System Identification

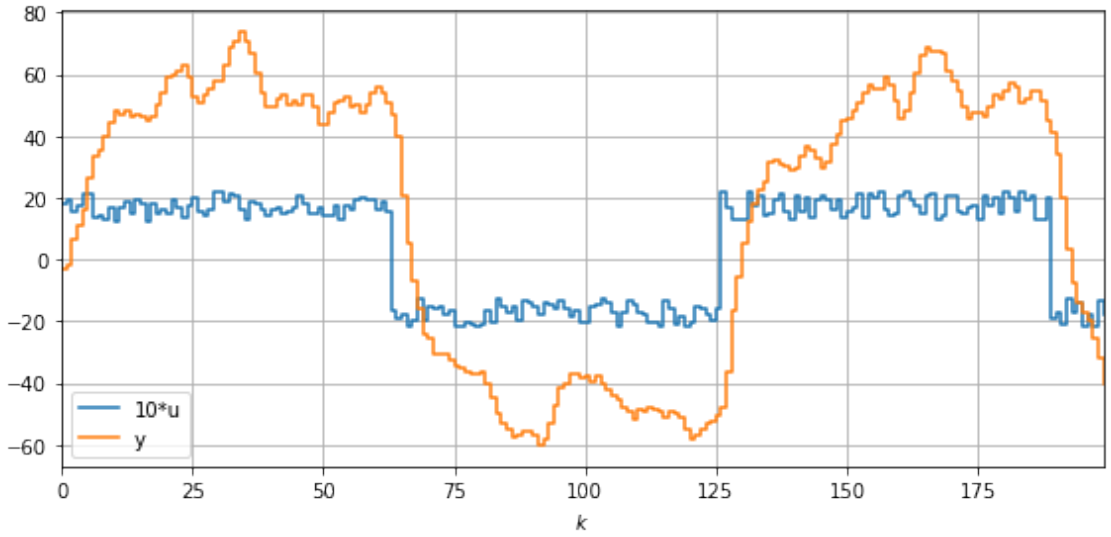
Guilherme Beal

May 22, 2023

1 Load and View Data

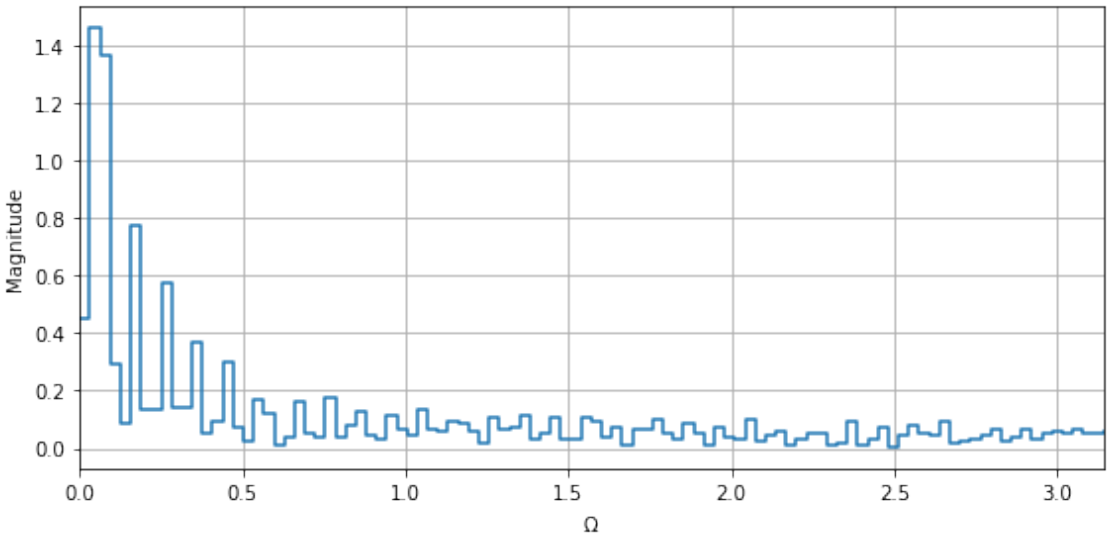
[ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

file = '../data.csv'
data = pd.read_csv(file, header=None, names=['u', 'y'])
N = len(data)

k = data.index.values
u = data.u.values
y = data.y.values

print('Number of data points:', N)
print(f'k in [{k[0]}, {k[-1]}]')

plt.figure(figsize=(8,4))
plt.plot(k, 10*u, label='10*u', drawstyle='steps-post')
plt.plot(k, y   , label='y'   , drawstyle='steps-post')
plt.xlim(k[0], k[-1])
plt.xlabel(r'$k$')
plt.grid()
plt.legend()
plt.tight_layout()
plt.show()
```

```
Number of data points: 200
k in [0, 199]
```



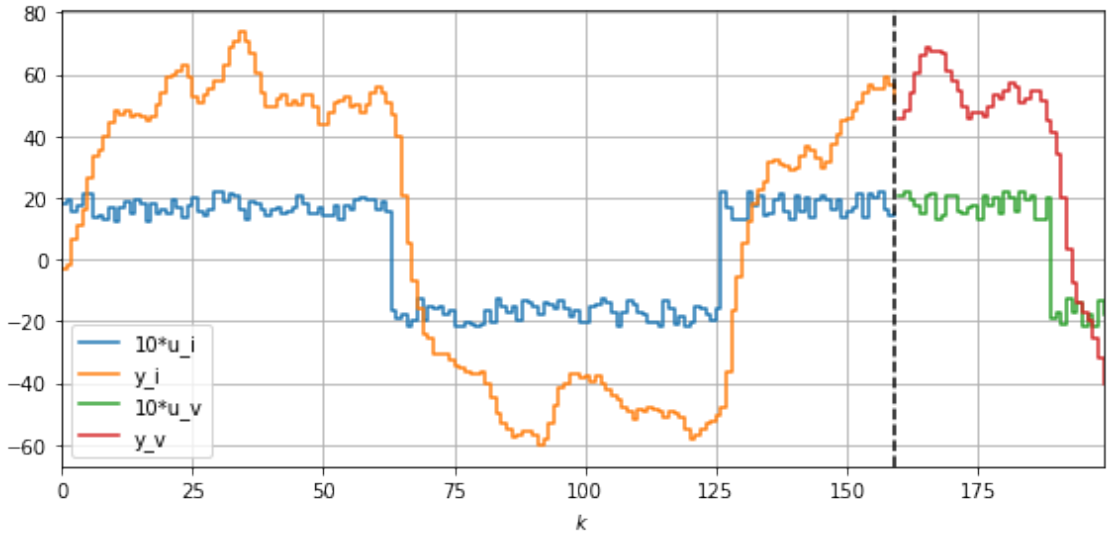
1.1 Input Fourier Transform

[ ]:
```
from scipy import fft

u_rfft = fft.rfft(u, norm='forward')
u_rfft[1:-1] = 2*u_rfft[1:-1]

u_rfft_mag = np.abs(u_rfft)
Omega = np.linspace(0, np.pi, len(u_rfft_mag))

