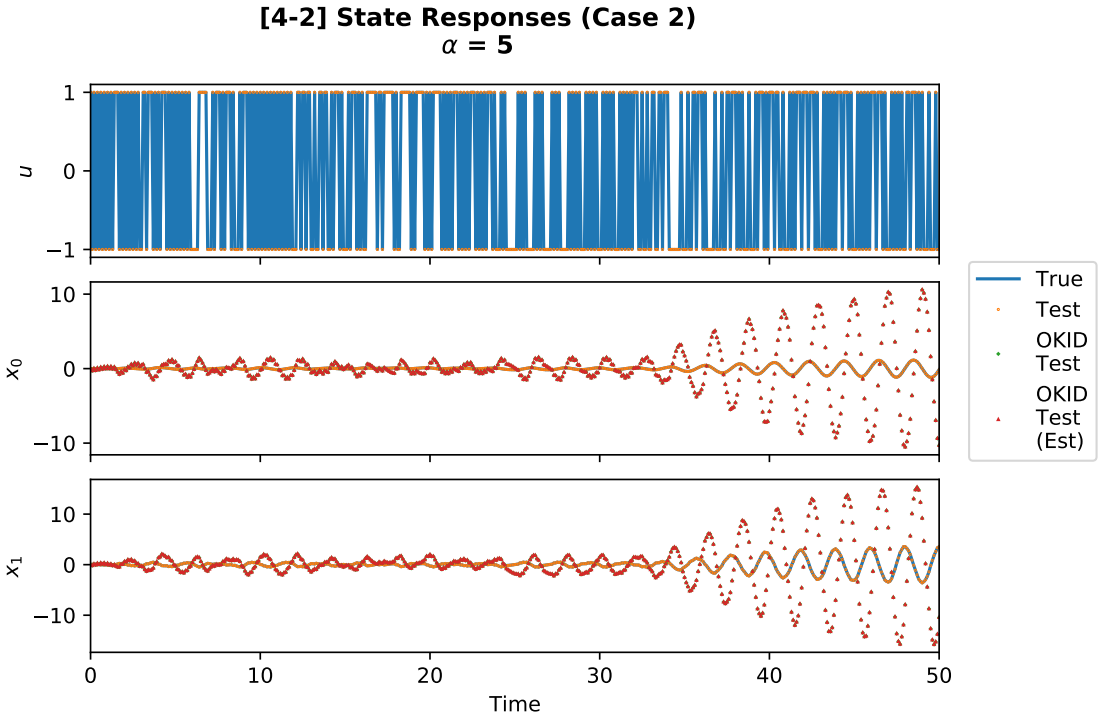
[8]:
```
# Response plots
ms = 0.5 # Marker size
for i, k in it.product(range(cases), range(len(alphas))):
    fig, axs = plt.subplots(1 + n, 1,
                            sharex = "col",
                            constrained_layout = True) # type:figure.Figure
    fig.suptitle(f"[{prob}] State Responses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

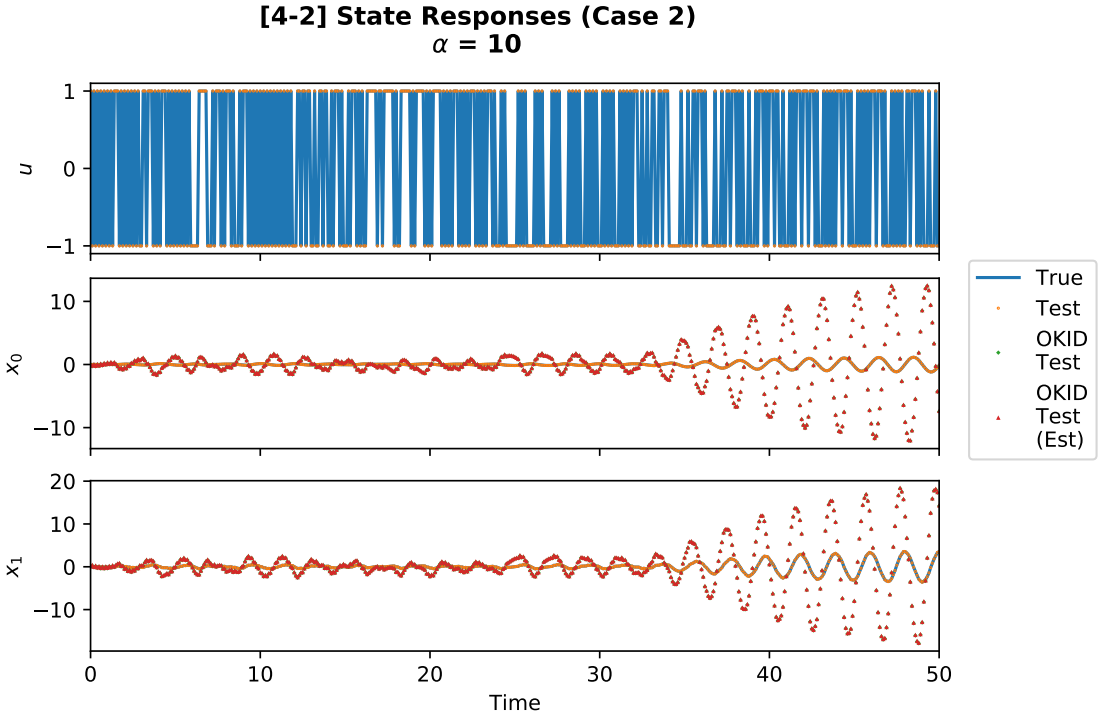
```
                    "o", ms = ms, mfc = "None")
        plt.setp(axs[0], ylabel = f"$u$", xlim = [0, t_max])