plt.figure(figsize=(8,4))
plt.plot(Omega, u_rfft_mag, drawstyle='steps-post')
plt.xlim(Omega[0], Omega[-1])
plt.xlabel(r'$\Omega$')
plt.ylabel('Magnitude')
plt.grid()
plt.tight_layout()
plt.show()
```



1.2 Separate Identification and Validation Data

[ ]:
```
N_fold = 160

k_i = k[:N_fold]
u_i = u[:N_fold]
y_i = y[:N_fold]

k_v = k[N_fold:]
u_v = u[N_fold:]
y_v = y[N_fold:]

plt.figure(figsize=(8,4))
plt.plot(k_i, 10*u_i, label='10*u_i', drawstyle='steps-post')
plt.plot(k_i, y_i   , label='y_i'   , drawstyle='steps-post')
plt.plot(k_v, 10*u_v, label='10*u_v', drawstyle='steps-post')
plt.plot(k_v, y_v   , label='y_v'   , drawstyle='steps-post')
plt.axvline(k[N_fold-1], color='black', linestyle='--')
plt.xlim(k[0], k[-1])
plt.xlabel(r'$k$')
plt.grid()
plt.legend()
plt.tight_layout()
plt.show()
```



2 Generic Model

$$A(q)\,y[k] = \frac{B(q)}{F(q)}\,u[k] + \frac{C(q)}{D(q)}\,e[k]$$

$$y[k] = G(q)\,u[k - n_k + 1] + H(q)\,e[k]$$

$$G(q) = \frac{B(q)}{A(q)\,F(q)} \qquad H(q) = \frac{C(q)}{A(q)\,D(q)}$$

$$A(q) = 1 - a_1\,q^{-1} - \cdots - a_{n_a}\,q^{-n_a}$$

$$B(q) = q^{-n_k}\,\left(b_1 + b_2\,q^{-1} + \cdots + b_{n_b+1}\,q^{-n_b}\right)$$

$$C(q) = 1 + c_1\,q^{-1} + \cdots + c_{n_c}\,q^{-n_c}$$

$$D(q) = 1 + d_1\,q^{-1} + \cdots + d_{n_d}\,q^{-n_d}$$

$$F(q) = 1 + f_1\,q^{-1} + \cdots + f_{n_f}\,q^{-n_f}$$

2.1 Prediction Error Method

$$\hat{y}[k] = L_u(q)\,u[k] + L_y(q)\,y[k]$$

$$L_u(q) = \frac{G(q)}{H(q)}$$

$$L_y(q) = 1 - \frac{1}{H(q)}$$

3 ARX

$$y[k] = G(q)\,u[k] + H(q)\,e[k]$$
$$G(q) = \frac{B(q)}{A(q)} \qquad H(q) = \frac{1}{A(q)}$$
$$A(q) = 1 - a_1\,q^{-1} - \cdots - a_{n_a}\,q^{-n_a}$$
$$B(q) = q^{-n_k}\,\left(b_1 + b_2\,q^{-1} + \cdots + b_{n_b+1}\,q^{-n_b}\right)$$

$$n_a = \{0, 1, 2, 3\} \qquad n_b = \{0, 1, 2\} \qquad n_k = \{1, 2, 3\}$$

[ ]:
```
from functions import arx

na_range = range(0, 3 + 1)
nb_range = range(0, 2 + 1)
nk_range = range(1, 3 + 1)

models_arx = arx(u_i, y_i, u_v, y_v, na_range, nb_range, nk_range)
```

4 ARMAX

$$y[k] = G(q)\,u[k] + H(q)\,e[k]$$
$$G(q) = \frac{B(q)}{A(q)} \qquad H(q) = \frac{C(q)}{A(q)}$$
$$A(q) = 1 - a_1\,q^{-1} - \cdots - a_{n_a}\,q^{-n_a}$$
$$B(q) = q^{-n_k}\,\left(b_1 + b_2\,q^{-1} + \cdots + b_{n_b+1}\,q^{-n_b}\right)$$
$$C(q) = 1 + c_1\,q^{-1} + \cdots + c_{n_c}\,q^{-n_c}$$

$$n_a = \{0, 1, 2, 3\} \qquad n_b = \{0, 1, 2\} \qquad n_c = \{1, 2, 3\} \qquad n_k = \{1, 2, 3\}$$

[ ]:
```
from functions import armax

na_range = range(0, 3 + 1)
nb_range = range(0, 2 + 1)
nc_range = range(1, 3 + 1)
nk_range = range(1, 3 + 1)

models_armax = armax(u_i, y_i, u_v, y_v, na_range, nb_range, nc_range, nk_range)
```

5 Output Error

$$y[k] = G(q)\,u[k] + H(q)\,e[k]$$
$$G(q) = \frac{B(q)}{F(q)} \qquad H(q) = 1$$
$$B(q) = q^{-n_k}\left(b_1 + b_2\,q^{-1} + \cdots + b_{n_b+1}\,q^{-n_b}\right)$$
$$F(q) = 1 + f_1\,q^{-1} + \cdots + f_{n_f}\,q^{-n_f}$$

$$n_b = \{0, 1, 2\} \qquad n_f = \{1, 2, 3\} \qquad n_k = \{1, 2, 3\}$$

[ ]:
```
from functions import oe

nb_range = range(0, 2 + 1)
nf_range = range(1, 3 + 1) # nf = 0 causa erro no pysid!
nk_range = range(1, 3 + 1)

models_oe = oe(u_i, y_i, u_v, y_v, nb_range, nf_range, nk_range)
```

6 Box-Jenkins

$$y[k] = G(q)\,u[k] + H(q)\,e[k]$$
$$G(q) = \frac{B(q)}{F(q)} \qquad H(q) = \frac{C(q)}{D(q)}$$
$$B(q) = q^{-n_k}\left(b_1 + b_2\,q^{-1} + \cdots + b_{n_b+1}\,q^{-n_b}\right)$$
$$C(q) = 1 + c_1\,q^{-1} + \cdots + c_{n_c}\,q^{-n_c}$$
$$D(q) = 1 + d_1\,q^{-1} + \cdots + d_{n_d}\,q^{-n_d}$$
$$F(q) = 1 + f_1\,q^{-1} + \cdots + f_{n_f}\,q^{-n_f}$$

$$n_b = \{0, 1, 2\} \qquad n_c = \{0, 1, 2, 3\} \qquad n_d = \{0, 1, 2, 3\} \qquad n_f = \{0, 1, 2, 3\} \qquad n_k = \{1, 2, 3\}$$

[ ]:
```
from functions import bj # diversos erros

nb_range = range(0, 2 + 1)
nc_range = range(0, 3 + 1)
nd_range = range(0, 3 + 1)
nf_range = range(0, 3 + 1)
nk_range = range(1, 3 + 1)
```

```
models_bj = bj(u_i, y_i, u_v, y_v, nb_range, nc_range, nd_range, nf_range,␣
 ↪nk_range)
```

7 Results

[ ]:
```
from functions import models_frame

models = pd.concat([models_frame(), models_arx, models_armax, models_oe,␣
 ↪models_bj], ignore_index=True)

print('Successful models:', len(models.loc[models.B.notnull()]))
print('Failed models:    ', len(models.loc[models.B.isnull()]))
```