        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")
```

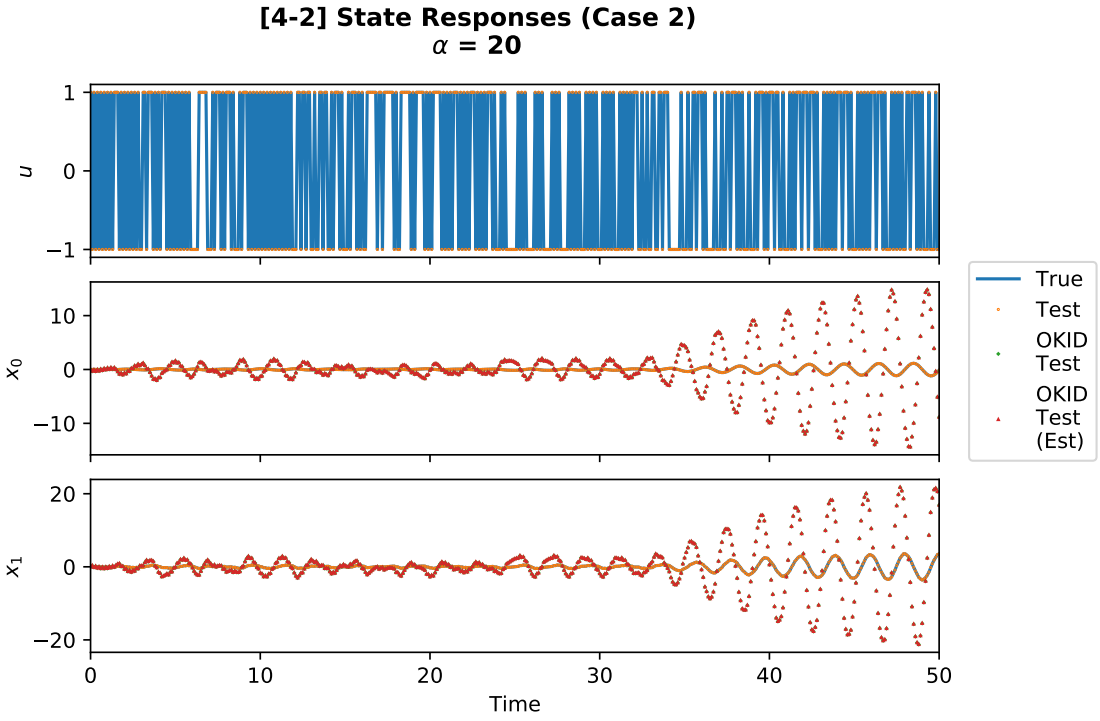


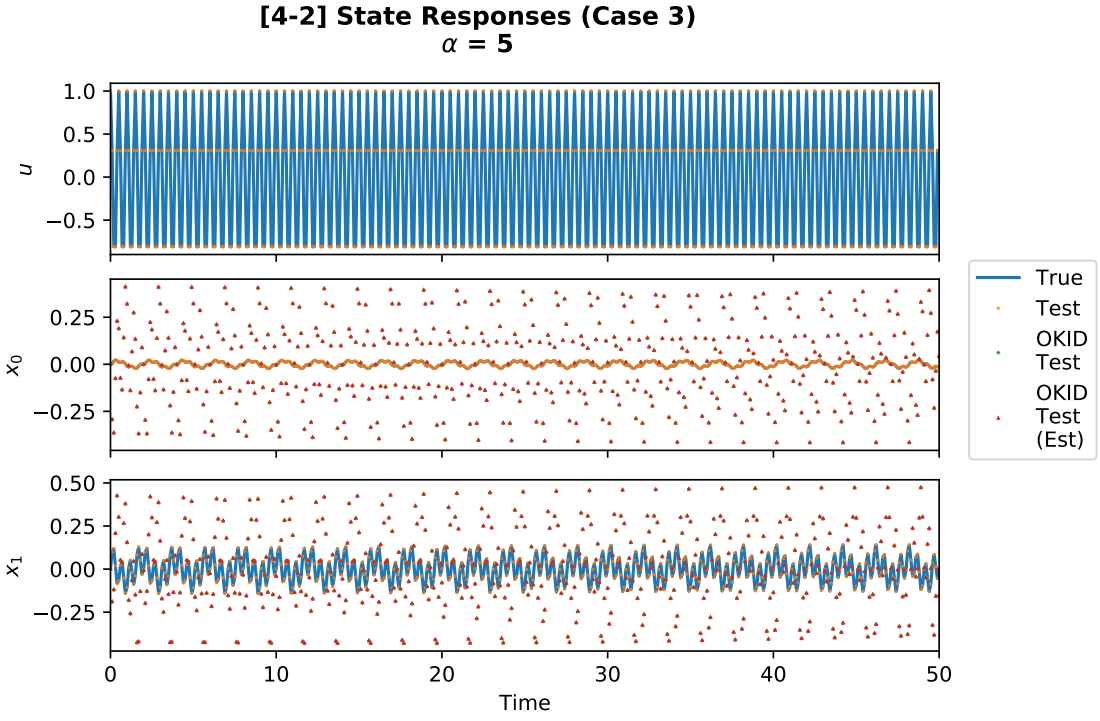

[4-2] State Responses (Case 1)
α = 10
u
x0
x1
Time
True
Train
Test
OKID Train
OKID Train (Est)
OKID Test
OKID Test (Est)

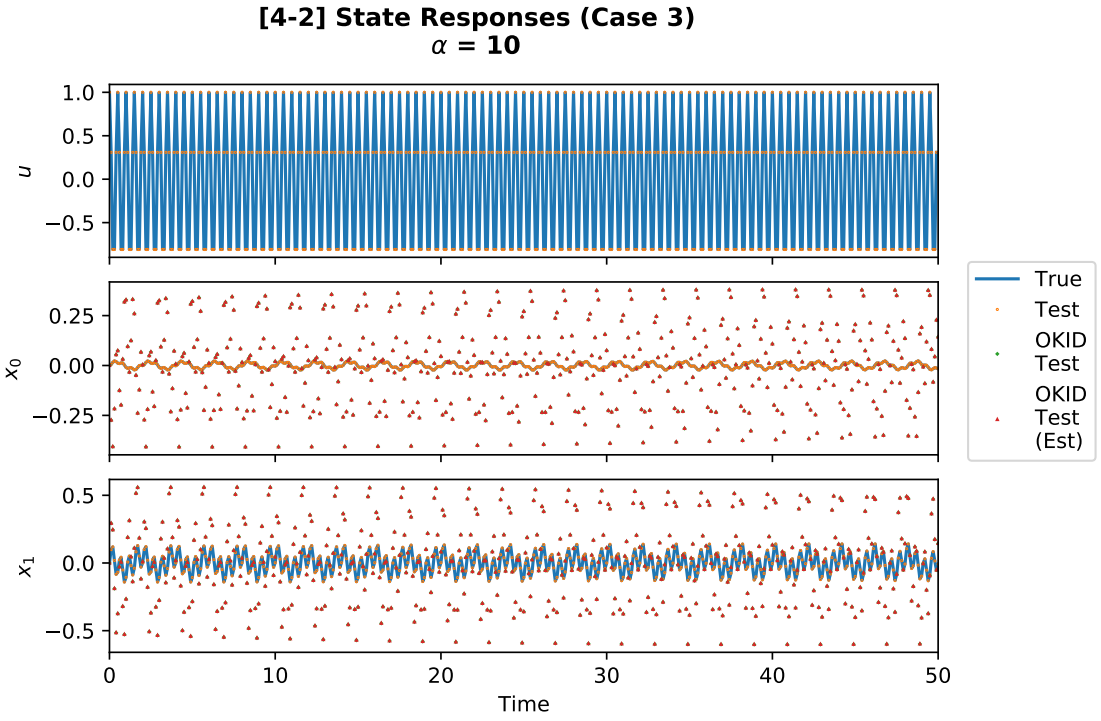

[4-2] State Responses (Case 1)
α = 20
u
x0
x1
Time
True
Train
Test
OKID Train
OKID Train (Est)
OKID Test
OKID Test (Est)

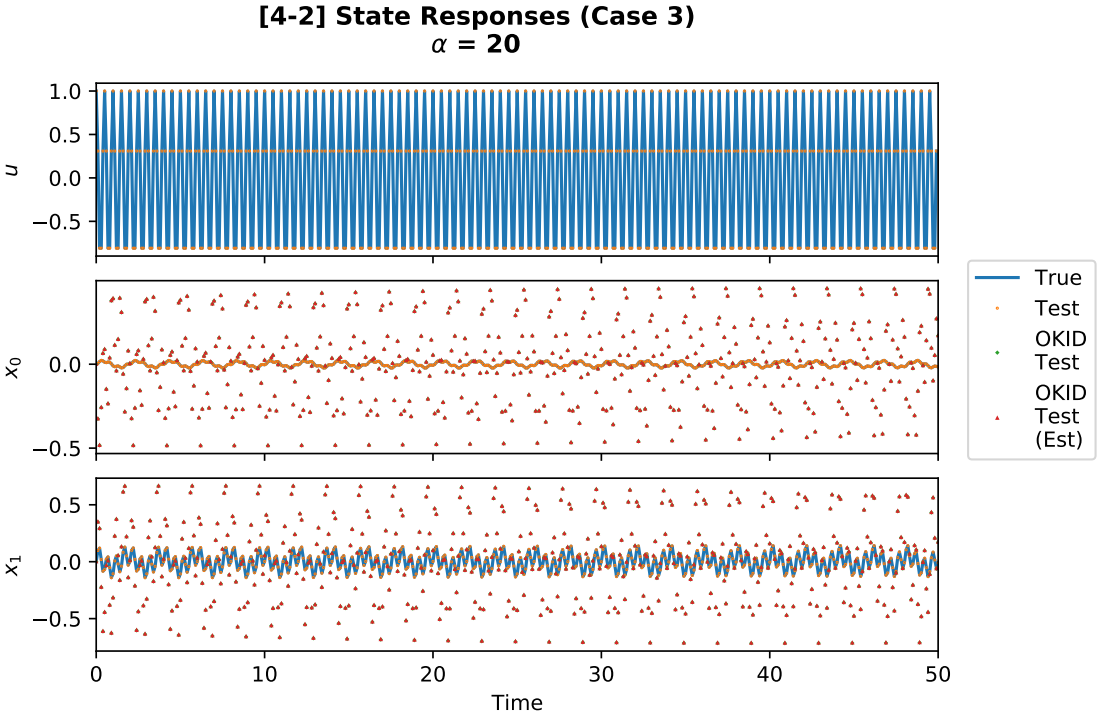

[4-2] State Responses (Case 2)
α = 5
u
x0
x1
Time
True
Test
OKID Test
OKID Test (Est)


[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)

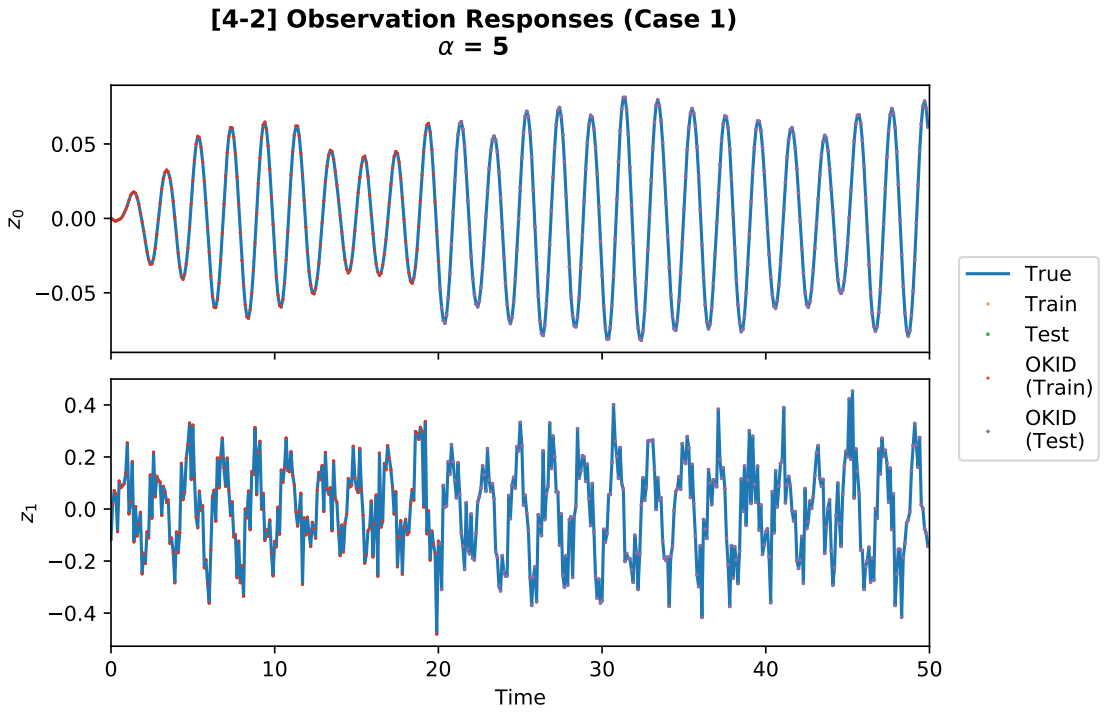

[4-2] State Responses (Case 3)
α = 10
u
x0
x1
Time
True
Test
OKID Test
OKID Test (Est)

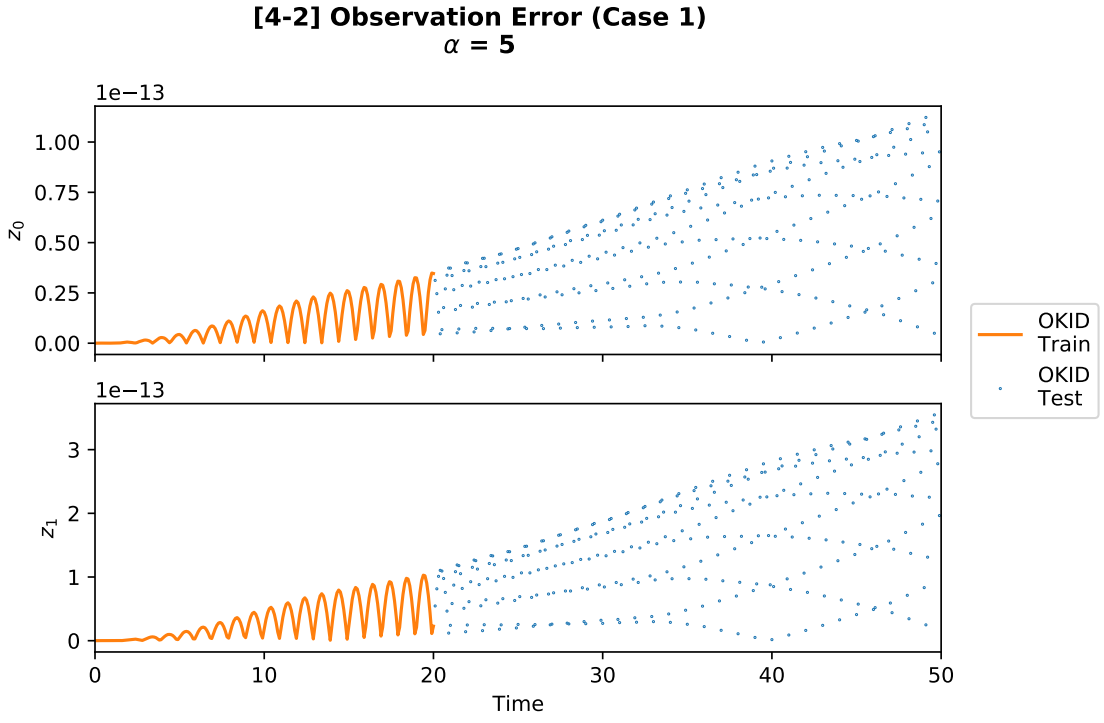

[4-2] State Responses (Case 3)
α = 20
u
x0
x1
Time
True
Test
OKID Test
OKID Test (Est)

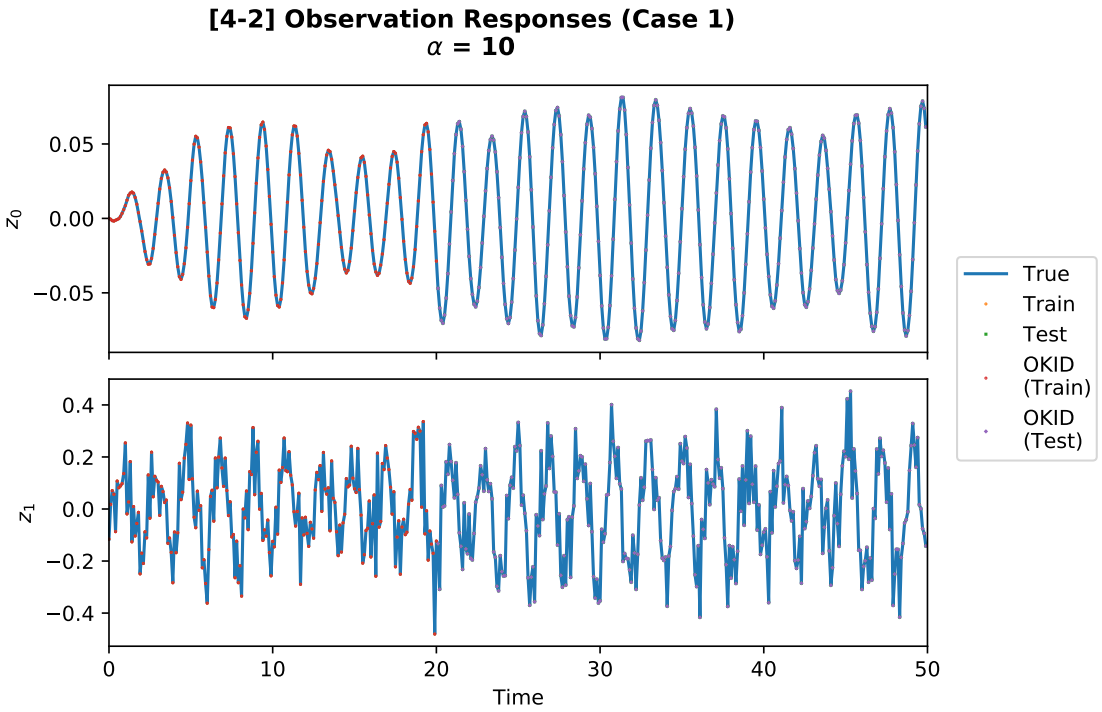
The choice of Hankel height did not affect the (already highly accurate) state estimation in this case.

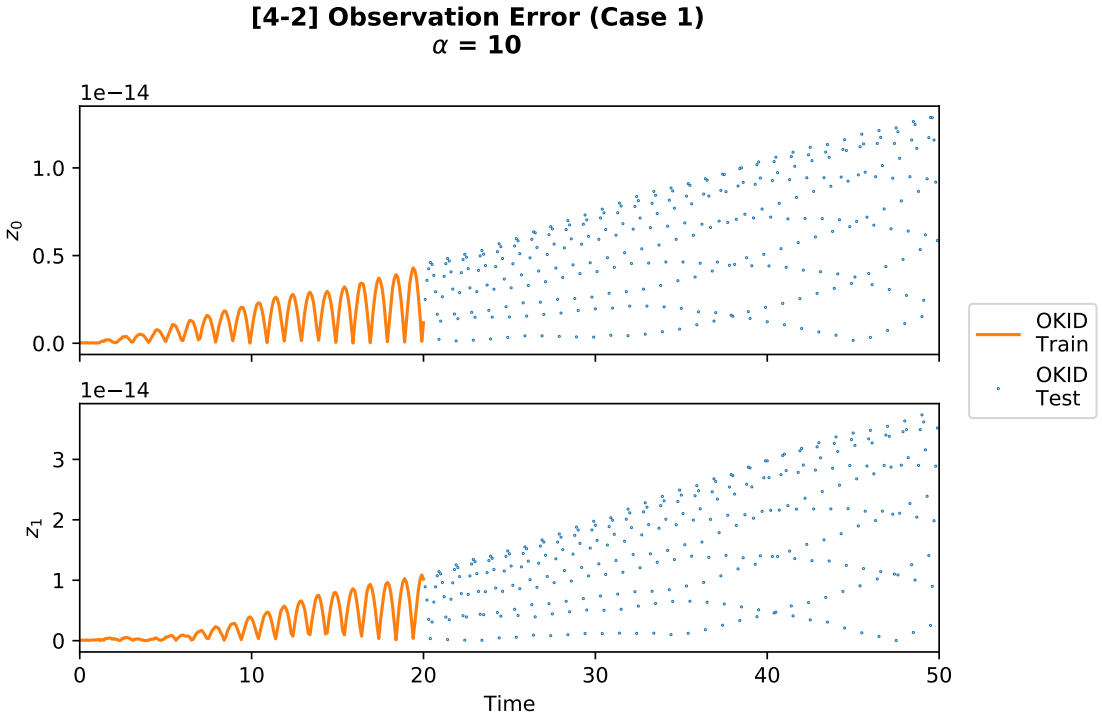
[9]:
```
# Observation plots
for i, k in it.product(range(cases), range(len(alphas))):
    # Raw observations
    raw_fig, axs = plt.subplots(m, 1,
                                sharex = "col", constrained_layout = True) #␣
↪type:figure.Figure
    raw_fig.suptitle(f"[{prob}] Observation Responses (Case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