```
Successful models: 531
Failed models:     216
```

7.1 Sort by Prediction Cost

$$J_p = \frac{1}{N}\sum_{k=1}^{N}\left(y[k] - \hat{y}[k]\right)^2$$

[ ]:
```
models.sort_values(by=['Jp'], inplace=True)
```

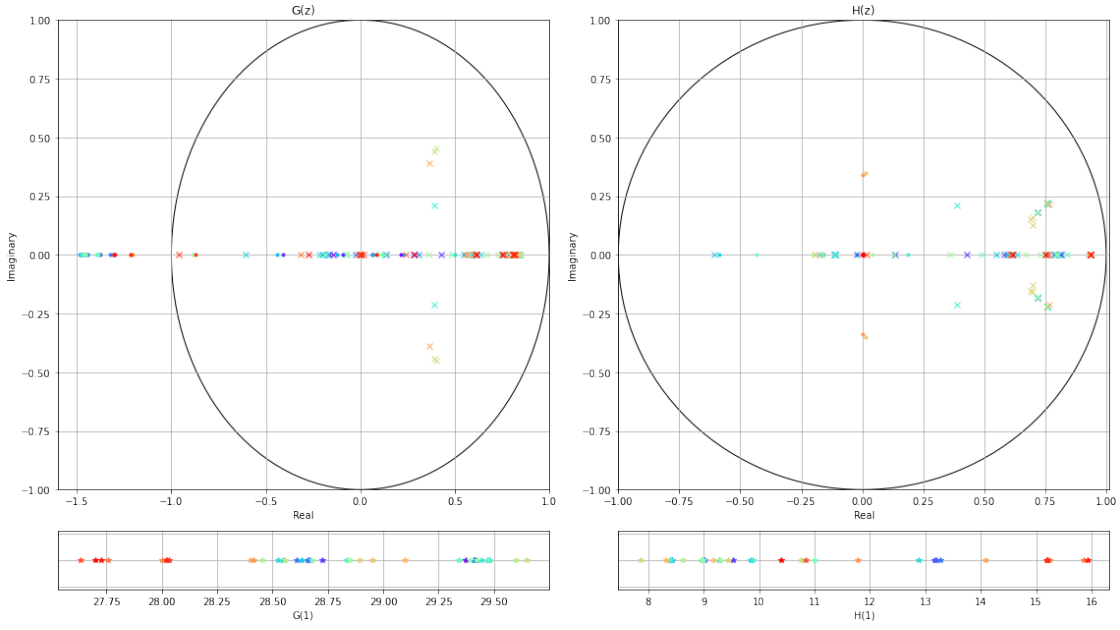
7.2 Display Models with Lowest Cost

[ ]:
```
qty = 40
models_disp = models.copy()
with np.printoptions(precision=4):
  for collumn in ['A', 'B', 'C', 'D', 'F', 'zG', 'pG', 'zH', 'pH']:
    models_disp[collumn] = models_disp[collumn].astype(str)

with pd.option_context('display.precision', 5):
  display(models_disp[['model', 'na', 'nb', 'nc', 'nd', 'nf', 'nk', 'Jp']].
 ↪head(qty))

fig, ax = plt.subplots(2, 2, figsize=(16,9), height_ratios=[8, 1])
colors = iter(plt.cm.rainbow(np.linspace(0, 1, qty)))
for i, (index, model) in enumerate(models.iterrows()):
  if i >= qty:
    break

  color = next(colors)
  for pole in model.pG:
    ax[0][0].plot(pole.real, pole.imag, 'x', c=color)
  for zero in model.zG:
    ax[0][0].plot(zero.real, zero.imag, '.', c=color)
```

```
  ax[1][0].plot(model.kG, 0, '*', c=color)
  for pole in model.pH:
    ax[0][1].plot(pole.real, pole.imag, 'x', c=color)
  for zero in model.zH:
    ax[0][1].plot(zero.real, zero.imag, '.', c=color)
  ax[1][1].plot(model.kH, 0, '*', c=color)

ax[0][0].set_title('G(z)')
ax[0][1].set_title('H(z)')
ax[1][0].set_xlabel('G(1)')
ax[1][1].set_xlabel('H(1)')
for i in range(0, 1+1):
  ax[0][i].add_artist(plt.Circle((0, 0), 1, fill=False))
  ax[0][i].set_xlabel('Real')
  ax[0][i].set_ylabel('Imaginary')
  xlim = ax[0][i].get_xlim()
  ylim = ax[0][i].get_ylim()
  ax[0][i].set_xlim(min(xlim[0], -1), max(xlim[1], 1))
  ax[0][i].set_ylim(min(ylim[0], -1), max(ylim[1], 1))
  ax[0][i].grid()

  ax[1][i].tick_params(axis='y',left=False, labelleft=False)
  ax[1][i].grid()

plt.tight_layout()
plt.show()
```

	model	na	nb	nc	nd	nf	nk	Jp
588	bj	NaN	2	0	2	3	1	5.48709
585	bj	NaN	2	0	2	2	1	5.5212
582	bj	NaN	2	0	2	1	1	5.52389
30	arx	3	1	NaN	NaN	NaN	1	5.55578
108	armax	2	2	1	NaN	NaN	1	5.5785
24	arx	2	2	NaN	NaN	NaN	1	5.58895
33	arx	3	2	NaN	NaN	NaN	1	5.63397
402	bj	NaN	1	0	3	1	1	5.68255
600	bj	NaN	2	0	3	3	1	5.6935
594	bj	NaN	2	0	3	1	1	5.721
597	bj	NaN	2	0	3	2	1	5.72537
21	arx	2	1	NaN	NaN	NaN	1	5.80084
126	armax	3	1	1	NaN	NaN	1	5.81388
135	armax	3	2	1	NaN	NaN	1	5.83216
408	bj	NaN	1	0	3	3	1	5.84946
117	armax	3	0	1	NaN	NaN	1	5.85251
405	bj	NaN	1	0	3	2	1	5.89492
390	bj	NaN	1	0	2	1	1	5.94107
81	armax	1	2	1	NaN	NaN	1	6.02086
27	arx	3	0	NaN	NaN	NaN	1	6.07386

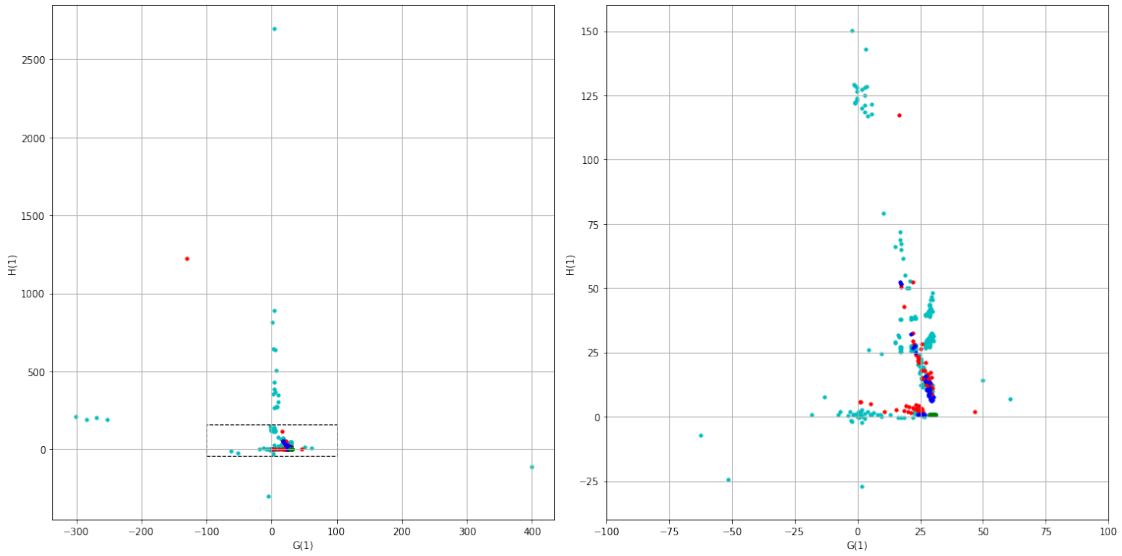
396	bj	NaN	1	0	2	3	1	6.09021
393	bj	NaN	1	0	2	2	1	6.15355
99	armax	2	1	1	NaN	NaN	1	6.23669
90	armax	2	0	1	NaN	NaN	1	6.24381
204	bj	NaN	0	0	2	3	1	6.32809
18	arx	2	0	NaN	NaN	NaN	1	6.38421
216	bj	NaN	0	0	3	3	1	6.69632
201	bj	NaN	0	0	2	2	1	6.71431
297	bj	NaN	0	2	2	2	1	6.84731
213	bj	NaN	0	0	3	2	1	6.96428
192	bj	NaN	0	0	1	3	1	7.02836
93	armax	2	0	2	NaN	NaN	1	7.30642
378	bj	NaN	1	0	1	1	1	7.39664
384	bj	NaN	1	0	1	3	1	7.40961
381	bj	NaN	1	0	1	2	1	7.41147
28	arx	3	0	NaN	NaN	NaN	2	7.49038
573	bj	NaN	2	0	1	2	1	7.54456
570	bj	NaN	2	0	1	1	1	7.5709
576	bj	NaN	2	0	1	3	1	7.58959
19	arx	2	0	NaN	NaN	NaN	2	7.64544



7.3 Display Static Gains Scatter

[ ]:
```
fig, ax = plt.subplots(1, 2, figsize=(16,8))

for i in range(0, 1+1):
```

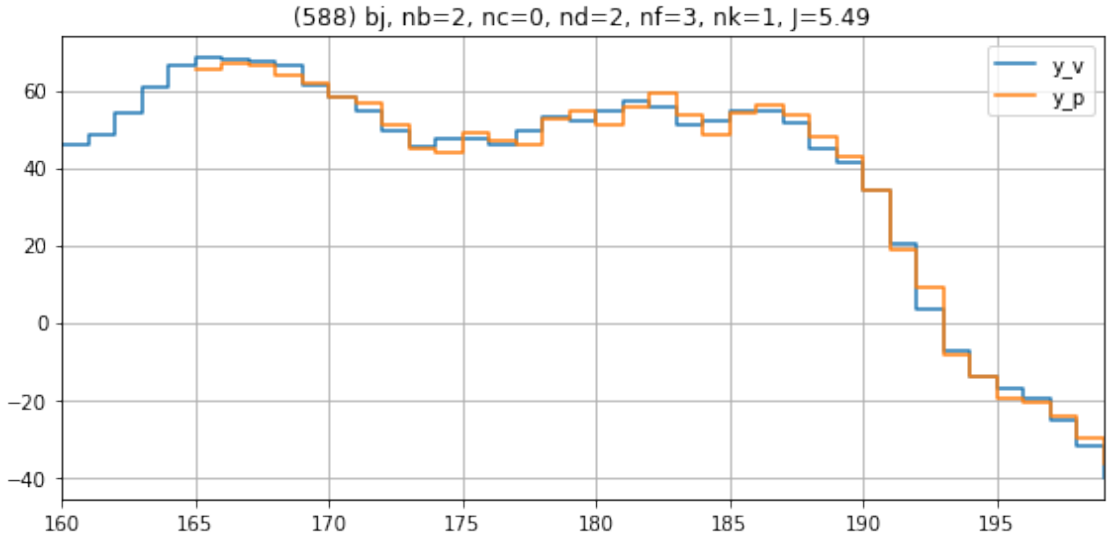
```
  ax[i].scatter(models.loc[models.model == 'bj'].kG,    models.loc[models.model␣
↪== 'bj'].kH,    s=10, color='c')
  ax[i].scatter(models.loc[models.model == 'armax'].kG, models.loc[models.model␣
↪== 'armax'].kH, s=10, color='r')
  ax[i].scatter(models.loc[models.model == 'arx'].kG,   models.loc[models.model␣
↪== 'arx'].kH,   s=10, color='b')
  ax[i].scatter(models.loc[models.model == 'oe'].kG,    models.loc[models.model␣
↪== 'oe'].kH,    s=10, color='g')
  ax[i].set_xlabel('G(1)')
  ax[i].set_ylabel('H(1)')
  ax[i].grid()

ax[0].add_patch(plt.Rectangle((-100,-40), 200, 200, fill=False, linestyle='--'))

ax[1].set_xlim((-100, 100))
ax[1].set_ylim((-40, 160))