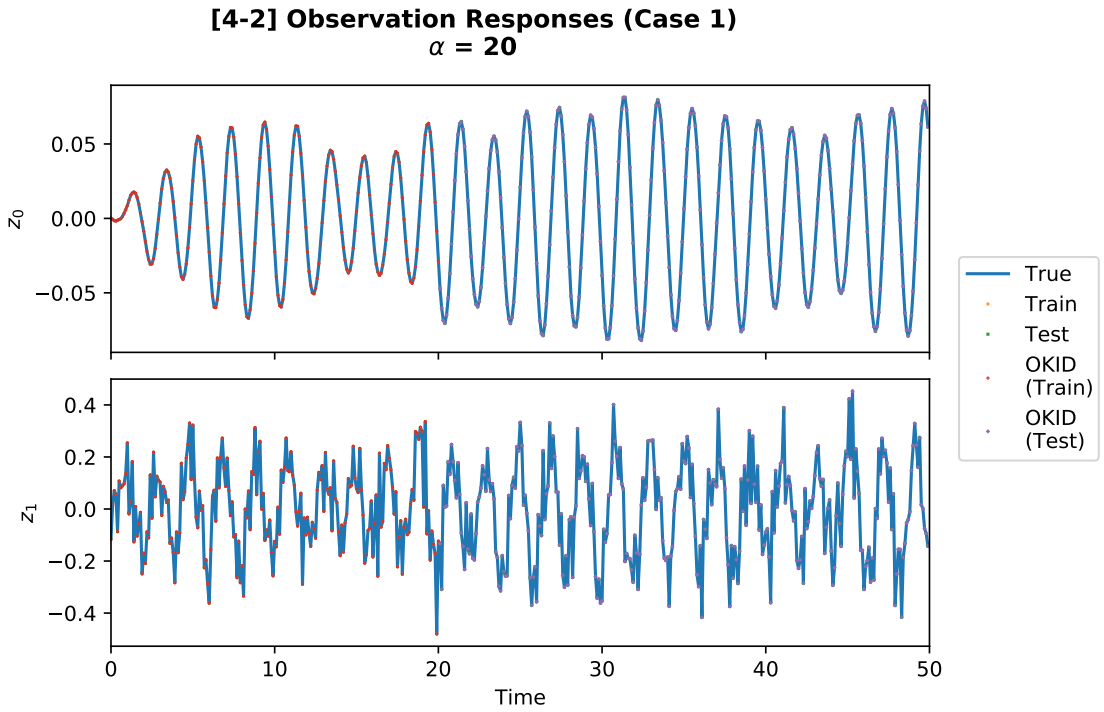
```
    # Observation error
    err_fig, axs = plt.subplots(m, 1,
                                sharex = "col", constrained_layout = True) #␣
↪type:figure.Figure
    err_fig.suptitle(f"[{prob}] Observation Error (Case {i + 1})\n$\\alpha$ =␣
↪{alphas[k]}",
                     fontweight = "bold")
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

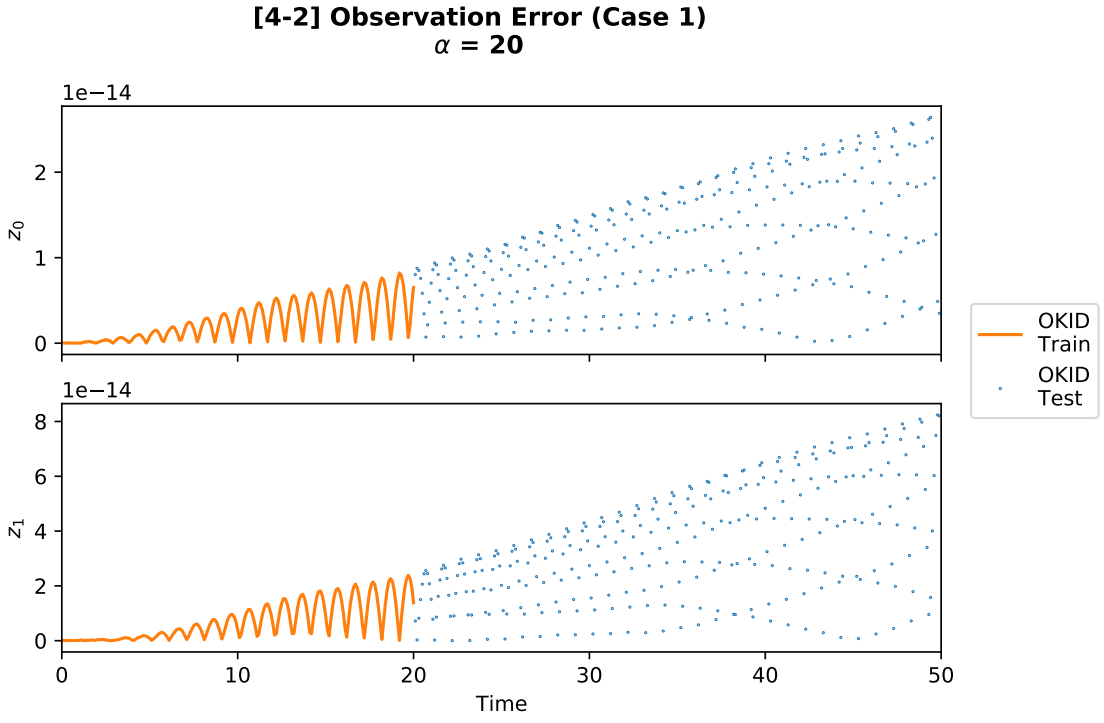

[4-2] Observation Responses (Case 1)
α = 5
0.05
0.00
−0.05
$z_0$
0.4
0.2
0.0
−0.2
−0.4
$z_1$
0
10
20
30
40
50
Time
True
Train
Test
OKID (Train)
OKID (Test)

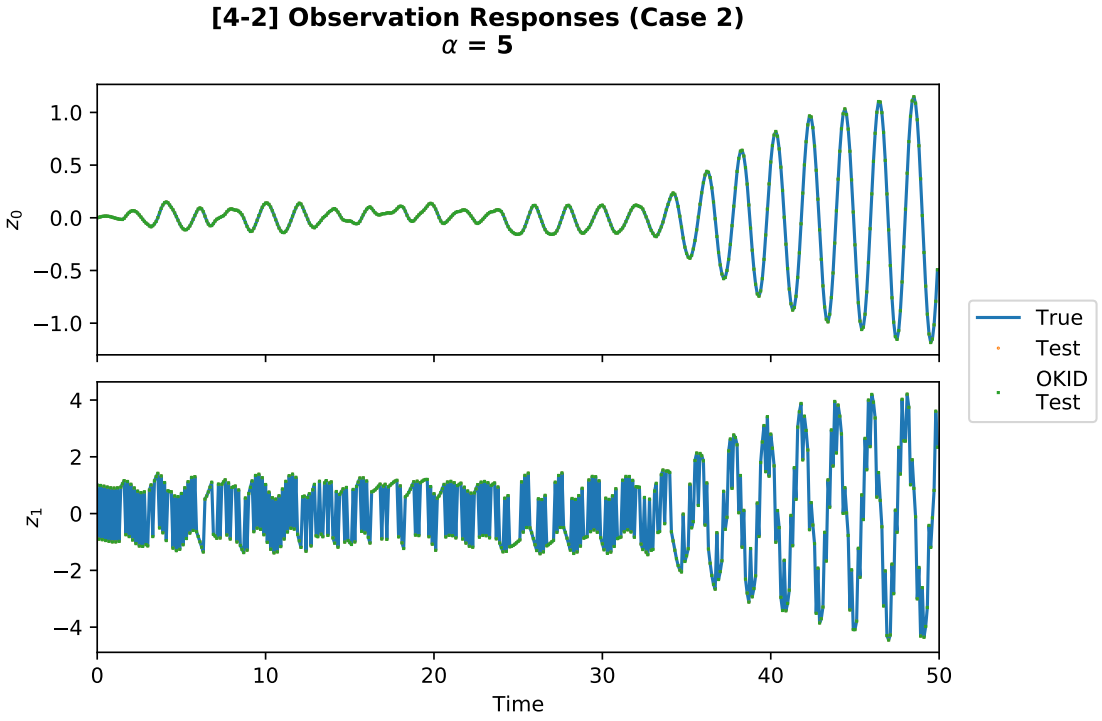

[4-2] Observation Error (Case 1)
α = 5
1e−13
1.00
0.75
0.50
0.25
0.00
$z_0$
1e−13
3
2
1
0
$z_1$
0
10
20
30
40
50
Time
OKID Train
OKID Test

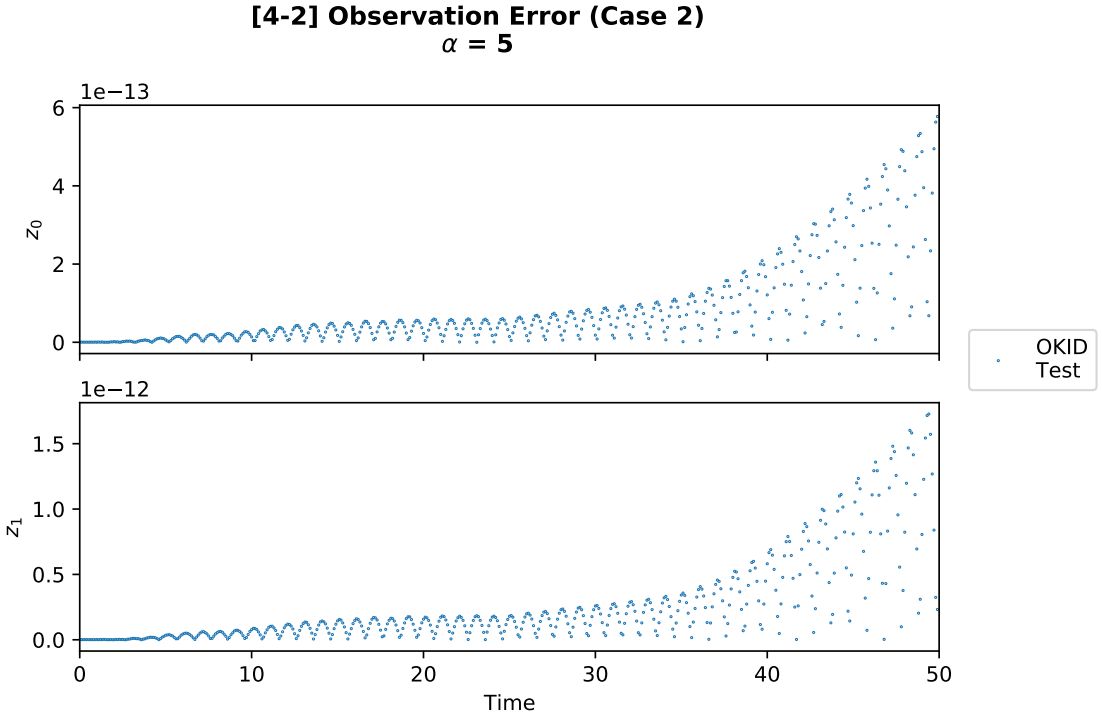

[4-2] Observation Responses (Case 1)
α = 10
0.05
0.00
−0.05
$z_0$
0.4
0.2
0.0
−0.2
−0.4
$z_1$
0
10
20
30
40
50
Time
True
Train
Test
OKID (Train)
OKID (Test)

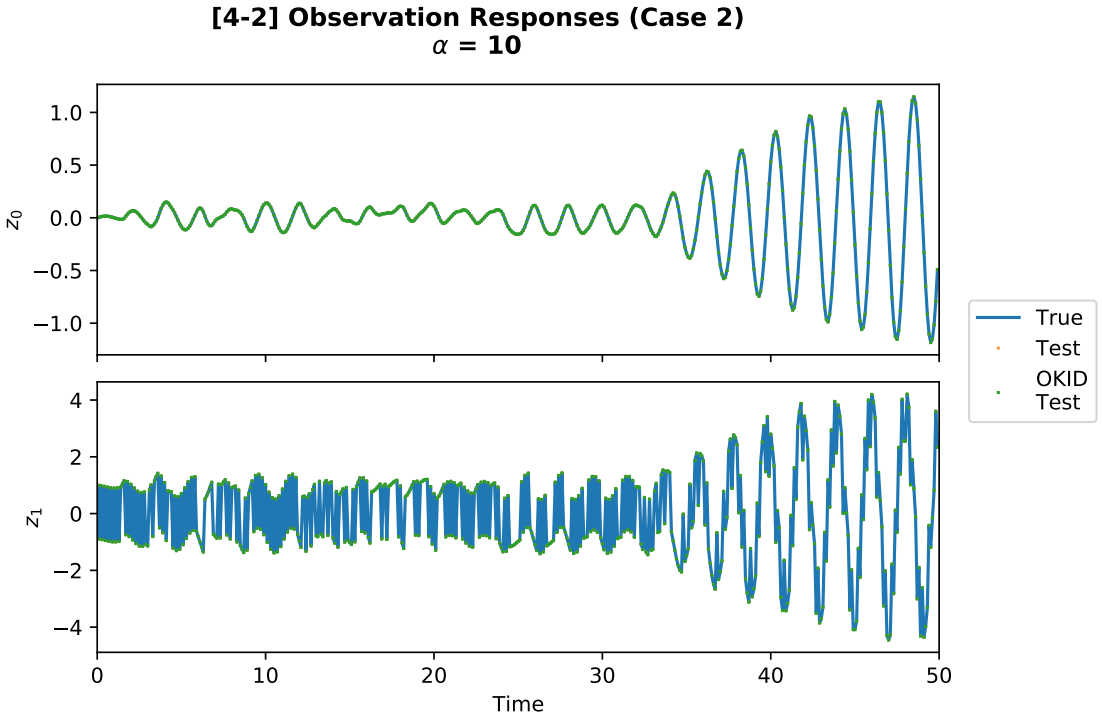

[4-2] Observation Error (Case 1)
α = 10
1e−14
1.0
0.5
0.0
$z_0$
1e−14
3
2
1
0
$z_1$
0
10
20
30
40
50
Time
OKID Train
OKID Test

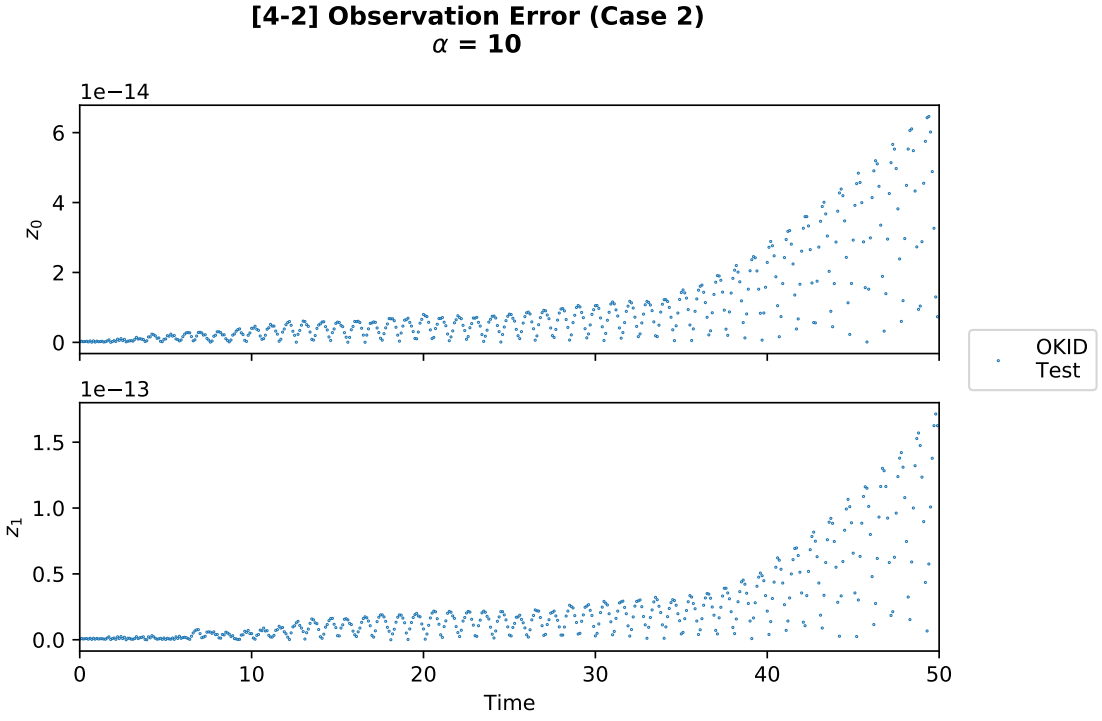

[4-2] Observation Responses (Case 1)
α = 20
0.05
0.00
−0.05
$z_0$
0.4
0.2
0.0
−0.2
−0.4
$z_1$
0
10
20
30
40
50
Time
True
Train
Test
OKID (Train)
OKID (Test)

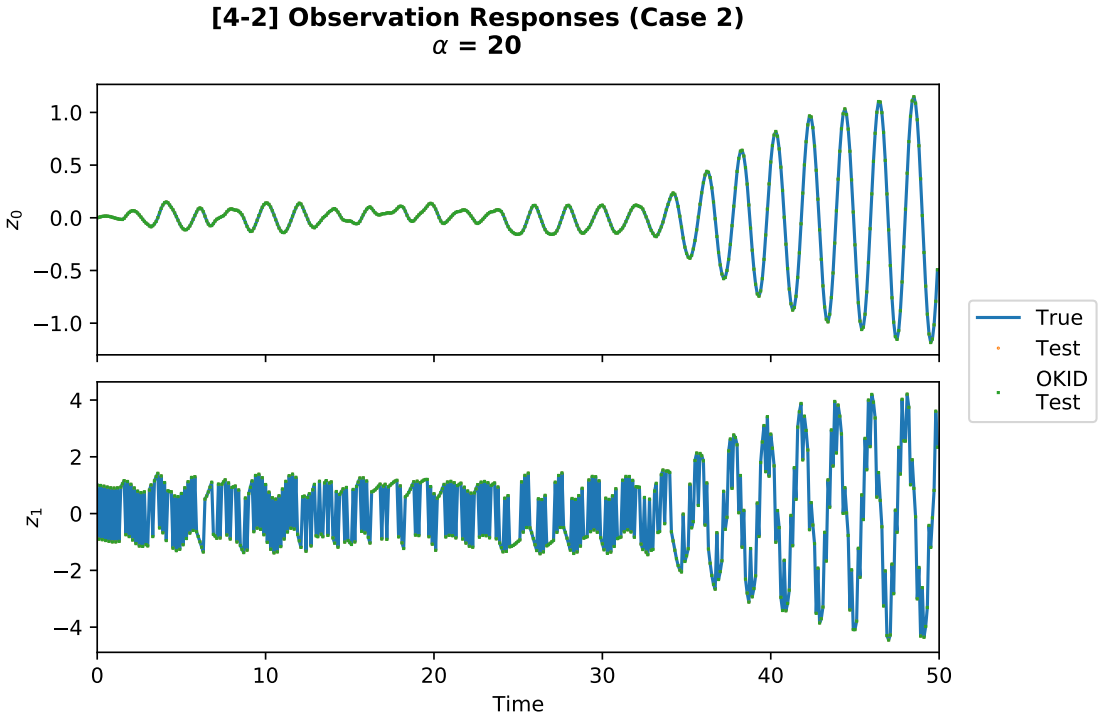

[4-2] Observation Error (Case 1)
α = 20
1e−14
2
1
0
$z_0$
1e−14
8
6
4
2
0
$z_1$
0
10
20
30
40
50
Time
OKID Train
OKID Test

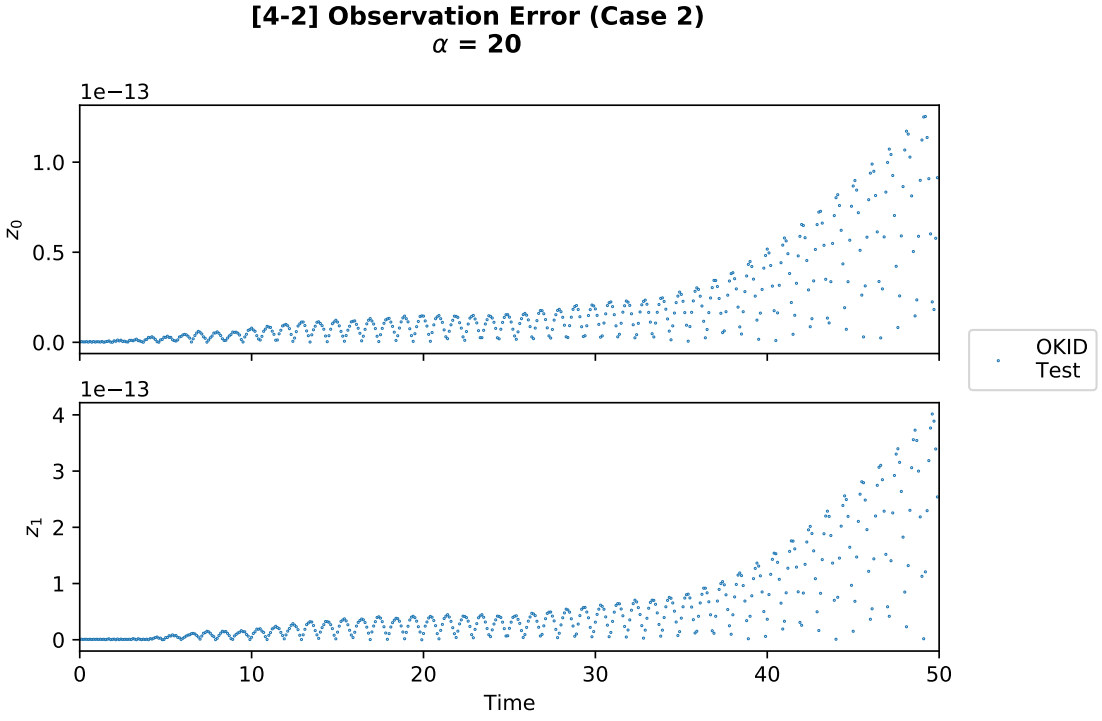

[4-2] Observation Responses (Case 2)
α = 5
$z_0$
$z_1$
Time
True
Test
OKID
Test

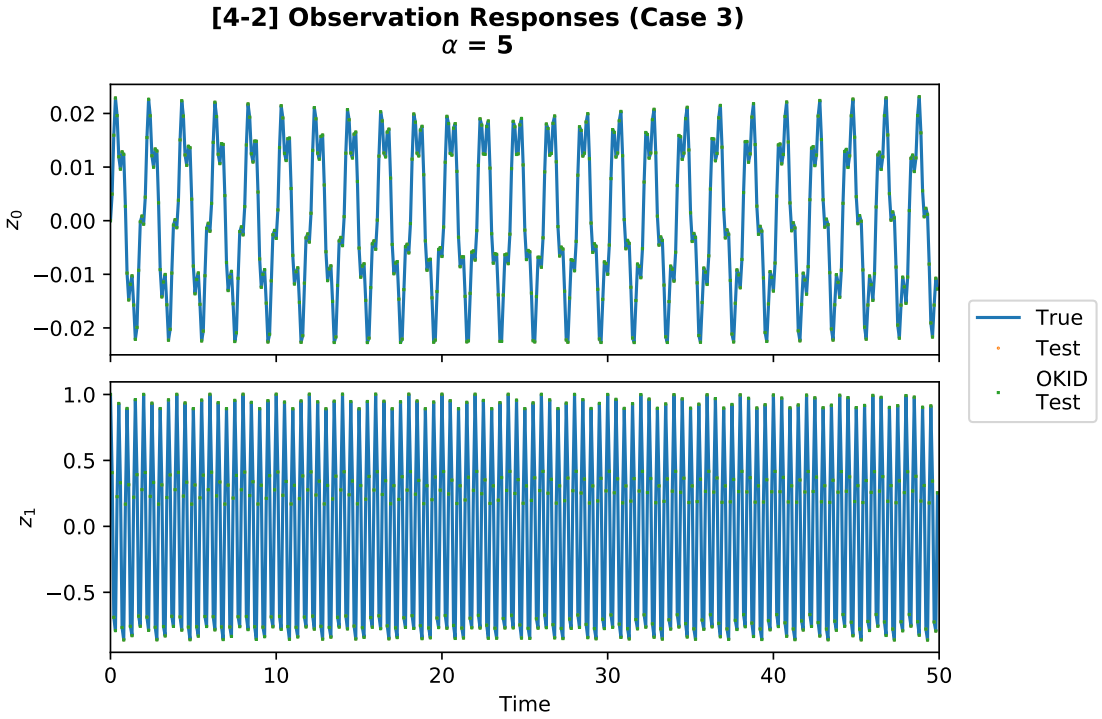

[4-2] Observation Error (Case 2)
α = 5
1e−13
1e−12
$z_0$
$z_1$
Time
OKID
Test

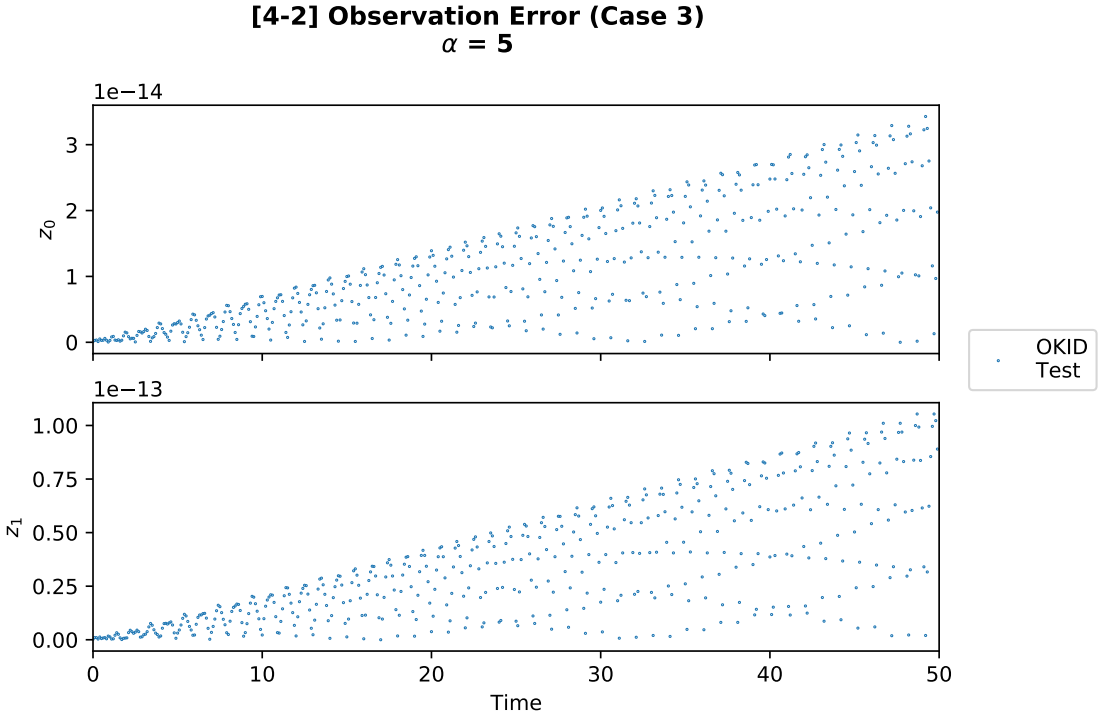

[4-2] Observation Responses (Case 2)
α = 10
z0
z1
Time
True
Test
OKID
Test

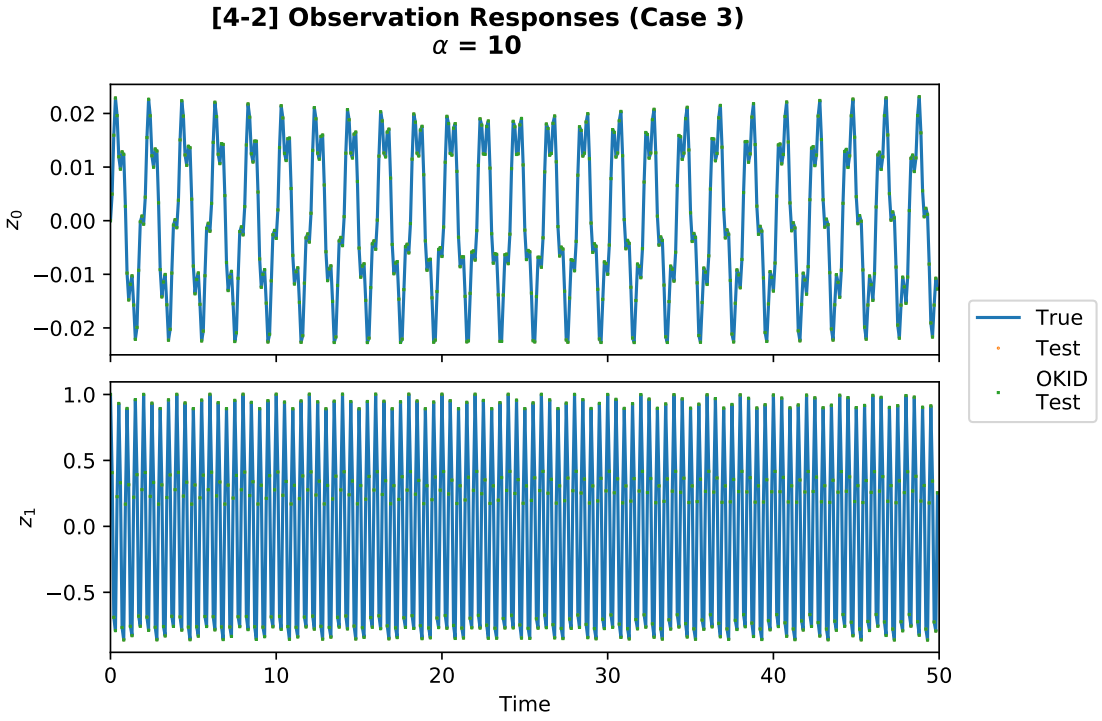

[4-2] Observation Error (Case 2)
α = 10
1e−14
1e−13
z0
z1
Time
OKID
Test

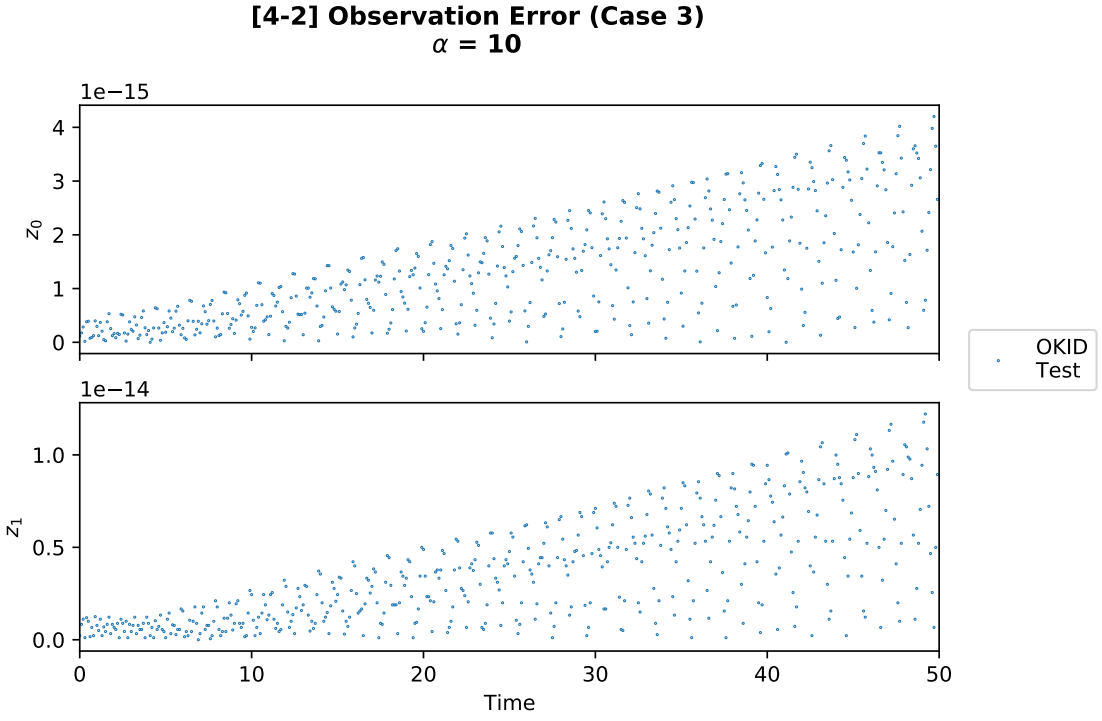

[4-2] Observation Responses (Case 2)
α = 20
$z_0$
$z_1$
Time
True
Test
OKID
Test

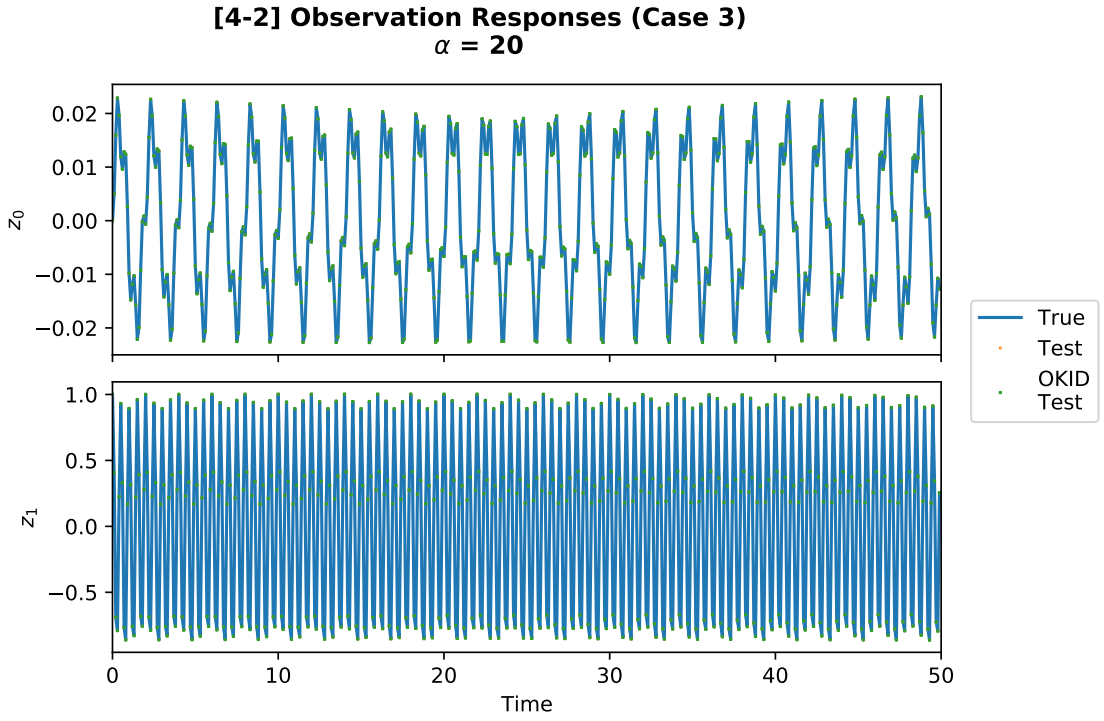

[4-2] Observation Error (Case 2)
α = 20
1e−13
1e−13
$z_0$
$z_1$
Time
OKID
Test

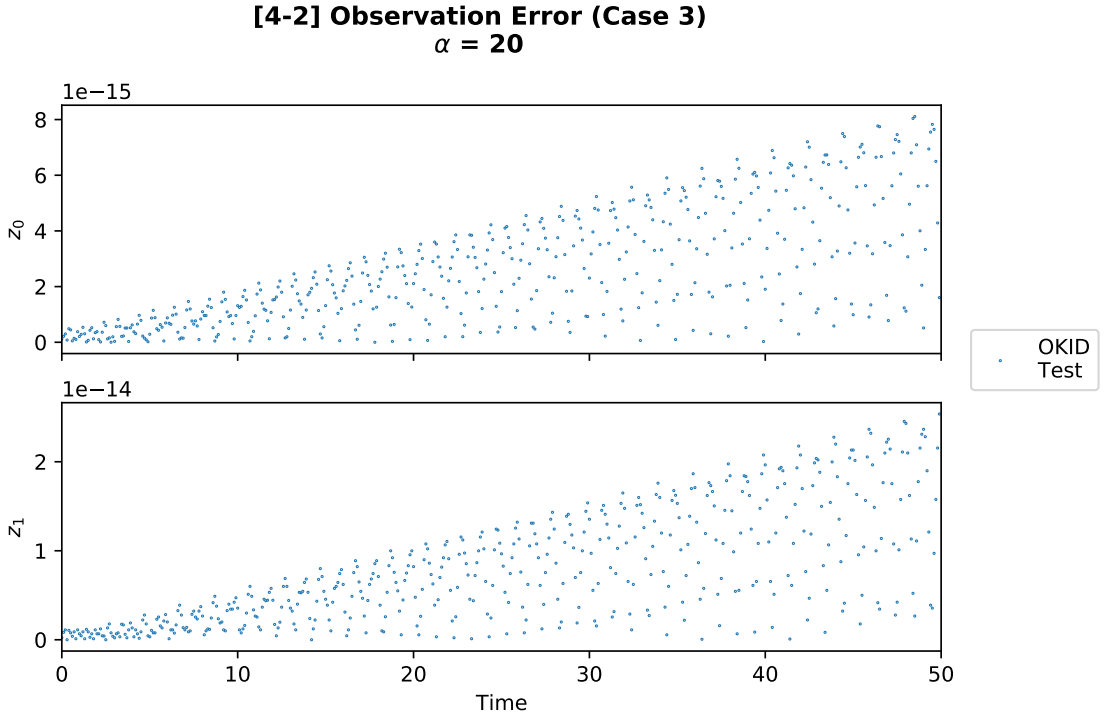

[4-2] Observation Responses (Case 3)
α = 5
0.02
0.01
0.00
−0.01
−0.02
$z_0$
1.0
0.5
0.0
−0.5
$z_1$
0
10
20
30
40
50
Time
True
Test
OKID
Test


[4-2] Observation Error (Case 3)
α = 5
1e−14
3
2
1
0
$z_0$
1e−13
1.00
0.75
0.50
0.25
0.00
$z_1$
0
10
20
30
40
50
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
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 = 20
1e−15
8
6
4
2
0
$z_0$
1e−14
2
1
0
$z_1$
0
10
20
30
40
50
Time
OKID
Test

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.

# 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 = "4-3"

# Set seed for consistent results
rng = np.random.default_rng(seed = 100)

# Simulation dimensions
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_sim = 0.1 # Simulation timestep duration
nt_sim = int(t_max/dt_sim) # Number of simulation timesteps
dt_sample = 0.2 # Sample timestep duration
nt_sample = int(t_max/dt_sample) # Number of sample timesteps
interval = int(dt_sample/dt_sim) # Sample index interval

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

# 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_sim)
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_sim]) # True input vectors
U_sim[0] = rng.normal(0, 0.1, [r, nt_sim]) # 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_sim + 1]) # True state vectors
Z_sim = np.zeros([cases, m, nt_sim]) # True observation vectors

# Sampled simulation values
U_sample = U_sim[:, :, ::interval] # Sampled input vectors
X_sample = np.zeros([cases, n, nt_sample + 1]) # Sampled state vectors
Z_sample = np.zeros([cases, n, nt_sample]) # Sampled observation vectors

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

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

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

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

[2]:
```
# OKID logistics
```