plt.tight_layout()
plt.show()
```

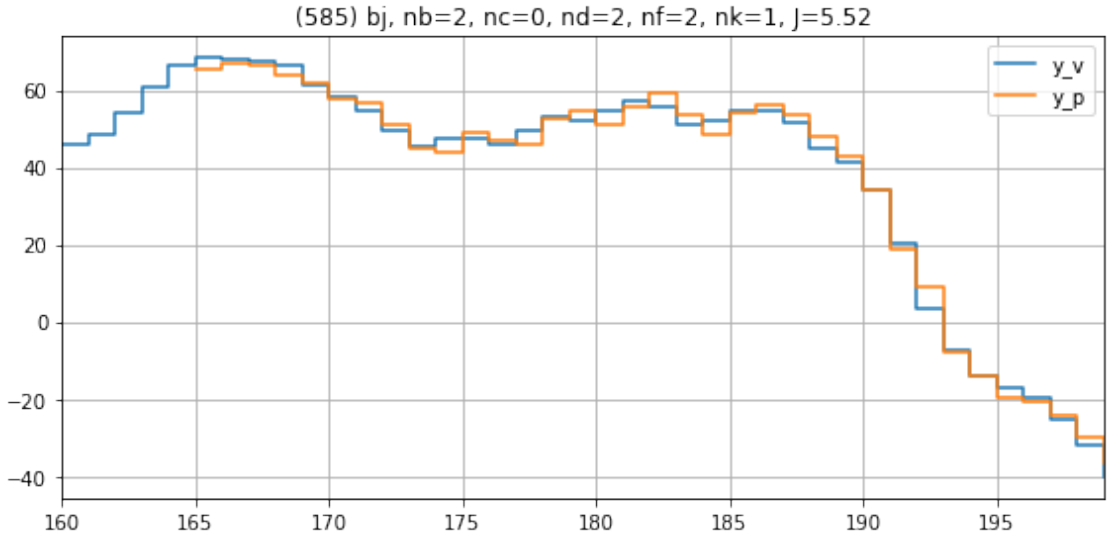


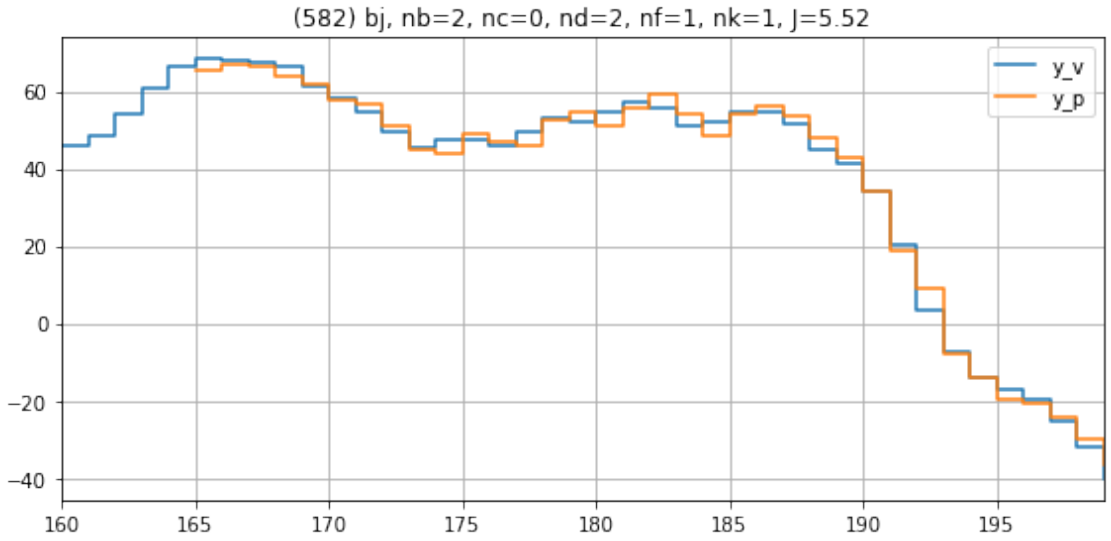
7.4 Display Predictions with Lowest Cost

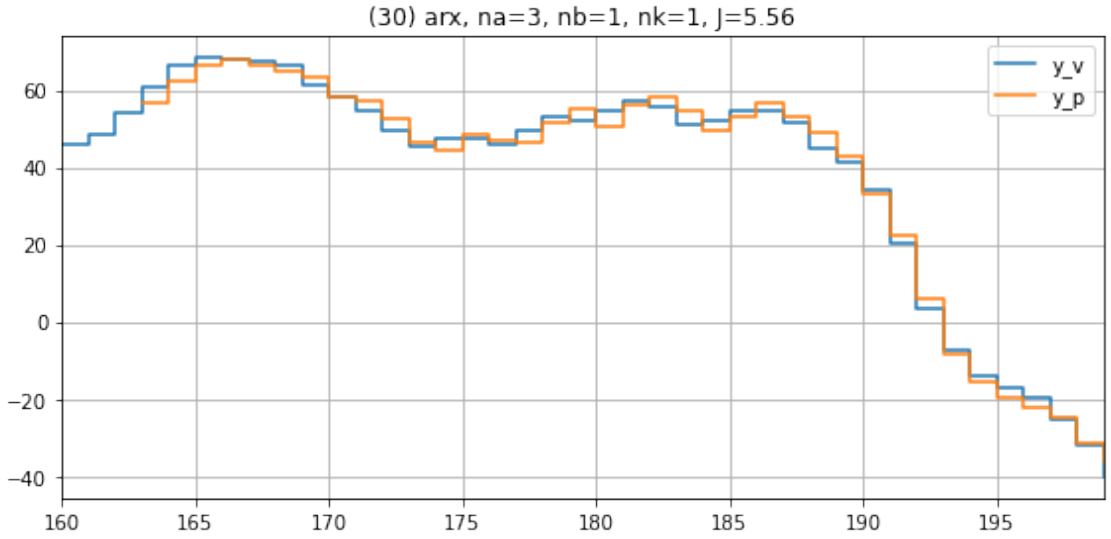
[ ]:
```
qty = 10
for i, (index, model) in enumerate(models.iterrows()):
  if i >= qty:
    break

  if np.isnan(model.yp).any():
```

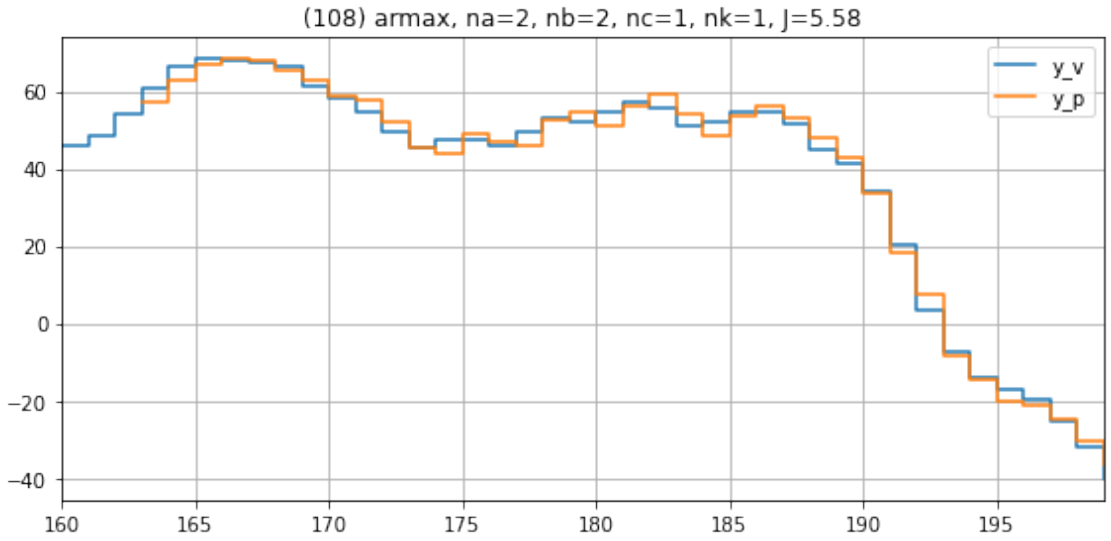
```
    continue