```
order = 10 # Order of OKID algorithm, number of Markov parameters to identify␣
 ↪after the zeroeth
alpha, beta = 3, 5 # 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

print(f"Min. OKID Order: {max([alpha + beta, (n_era/m) + (n_era/r)]):n}")
print(f"Max. OKID Order: {(nt_train - r)/(r + m):n}")
print(f"Proposed OKID Order: {order:n}")
```

```
Min. OKID Order: 8
Max. OKID Order: 33.3333
Proposed OKID Order: 10
```

Note that we have set $l_0 = 10$, $\alpha = 3$, and $\beta = 5$ for this simulation. We choose the sampling frequency to be 5 Hz rather than 10 Hz to determine the effect of sampling frequency on the system identification.

[3]:
```
# OKID System Markov parameters
Y_okid = np.zeros([order + 1, m, r])
# OKID Observer Markov Gain parameters
Y_og_okid = np.zeros([order, m, m])
# OKID state vector, drawn from state space model derived from OKID/ERA
X_okid_train = np.zeros([n_era, nt_train + 1])
X_okid_test = np.zeros([cases, n_era, nt_test + 1])
X_okid_train_obs = np.zeros([n_era, nt_train + 1])
X_okid_test_obs = np.zeros([cases, n_era, nt_test + 1])
# OKID observations, drawn from state space model derived from OKID/ERA
Z_okid_train = np.zeros([n_era, nt_train])
Z_okid_test = np.zeros([cases, n_era, nt_test])
Z_okid_train_obs = np.zeros([n_era, nt_train])
Z_okid_test_obs = np.zeros([cases, n_era, nt_test])

# Singular values of the Hankel matrix constructed through OKID Markov␣
 ↪parameters
S_okid = np.zeros([min(alpha*m, beta*r)])
eig_A_okid = np.zeros([n_era], dtype = complex)

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

```
D_okid_obs = np.zeros([m, r + m])
```