  if model.model == 'arx':
    title = f'({index}) {model.model}, na={model.na}, nb={model.nb}, nk={model.
↪nk}, J={model.Jp:.3g}'
  elif model.model == 'armax':
    title = f'({index}) {model.model}, na={model.na}, nb={model.nb}, nc={model.
↪nc}, nk={model.nk}, J={model.Jp:.3g}'
  elif model.model == 'oe':
    title = f'({index}) {model.model}, nb={model.nb}, nf={model.nf}, nk={model.
↪nk}, J={model.Jp:.3g}'
  elif model.model == 'bj':
    title = f'({index}) {model.model}, nb={model.nb}, nc={model.nc}, nd={model.
↪nd}, nf={model.nf}, nk={model.nk}, J={model.Jp:.3g}'
  else:
    assert(False)

  plt.figure(figsize=(8,4))
  plt.title(title)
  plt.plot(k_v, y_v, label='y_v', drawstyle='steps-post')
  plt.plot(k_v[int(model.delay):], model.yp, label='y_p',␣
↪drawstyle='steps-post')
  plt.xlim(k_v[0], k_v[-1])
  plt.grid()
  plt.legend()
  plt.tight_layout()
  plt.show()
```

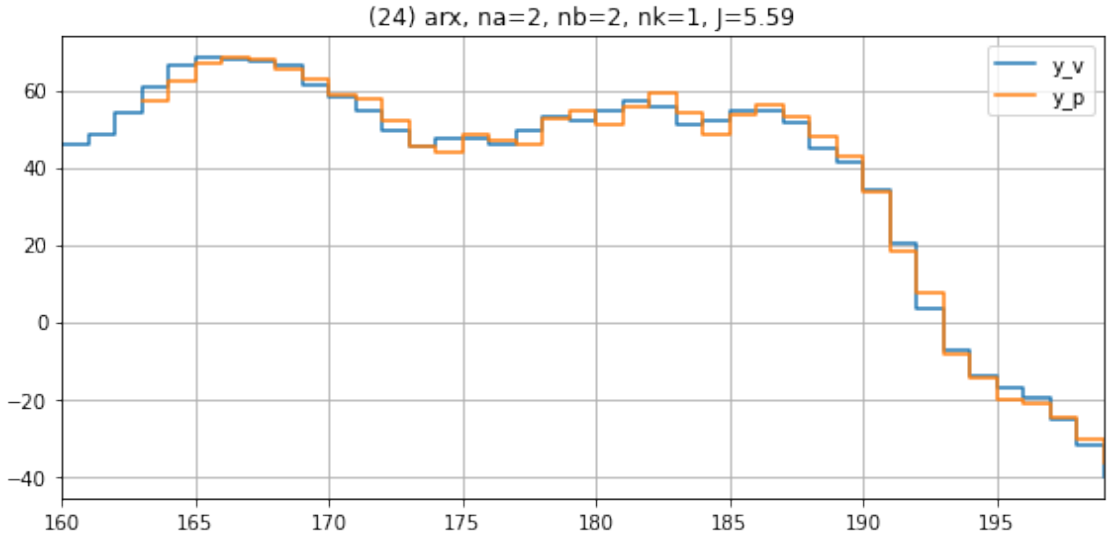


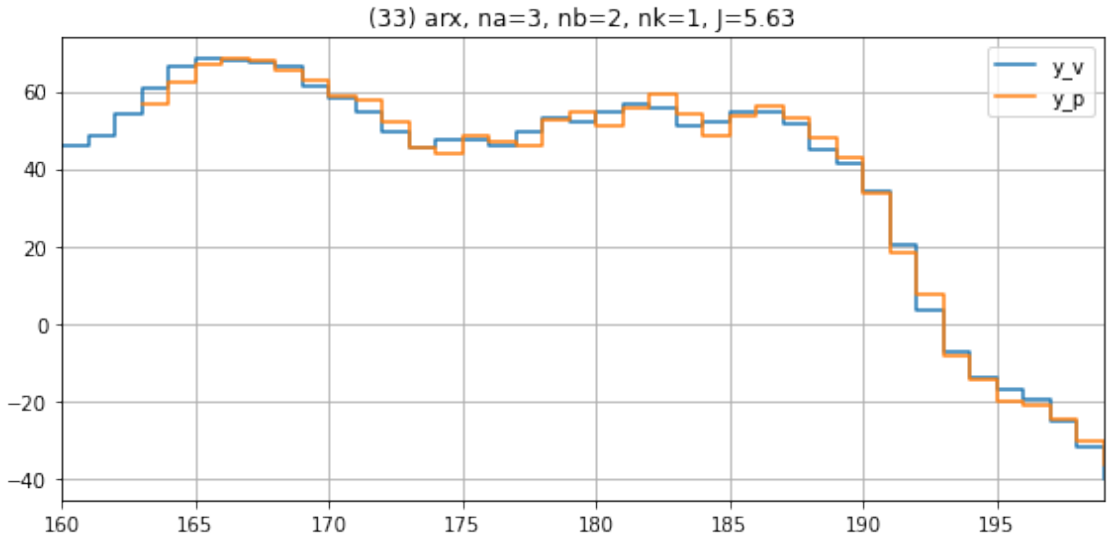

(585) bj, nb=2, nc=0, nd=2, nf=2, nk=1, J=5.52
y_v
y_p
60
40
20
0
−20
−40
160
165
170
175
180
185
190
195

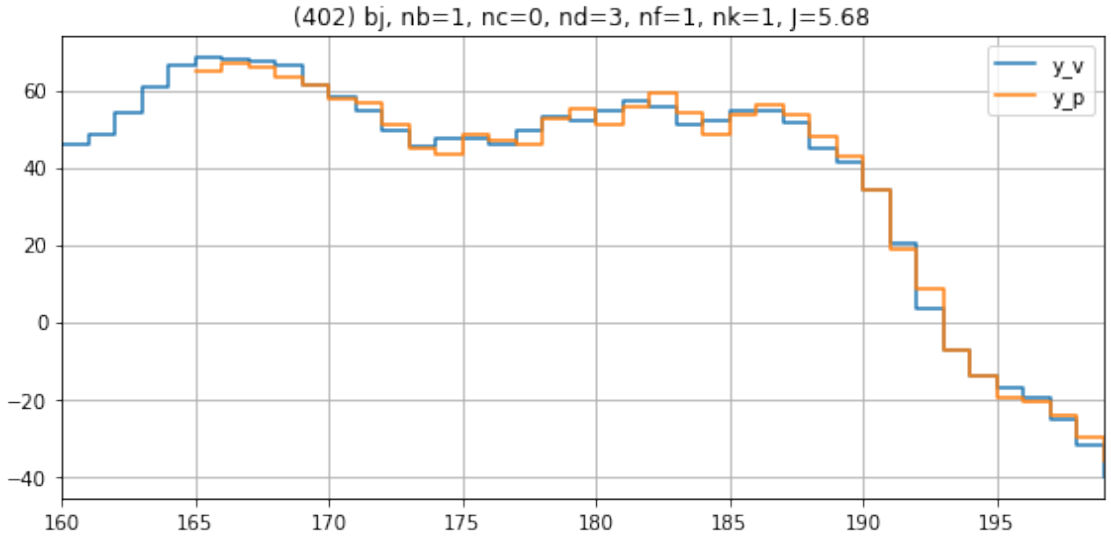

(582) bj, nb=2, nc=0, nd=2, nf=1, nk=1, J=5.52
y_v
y_p
60
40
20
0
−20
−40
160
165
170
175
180
185
190
195

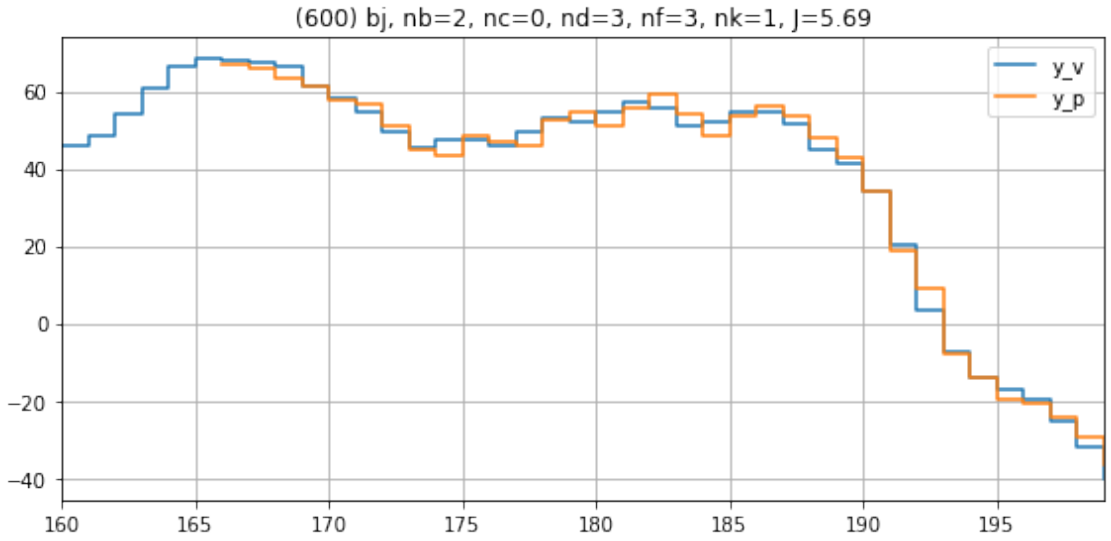

(30) arx, na=3, nb=1, nk=1, J=5.56
y_v
y_p
60
40
20
0
−20
−40
160
165
170
175
180
185
190
195

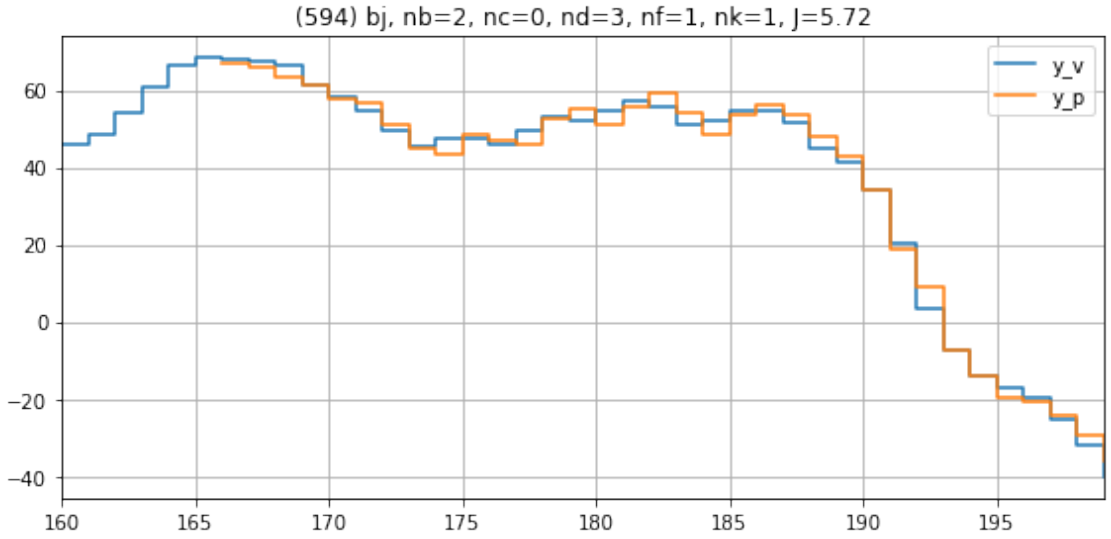

(108) armax, na=2, nb=2, nc=1, nk=1, J=5.58
y_v
y_p
60
40
20
0
−20
−40
160
165
170
175
180
185
190
195


(24) arx, na=2, nb=2, nk=1, J=5.59
y_v
y_p
60
40
20
0
−20
−40
160
165
170
175
180
185
190
195


(33) arx, na=3, nb=2, nk=1, J=5.63
y_v
y_p
60
40
20
0
−20
−40
160
165
170
175
180
185
190
195


(402) bj, nb=1, nc=0, nd=3, nf=1, nk=1, J=5.68
y_v
y_p
60
40
20
0
−20
−40
160
165
170
175
180
185
190
195


(600) bj, nb=2, nc=0, nd=3, nf=3, nk=1, J=5.69
y_v
y_p
60
40
20
0
−20
−40
160
165
170
175
180
185
190
195



7.5 Display Best ARX

[ ]:
```
with pd.option_context('display.precision', 5):
  display(models_disp.loc[models_disp.model == 'arx'][['na', 'nb', 'nk', 'Jp',␣
 ↪'A', 'B',]].head(10))
```

```
   na nb nk       Jp                                  A  \
30  3  1  1  5.55578  [ 1.     -1.3524  0.5032 -0.0458]
24  2  2  1  5.58895          [ 1.     -1.4068  0.4826]
33  3  2  1  5.63397  [ 1.     -1.3995  0.4627  0.0122]
21  2  1  1  5.80084          [ 1.     -1.3097  0.4174]
27  3  0  1  6.07386  [ 1.     -1.4147  0.5565 -0.051 ]
18  2  0  1  6.38421          [ 1.     -1.3708  0.4638]
28  3  0  2  7.49038  [ 1.     -1.3873  0.4881 -0.0085]
19  2  0  2  7.64544          [ 1.     -1.366   0.4622]
22  2  1  2  7.89001          [ 1.     -1.4641  0.5285]
31  3  1  2  8.02264  [ 1.     -1.4339  0.4481  0.0488]

                                      B
30             [0.     2.1432 0.8729]
24  [ 0.      2.1625  1.6108 -1.6016]
33  [ 0.      2.1558  1.6239 -1.6225]
21             [0.     2.1418 0.9413]
27                    [0.     2.6055]
18                    [0.     2.6559]
28             [0.     0.     2.5496]
19             [0.     0.     2.6626]
22  [ 0.      0.      3.2668 -1.5127]
31  [ 0.      0.      3.2988 -1.5975]
```

7.6 Display Best ARMAX