[4]:
```
# Simulation
for i in range(cases):
    X_sim[i], Z_sim[i] = sim_ss(A, B, C, D, X_0 = X_0_sim, U = U_sim[i], nt =␣
↪nt_sim)
    # Sample at lower frequency
    X_sample[i], Z_sample[i] = X_sim[i, :, ::interval], Z_sim[i, :, ::interval]
    if i == 0:
        # Split between train and test data for case 1
        X_train, Z_train = X_sample[i, :, :train_cutoff], Z_sample[i, :, :
↪train_cutoff]
        # Identify System Markov parameters and Observer Gain Markov parameters
        Y_okid, Y_og_okid = okid(Z_train, U_train,
                                 l_0 = order, alpha = alpha, beta = beta, n =␣
↪n_era)
        # Identify state space model using System Markov parameters for ERA
        A_okid, B_okid, C_okid, D_okid, S_okid = \
            era(Y_okid, alpha = alpha, beta = beta, n = n_era)
        # Construct observability matrix
        O_p_okid = np.array([C_okid @ np.linalg.matrix_power(A_okid, i)
                             for i in range(order)])
        G_okid = 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 = A_okid + G_okid @ C_okid
        B_okid_obs = np.concatenate([B_okid + G_okid @ D_okid, -G_okid], 1)
        C_okid_obs = C_okid
        D_okid_obs = np.concatenate([D_okid, np.zeros([m, m])], 1)
        V_train = np.concatenate([U_train, Z_train], 0)
        # Simulate OKID realization with "raw" state and OKID realization with␣
↪estimated state
        X_okid_train, Z_okid_train = \
            sim_ss(A_okid, B_okid, C_okid, D_okid,
                   X_0 = X_0_okid, U = U_train, nt = nt_train)
        X_okid_train_obs, Z_okid_train_obs = \
            sim_ss(A_okid_obs, B_okid_obs, C_okid_obs, D_okid_obs,
                   X_0 = X_0_okid, U = V_train, nt = nt_train)
        # Display outputs
        etch(f"A_{{OKID}}(f_s = {1/dt_sample:0.2f})", A_okid)
        etch(f"B_{{OKID}}(f_s = {1/dt_sample:0.2f})", B_okid)
        etch(f"C_{{OKID}}(f_s = {1/dt_sample:0.2f})", C_okid)
        etch(f"D_{{OKID}}(f_s = {1/dt_sample:0.2f})", D_okid)
        etch(f"G_{{OKID}}(f_s = {1/dt_sample:0.2f})", G_okid)
        # Calculate and display eigenvalues
        eig_A_okid = spla.eig(d2c(A_okid, B_okid, dt_sample)[0])[0] #␣
↪Eigenvalues of identified system
```

```
        etch(f"\hat{{\lambda}}(f_s = {1/dt_sample:0.2f})", eig_A_okid)
        etch(f"\hat{{\omega}}_{{n}}(f_s = {1/dt_sample:0.2f})", np.
↪abs(eig_A_okid))
        etch(f"\hat{{\zeta}}(f_s = {1/dt_sample:0.2f})", -np.cos(np.
↪angle(eig_A_okid)))
    X_test[i], Z_test[i] = X_sample[i], Z_sample[i]
    X_okid_test[i], Z_okid_test[i] = \
        sim_ss(A_okid, B_okid, C_okid, D_okid,
               X_0 = X_0_okid, U = U_test[i], nt = nt_test)
    V_test[i] = np.concatenate([U_test[i], Z_test[i]], 0)
    X_okid_test_obs[i], Z_okid_test_obs[i] = \
        sim_ss(A_okid_obs, B_okid_obs, C_okid_obs, D_okid_obs,
               X_0 = X_0_okid, U = V_test[i], nt = nt_test)
```

```
Rank of H(0): 5
Rank of H(1): 5
```

$$A_{OKID}(f_s = 5.00) = \begin{bmatrix} 0.83094 & 0.74587 \\ -0.46364 & 0.81261 \end{bmatrix}$$

$$B_{OKID}(f_s = 5.00) = \begin{bmatrix} -0.1028 \\ 0.22645 \end{bmatrix}$$

$$C_{OKID}(f_s = 5.00) = \begin{bmatrix} 0.04672 & 0.0903 \\ -0.25341 & 0.23411 \end{bmatrix}$$

$$D_{OKID}(f_s = 5.00) = \begin{bmatrix} -0.00029 \\ 0.99433 \end{bmatrix}$$

$$G_{OKID}(f_s = 5.00) = \begin{bmatrix} -0.66531 & -0.46935 \\ -0.88417 & -1.05222 \end{bmatrix}$$

$$\hat{\lambda}(f_s = 5.00) = \begin{bmatrix} 0.05207 + 3.1053i \\ 0.05207 - 3.1053i \end{bmatrix}$$

$$\hat{\omega}_n(f_s = 5.00) = \begin{bmatrix} 3.10574 \\ 3.10574 \end{bmatrix}$$

$$\hat{\zeta}(f_s = 5.00) = \begin{bmatrix} -0.01677 \\ -0.01677 \end{bmatrix}$$

The sampling frequency being halved leads to the detection of slight damping in the system where there is in fact none. The identified natural frequency is close to the true value. However, the fact that the detected eigenvalues have positive (unstable) real parts means that the identification as a whole is quite poor, even with the observer.

[5]:
```
RMS_train = np.sqrt(np.mean((Z_okid_train - Z_train)**2, axis = 1))
print(f"RMS Error of sim. for system found via OKID for train data, sampling␣
↪frequency = {1/dt_sample:0.2f}: {RMS_train}")
RMS_test = np.zeros([cases, m])
for i in range(cases):
```

```
    RMS_test[i] = np.sqrt(np.mean((Z_okid_test[i] - Z_test[i])**2, axis = 1))
    print(f"RMS Error of sim. for system found via OKID for test data, case␣
↪{i}, sampling frequency = {1/dt_sample:0.2f}: {RMS_test[i]}")
```

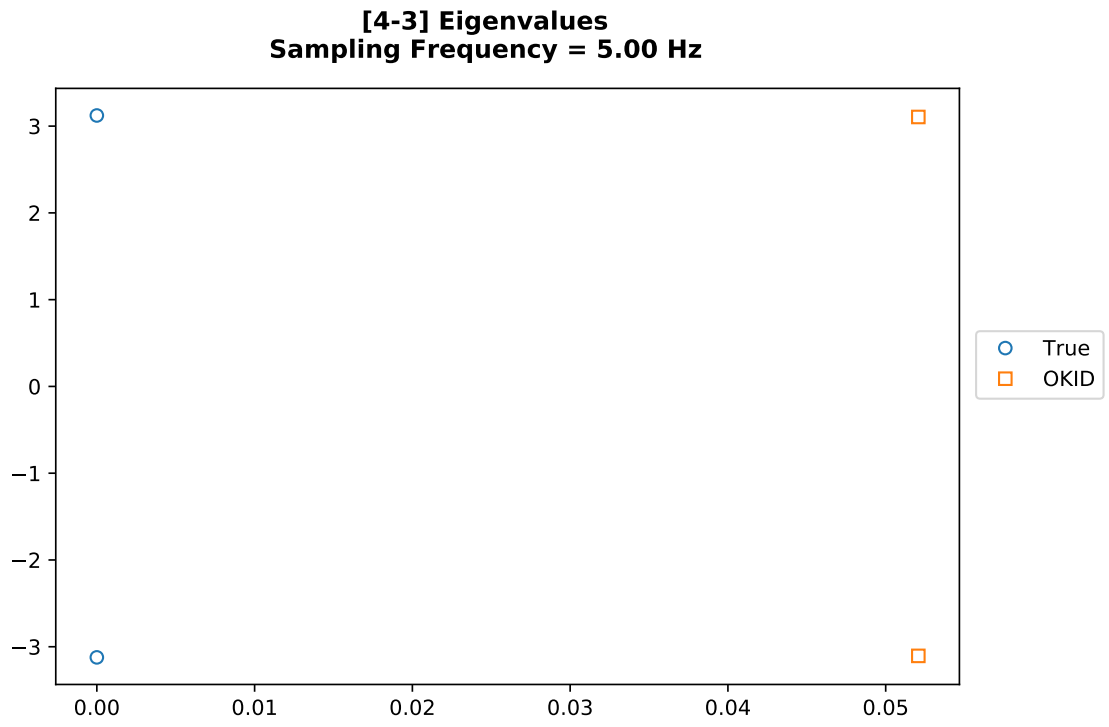
```
RMS Error of sim. for system found via OKID for train data, sampling frequency =
5.00: [0.01258072 0.02951865]
RMS Error of sim. for system found via OKID for test data, case 0, sampling
frequency = 5.00: [0.06219972 0.24769425]
RMS Error of sim. for system found via OKID for test data, case 1, sampling
frequency = 5.00: [0.09802495 0.37483967]
RMS Error of sim. for system found via OKID for test data, case 2, sampling
frequency = 5.00: [0.0483794  0.19170376]
```

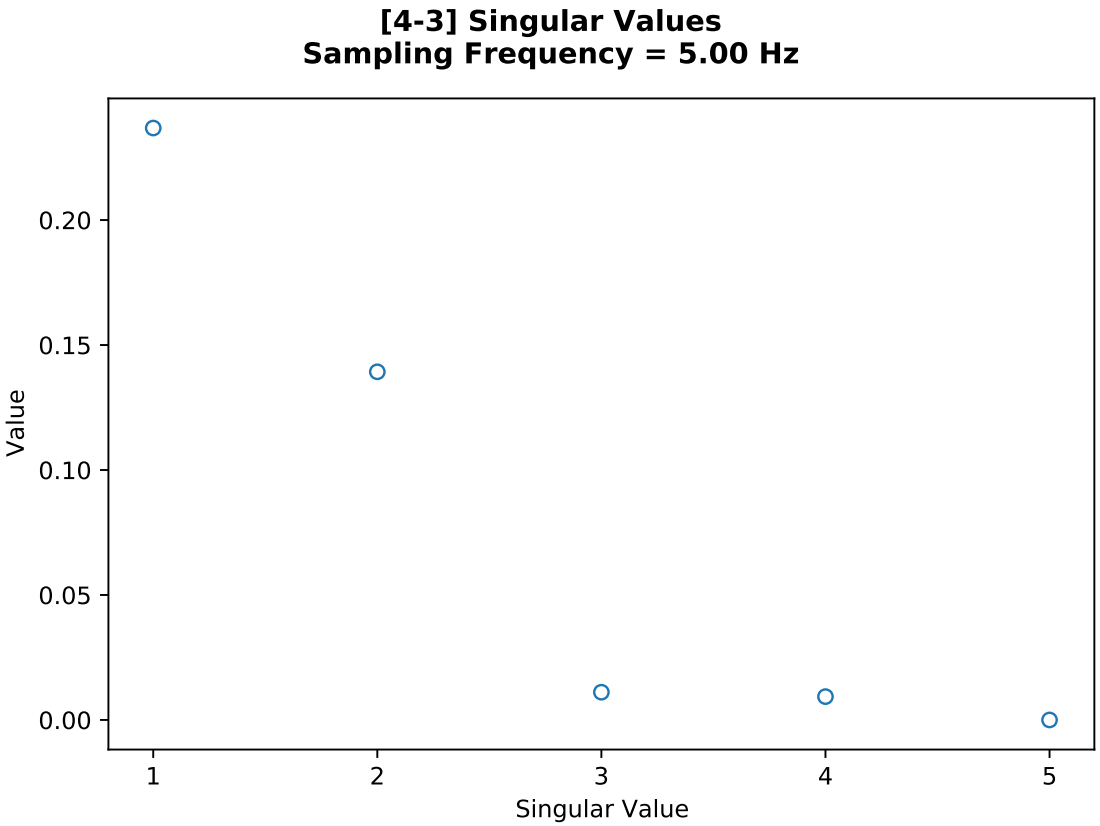
The RMS error in the estimation is quite high for both the training and testing data. The training case has an input frequency of double the sampling frequency, again contributing to the poor accuracy of the identified system as it cannot cope with this.

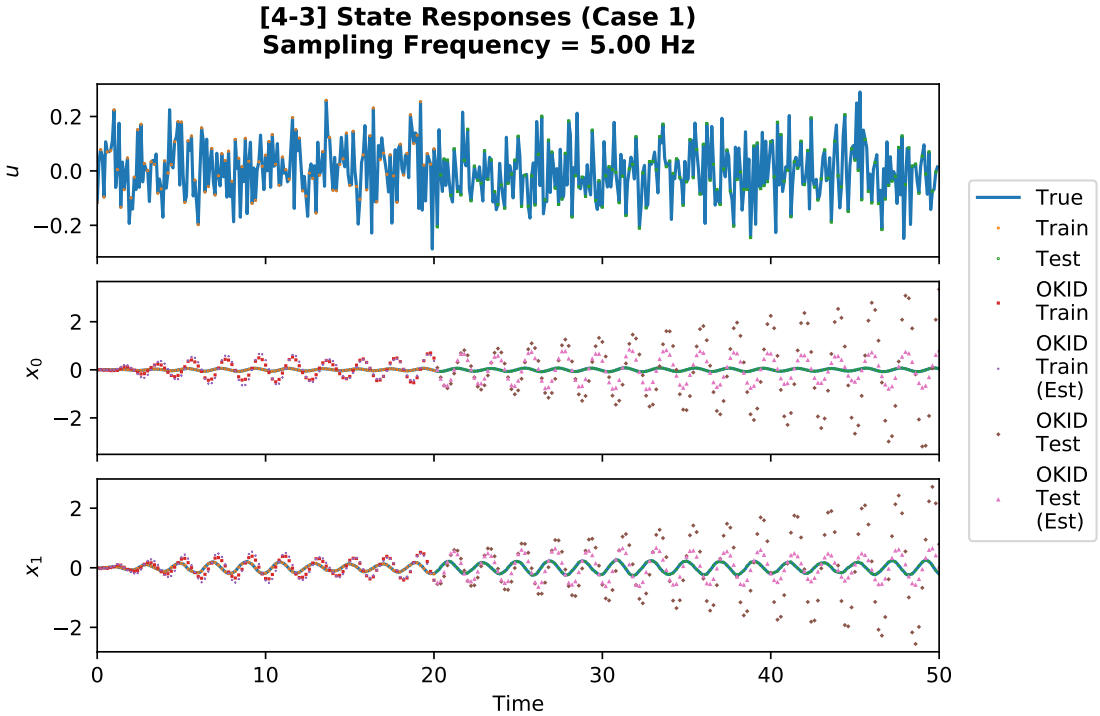
[6]:
```
# Eigenvalue plots
fig, ax = plt.subplots(constrained_layout = True) # type:figure.Figure
fig.suptitle(f"[{prob}] Eigenvalues\nSampling Frequency = {1/dt_sample:0.2f}␣
↪Hz", fontweight = "bold")

ax.plot(np.real(eig_A), np.imag(eig_A),
        "o", mfc = "None")
ax.plot(np.real(eig_A_okid), np.imag(eig_A_okid),
        "s", mfc = "None")

fig.legend(labels = ("True", "OKID"),
           bbox_to_anchor = (1, 0.5), loc = 6)
fig.savefig(figs_dir / f"midterm_{prob}_eigval.pdf",
            dpi = 80, bbox_inches = "tight")

# Singular Value plots
fig, ax = plt.subplots(constrained_layout = True) # type:figure.Figure
fig.suptitle(f"[{prob}] Singular Values\nSampling Frequency = {1/dt_sample:0.
↪2f} Hz", fontweight = "bold")

ax.plot(np.linspace(1, len(S_okid), len(S_okid)), S_okid,
        "o", mfc = "None")
plt.setp(ax, xlabel = f"Singular Value", ylabel = f"Value",
        xticks = np.arange(1, len(S_okid) + 1))

fig.savefig(figs_dir / f"midterm_{prob}_singval.pdf",
            dpi = 80, bbox_inches = "tight")

# Response plots
ms = 0.5 # Marker size
for i in range(cases):
```