```
[ ]: with pd.option_context('display.precision', 5):
       display(models_disp.loc[models_disp.model == 'armax'][['na', 'nb', 'nc',␣
      ↪'nk', 'Jp', 'A', 'B', 'C']].head(10))
```

```
     na nb nc nk       Jp                                A  \
108   2  2  1  1   5.5785          [ 1.     -1.3952  0.4728]
126   3  1  1  1  5.81388  [ 1.     -0.7403 -0.3703  0.272 ]
135   3  2  1  1  5.83216  [ 1.     -1.2479  0.2488  0.0891]
117   3  0  1  1  5.85251  [ 1.     -1.5819  0.8218 -0.1578]
81    1  2  1  1  6.02086                    [ 1.   -0.84]
99    2  1  1  1  6.23669          [ 1.     -1.1588  0.2881]
90    2  0  1  1  6.24381          [ 1.     -1.3996  0.4888]
93    2  0  2  1  7.30642          [ 1.     -1.3664  0.4606]
120   3  0  2  1  7.81916  [ 1.     -1.5948  0.9398 -0.2549]
72    1  1  1  1  7.96677                  [ 1.     -0.8326]

                                      B                          C
108  [ 0.      2.1016  1.7255 -1.6017]            [1.     0.0216]
126             [0.     1.8562 2.7502]            [1.     0.5879]
135  [ 0.      2.0695  2.0321 -1.5339]            [1.     0.1593]
117                    [0.     2.3682]          [ 1.     -0.1876]
81   [ 0.      2.239   2.843  -0.3675]            [1.     0.4336]
99              [0.     1.9753 1.7543]            [1.     0.2029]
90                       [0.    2.537]          [ 1.     -0.0399]
93                     [0.     2.6759]  [ 1.     -0.0062  0.1153]
120                    [0.     2.6218]  [ 1.     -0.2164  0.2494]
72              [0.     2.1322 2.7844]             [1.    0.437]
```

7.7 Display Best OE

```
[ ]: with pd.option_context('display.precision', 5):
       display(models_disp.loc[models_disp.model == 'oe'][['nb', 'nf', 'nk', 'Jp',␣
      ↪'B', 'F',]].head(10))
```

```
     nb nf nk         Jp                                                B  \
166   2  2  2  167.74747          [ 0.      0.      7.7118 -6.9075  0.2866]
169   2  3  2  167.91915          [ 0.      0.      7.978  -3.2287 -3.063 ]
167   2  2  3  175.23633  [ 0.      0.      0.     10.2879 -9.6789  0.9381]
170   2  3  3   175.2679  [ 0.      0.      0.     10.5427 -5.0892 -3.0755]
159   1  3  1  181.59611                          [ 0.      3.3507 -2.7665]
165   2  2  1  184.14633                  [ 0.      3.5064 -0.0782 -2.6062]
163   2  1  2  187.49255          [ 0.      0.      8.083  -1.7842 -2.4118]
164   2  1  3  187.66964  [ 0.      0.      0.     10.6192 -4.0831 -2.7328]
157   1  2  2  193.48808                  [ 0.      0.      7.5631 -6.4238]
160   1  3  2  194.35584                  [ 0.      0.      7.8384 -6.694 ]
```

```
                                        F
166          [ 1.     -1.5244  0.5601]
169  [ 1.     -0.9518 -0.3204  0.3272]
167          [ 1.     -1.4015  0.4531]
170  [ 1.     -0.8704 -0.2848  0.2346]
159  [ 1.     -2.1778  1.6256 -0.429 ]
165          [ 1.     -1.6008  0.6273]
163                    [ 1.    -0.871]
164                    [ 1.    -0.872]
157          [ 1.     -1.5097  0.5469]
160  [ 1.     -1.4545  0.4468  0.045 ]
```

7.8 Display Best BJ

[ ]:
```
with pd.option_context('display.precision', 5):
  display(models_disp.loc[models_disp.model == 'bj'][['nb', 'nc', 'nd', 'nf',␣
 ↪'nk', 'Jp', 'B', 'C', 'D', 'F']].head(10))
```

```
    nb nc nd nf nk       Jp                                  B     C  \
588  2  0  2  3  1  5.48709  [ 0.      2.0629  2.5924 -0.6554]  [1.]
585  2  0  2  2  1   5.5212      [0.     2.0782 3.2191 0.2747]  [1.]
582  2  0  2  1  1  5.52389  [ 0.      2.0827  2.8744 -0.1977]  [1.]
402  1  0  3  1  1  5.68255             [0.     2.039  2.8255]  [1.]
600  2  0  3  3  1   5.6935  [ 0.      2.0368  2.5593 -0.6899]  [1.]
594  2  0  3  1  1    5.721  [ 0.      2.0651  2.8472 -0.1961]  [1.]
597  2  0  3  2  1  5.72537      [0.     2.0595 3.266  0.3694]  [1.]
408  1  0  3  3  1  5.84946             [0.     2.0456 3.023 ]  [1.]
405  1  0  3  2  1  5.89492             [0.     2.0645 3.002 ]  [1.]
390  1  0  2  1  1  5.94107             [0.     2.0522 2.8633]  [1.]

                                        D                                  F
588          [ 1.     -1.4404  0.5514]  [ 1.     -0.9704  0.0724  0.0341]
585          [ 1.     -1.4398  0.5513]          [ 1.     -0.6716 -0.139 ]
582          [ 1.     -1.4396  0.5513]                  [ 1.     -0.8382]
402  [ 1.     -1.4025  0.4521  0.0692]                  [ 1.     -0.8346]
600  [ 1.     -1.4028  0.4495  0.072 ]  [ 1.     -0.9766  0.0658  0.0435]
594  [ 1.     -1.4031  0.4531  0.0692]                     [ 1.    -0.84]
597  [ 1.     -1.4029  0.4519  0.0701]          [ 1.     -0.6367 -0.1701]
408  [ 1.     -1.4025  0.4502  0.0713]  [ 1.     -0.7514 -0.1005  0.024 ]
405  [ 1.     -1.4031  0.4527  0.0696]          [ 1.     -0.7657 -0.0624]
390          [ 1.     -1.4391  0.5502]                  [ 1.     -0.8325]
```

7.9 Display Model in Class

[ ]:
```
model = models.loc[(models.model == 'arx') & (models.na == 2) & (models.nb ==␣
 ↪2) & (models.nk == 1)]
assert(len(model) == 1)
model = model.iloc[0]
```

```
print('G =')
display(model.G)
print('H =')
display(model.H)
print('J_p =', model.Jp)

plt.figure(figsize=(8,4))
plt.plot(k_v, y_v, label='y_v', drawstyle='steps-post')
plt.plot(k_v[int(model.delay):], model.yp, label='y_p', drawstyle='steps-post')
plt.xlim(k_v[0], k_v[-1])
plt.grid()
plt.legend()
plt.tight_layout()
plt.show()
```

G =

$$\frac{2.162z^2 + 1.611z - 1.602}{z^3 - 1.407z^2 + 0.4826z}$$

H =

$$\frac{z^2}{z^2 - 1.407z + 0.4826}$$

J_p = 5.588946926532913