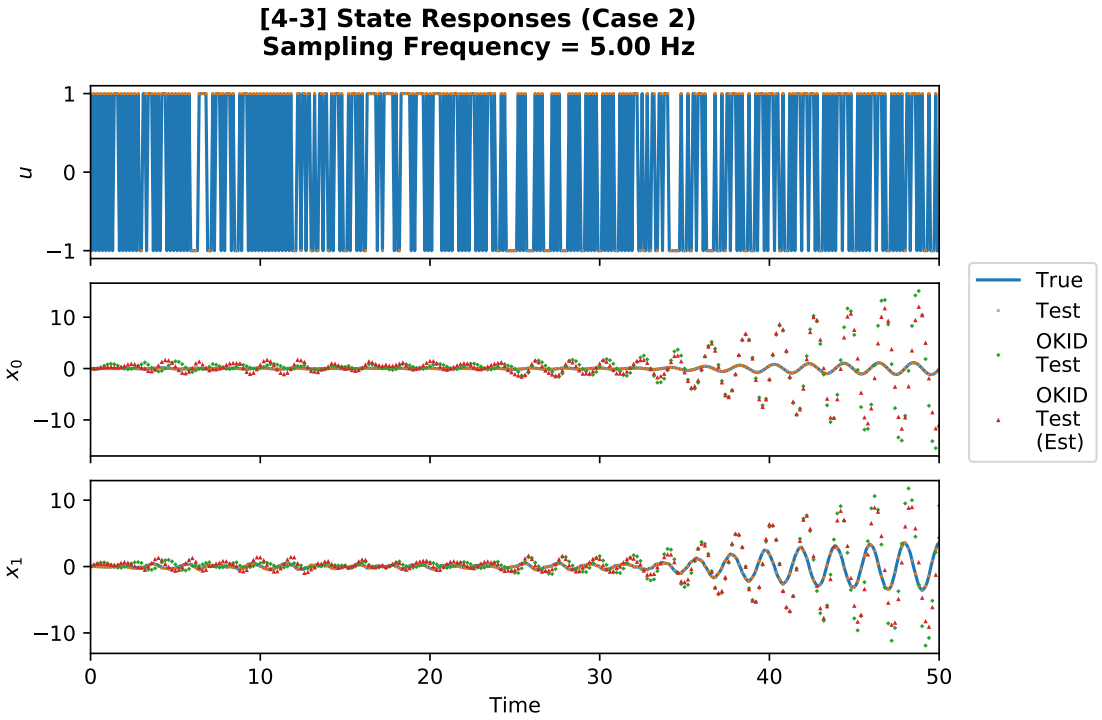
```
    fig, axs = plt.subplots(1 + n, 1,
                            sharex = "col",
                            constrained_layout = True) # type:figure.Figure
    fig.suptitle(f"[{prob}] State Responses (Case {i + 1})\nSampling Frequency␣
↪= {1/dt_sample:0.2f} Hz",
                 fontweight = "bold")

    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[j],
                            "o", ms = ms, mfc = "None")
            axs[j + 1].plot(t_test[train_cutoff:], X_test[i, j, train_cutoff:],
                            "o", ms = ms, mfc = "None")
            axs[j + 1].plot(t_train, X_okid_train[j, :-1],
                            "s", ms = ms, mfc = "None")
            axs[j + 1].plot(t_train, X_okid_train_obs[j, :-1],
                            "*", ms = ms, mfc = "None")
            axs[j + 1].plot(t_test[train_cutoff:], X_okid_test[i, j,␣
↪train_cutoff:],
                            "D", ms = ms, mfc = "None")
            axs[j + 1].plot(t_test[train_cutoff:], X_okid_test_obs[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[i, j],
```

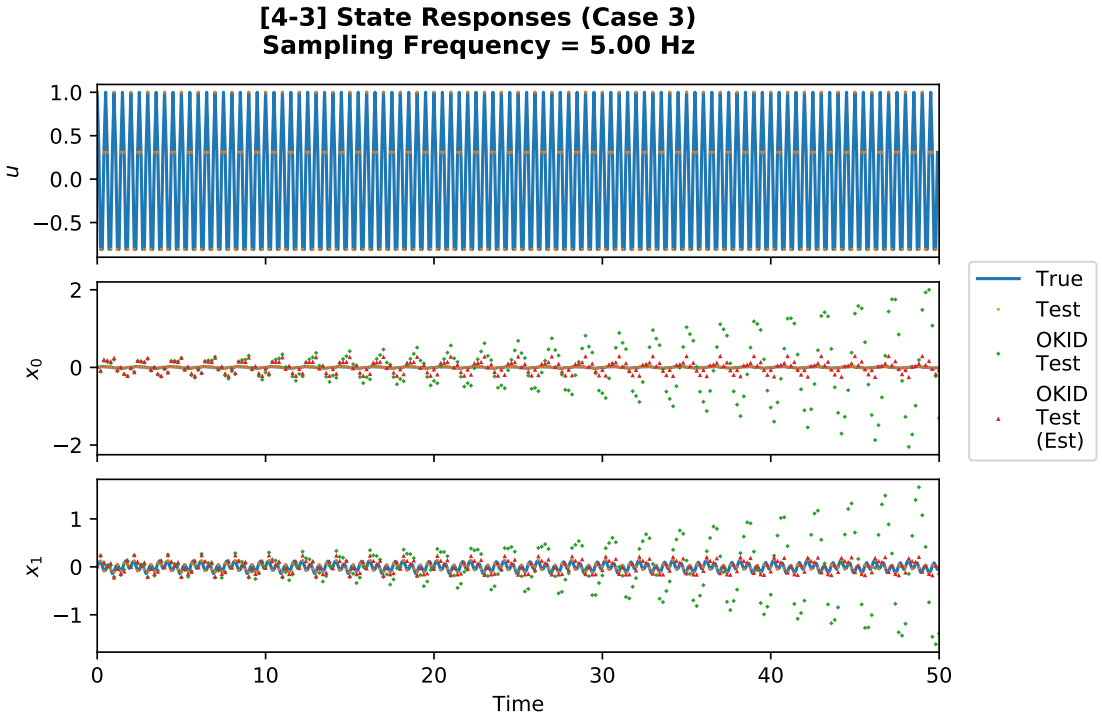
```
                        "o", ms = ms, mfc = "None")
        axs[j + 1].plot(t_test, X_okid_test[i, j],
                        "D", ms = ms, mfc = "None")
        axs[j + 1].plot(t_test, X_okid_test_obs[i, j],
                        "^", ms = ms, mfc = "None")
        plt.setp(axs[j + 1], ylabel = 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}.pdf",
            bbox_inches = "tight")
```




[4-3] Singular Values
Sampling Frequency = 5.00 Hz
Value
0.20
0.15
0.10
0.05
0.00
1
2
3
4
5
Singular Value


[4-3] State Responses (Case 1)
Sampling Frequency = 5.00 Hz
u
0.2
0.0
−0.2
x0
2
0
−2
x1
2
0
−2
0
10
20
30
40
50
Time
True
Train
Test
OKID Train
OKID Train (Est)
OKID Test
OKID Test (Est)

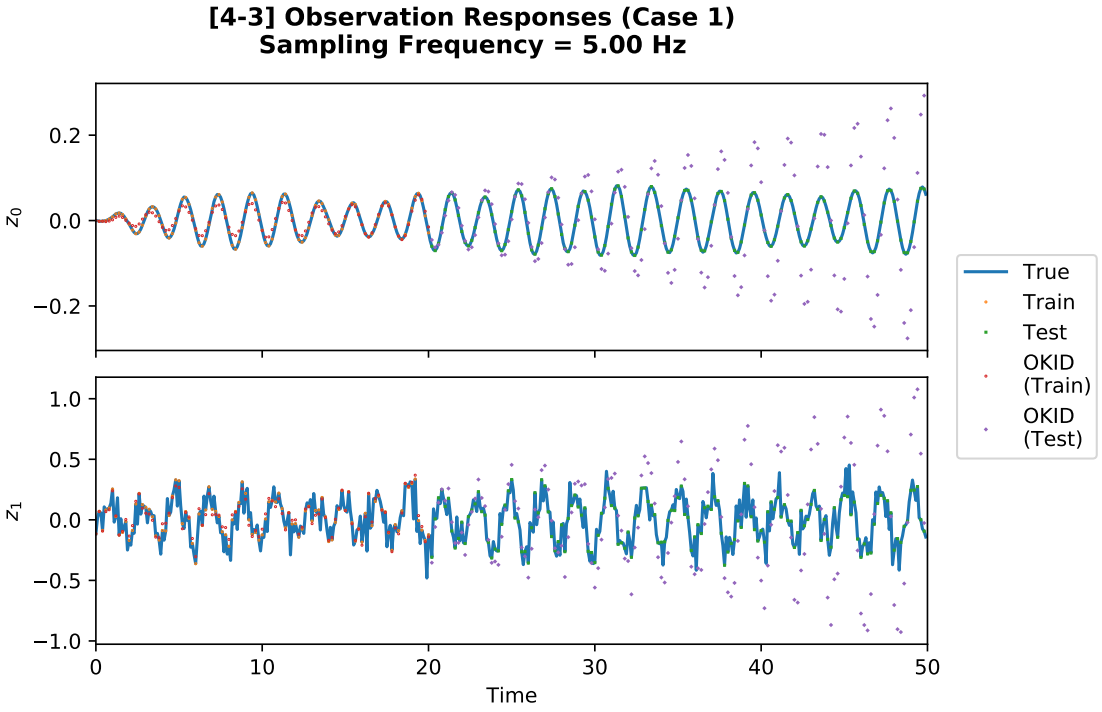

[4-3] State Responses (Case 2)
Sampling Frequency = 5.00 Hz
u
x0
x1
Time
True
Test
OKID Test
OKID Test (Est)

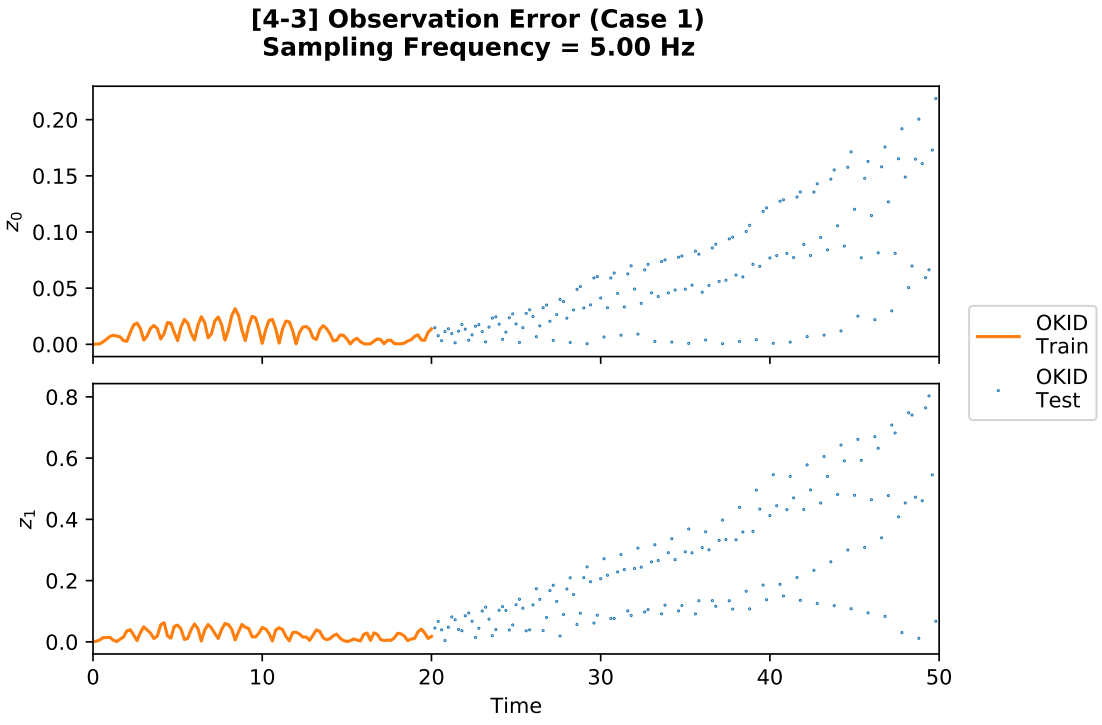

[4-3] State Responses (Case 3)
Sampling Frequency = 5.00 Hz
u
x0
x1
Time
True
Test
OKID Test
OKID Test (Est)

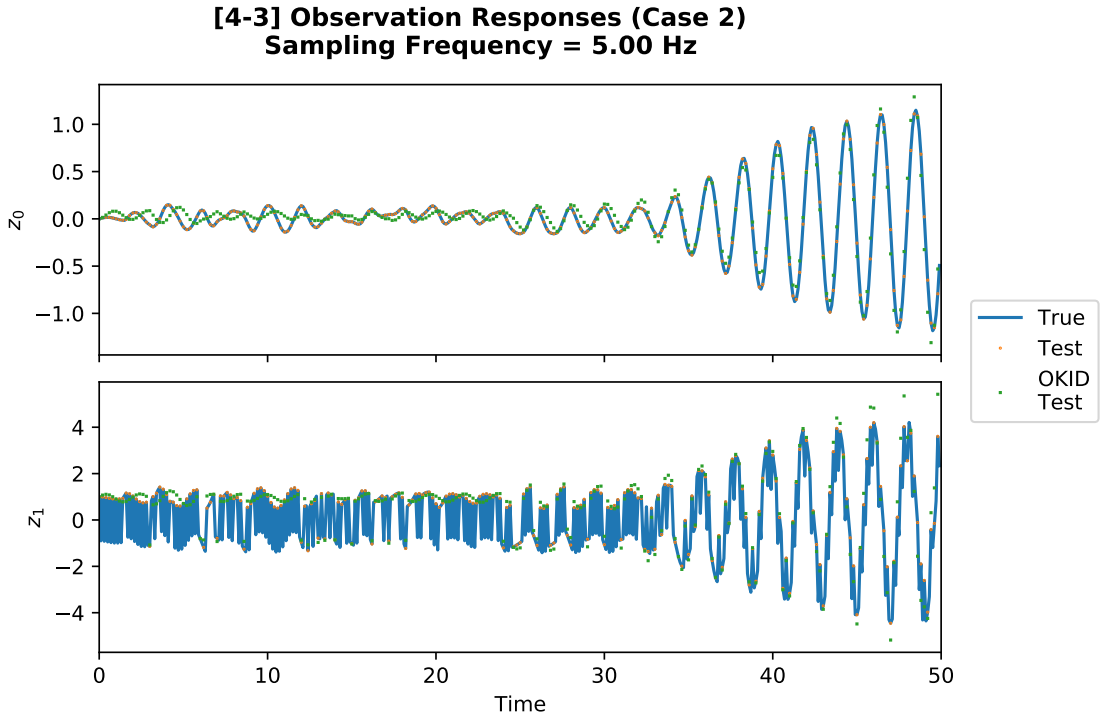
The observer appears to cause the state estimate to grow in time for all three cases, which is testament to the fact that the identification of the system is very poor.

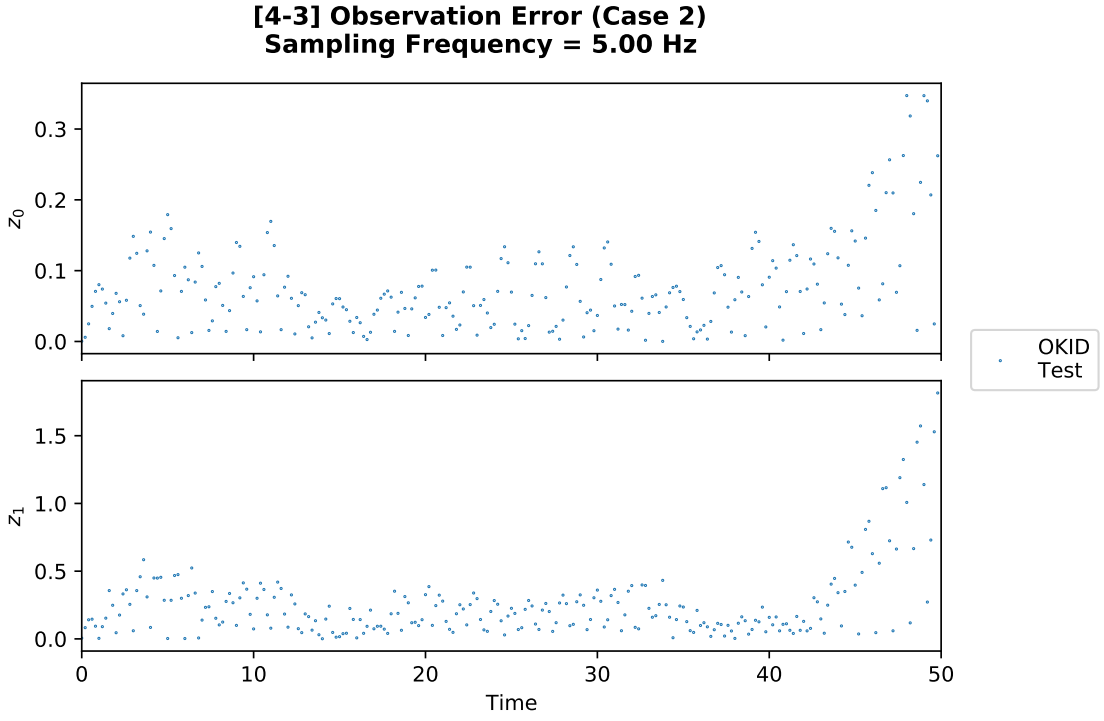
[7]:
```
# Observation plots
for i in range(cases):
    fig, axs = plt.subplots(m, 1,
                            sharex = "col",
                            constrained_layout = True) # type:figure.Figure
    fig.suptitle(f"[{prob}] Observation Responses (Case {i + 1})\nSampling␣
↪Frequency = {1/dt_sample:0.2f} Hz",
                 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[j],
                        "o", ms = ms, mfc = "None")
            axs[j].plot(t_test[train_cutoff:-1], Z_test[i, j, train_cutoff:],
                        "s", ms = ms, mfc = "None")
            axs[j].plot(t_train, Z_okid_train[j],
                        "o", ms = ms, mfc = "None")
            axs[j].plot(t_test[train_cutoff:-1], Z_okid_test[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")
        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[i, j],
                        "o", ms = ms, mfc = "None")
            axs[j].plot(t_test[:-1], Z_okid_test[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")
        fig.legend(labels = ["True", "Test", "OKID\nTest"],
                   bbox_to_anchor = (1, 0.5), loc = 6)
    fig.savefig(figs_dir / f"midterm_{prob}_obs_case{i + 1}.pdf",
                dpi = 80, bbox_inches = "tight")
```

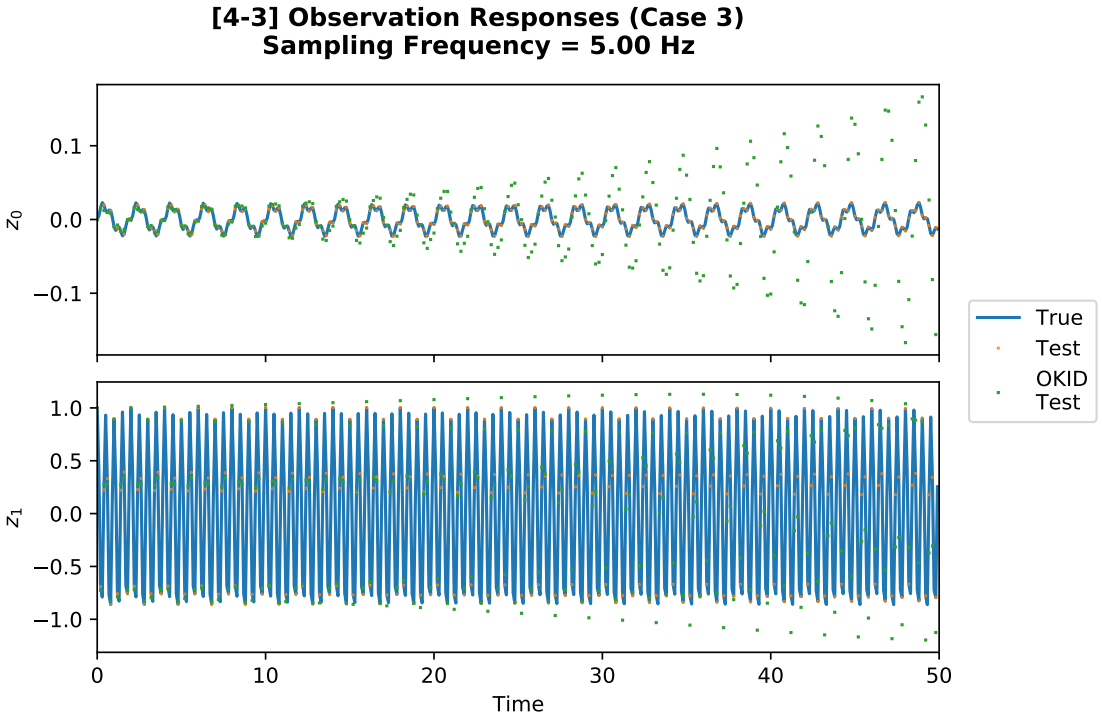
```
    fig, axs = plt.subplots(m, 1,
                            sharex = "col",
                            constrained_layout = True) # type:figure.Figure
    fig.suptitle(f"[{prob}] Observation Error (Case {i + 1})\nSampling␣
↪Frequency = {1/dt_sample:0.2f} Hz",
                 fontweight = "bold")
    if i == 0:
        for j in range(m):
            axs[j].plot(t_train, np.abs(Z_okid_train[j] - Z_train[j]),
                        c = "C1")
            axs[j].plot(t_test[train_cutoff:-1], np.abs(Z_okid_test[i, j,␣
↪train_cutoff:] - Z_test[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")
        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[i, j] - Z_test[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")
        fig.legend(labels = ["OKID\nTest"],
                   bbox_to_anchor = (1, 0.5), loc = 6)
    fig.savefig(figs_dir / f"midterm_{prob}_obs-error_case{i + 1}.pdf",
                dpi = 80, bbox_inches = "tight")
```

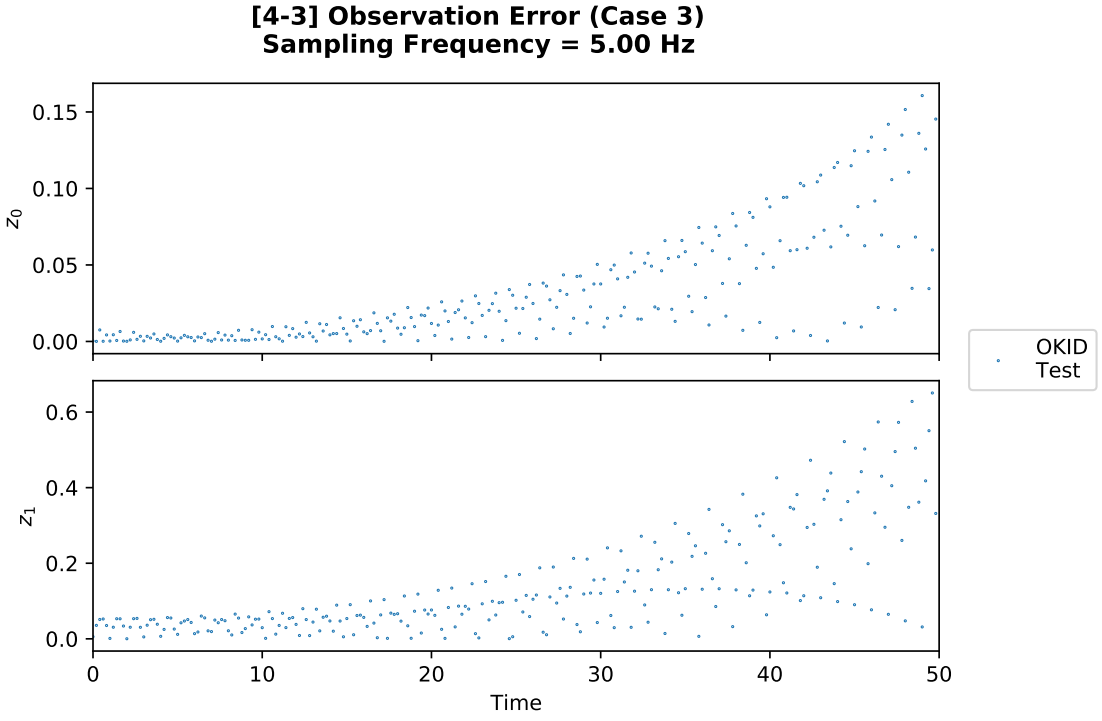

[4-3] Observation Responses (Case 1)
Sampling Frequency = 5.00 Hz
$z_0$
$z_1$
Time
True
Train
Test
OKID (Train)
OKID (Test)


[4-3] Observation Error (Case 1)
Sampling Frequency = 5.00 Hz
$z_0$
$z_1$
Time
OKID Train
OKID Test


[4-3] Observation Responses (Case 2)
Sampling Frequency = 5.00 Hz
$z_0$
$z_1$
Time
True
Test
OKID
Test


[4-3] Observation Error (Case 2)
Sampling Frequency = 5.00 Hz
$z_0$
$z_1$
Time
OKID
Test


[4-3] Observation Responses (Case 3)
Sampling Frequency = 5.00 Hz
$z_0$
$z_1$
Time
True
Test
OKID
Test


[4-3] Observation Error (Case 3)
Sampling Frequency = 5.00 Hz
$z_0$
$z_1$
Time
OKID
Test

On the whole, the estimation is quite poor in this case due to the loss of half of all the random input and the corresponding output data. Due to this reason, decreasing the sample frequency significantly worsened the accuracy of the estimation. For a simpler training dataset where the input was more uniform and less random, the estimation accuracy for lower sampling frequencies may have been slightly improved.