

## Python\functions.py

```
1 import numpy as np
2 import pandas as pd
3 import pysid
4 import control
5
6 def mean_squared_error(x, y):
7     return np.square(x - y).mean()
8
9 def transferFunction(num, den):
10     z = control.TransferFunction.z
11     num_z = 0
12     den_z = 0
13     for i, n in enumerate(num):
14         num_z += n * z**(-i)
15     for i, d in enumerate(den):
16         den_z += d * z**(-i)
17     return control.minreal(num_z/den_z, verbose=False)
18
19 def predict(u, y, G, H):
20     L_u = control.minreal(G/H, verbose=False)
21     L_y = control.minreal(1 - 1/H, verbose=False)
22
23     delay = int(max(len(L_u.den[0][0]), len(L_y.den[0][0])) - 1)
24     assert(delay ≥ 1)
25
26     y_u = control.forced_response(sys=L_u, U=u, return_x=False)[1]
27     y_y = control.forced_response(sys=L_y, U=y, return_x=False)[1]
28     assert(len(y_u) == len(y_y))
29
30     return y_u[delay:] + y_y[delay:], delay
31
32 def akaike(N, mse, m):
33     aic = N*np.log(mse) + 2*m
34     aic_c = aic + (2*m*(m+1))/(N - m - 1)
35     return aic, aic_c
36
37 def models_frame():
38     return pd.DataFrame(columns=[
39         'model',
40         'na', 'nb', 'nc', 'nd', 'nf', 'nk',
41         'Ji', 'Jv',
42         'AICi', 'AICCi', 'AICv', 'AICcv',
43         'A', 'B', 'C', 'D', 'F',
44         'G', 'H', 'zG', 'pG', 'kG', 'zH', 'pH', 'kH',
45         'yp', 'delay',
46         'ei', 'ev',
47     ])
48
49 def arx(u_i, y_i, u_v, y_v, na_range, nb_range, nk_range):
50     models = pd.DataFrame()
51     for na in na_range:
52         for nb in nb_range:
53             for nk in nk_range:
54                 id = pysid.arx(na=na, nb=nb, nk=nk, u=u_i, y=y_i)
55                 A = id.A[0][0]
56                 B = id.B[0][0]
```

```

57
58     assert(A[0] == 1)
59
60     G = transferFunction(B, A)
61     H = transferFunction([1], A)
62
63     y_p_i, delay_i = predict(u_i, y_i, G, H)
64     y_p_v, delay_v = predict(u_v, y_v, G, H)
65     assert(delay_i == delay_v)
66
67     J_p_i = mean_squared_error(y_i[delay_i:], y_p_i)
68     J_p_v = mean_squared_error(y_v[delay_v:], y_p_v)
69
70     aic_i, aicc_i = akaike(len(u_i), J_p_i, na + nb + 1)
71     aic_v, aicc_v = akaike(len(u_v), J_p_v, na + nb + 1)
72
73     e_i = y_i[delay_i:] - y_p_i
74     e_v = y_v[delay_v:] - y_p_v
75
76     models = pd.concat([models, pd.DataFrame({
77         'model': 'ARX',
78         'na': [na],
79         'nb': [nb],
80         'nk': [nk],
81         'A': [A],
82         'B': [B],
83         'G': [G],
84         'zG': [G.zeros()],
85         'pG': [G.poles()],
86         'kG': [G.dcgain()],
87         'H': [H],
88         'zH': [H.zeros()],
89         'pH': [H.poles()],
90         'kH': [H.dcgain()],
91         'yp': [y_p_v],
92         'Ji': [J_p_i],
93         'Jv': [J_p_v],
94         'AICi': [aic_i],
95         'AICCi': [aicc_i],
96         'AICv': [aic_v],
97         'AICcv': [aicc_v],
98         'delay': [delay_v],
99         'ei': [e_i],
100        'ev': [e_v],
101    })], ignore_index=True)
102
103     return models
104
105 def armax(u_i, y_i, u_v, y_v, na_range, nb_range, nc_range, nk_range):
106     models = pd.DataFrame()
107     for na in na_range:
108         for nb in nb_range:
109             for nc in nc_range:
110                 for nk in nk_range:
111                     id = pysid.armax(na=na, nb=nb, nc=nc, nk=nk, u=u_i, y=y_i)
112                     A = id.A[0][0]
113                     B = id.B[0][0]
114                     C = id.C[0]
115
116                     assert(A[0] == 1)

```

```

117     assert(C[0] == 1)
118
119     G = transferFunction(B, A)
120     H = transferFunction(C, A)
121
122     y_p_i, delay_i = predict(u_i, y_i, G, H)
123     y_p_v, delay_v = predict(u_v, y_v, G, H)
124     assert(delay_i == delay_v)
125
126     J_p_i = mean_squared_error(y_i[delay_i:], y_p_i)
127     J_p_v = mean_squared_error(y_v[delay_v:], y_p_v)
128
129     aic_i, aicc_i = akaike(len(u_i), J_p_i, na + nb + 1 + nc)
130     aic_v, aicc_v = akaike(len(u_v), J_p_v, na + nb + 1 + nc)
131
132     e_i = y_i[delay_i:] - y_p_i
133     e_v = y_v[delay_v:] - y_p_v
134
135     models = pd.concat([models, pd.DataFrame({
136         'model': 'ARMAX',
137         'na': [na],
138         'nb': [nb],
139         'nc': [nc],
140         'nk': [nk],
141         'A': [A],
142         'B': [B],
143         'C': [C],
144         'G': [G],
145         'zG': [G.zeros()],
146         'pG': [G.poles()],
147         'kG': [G.dcgain()],
148         'H': [H],
149         'zH': [H.zeros()],
150         'pH': [H.poles()],
151         'kH': [H.dcgain()],
152         'yp': [y_p_v],
153         'Ji': [J_p_i],
154         'Jv': [J_p_v],
155         'AICi': [aic_i],
156         'AICCi': [aicc_i],
157         'AICv': [aic_v],
158         'AICcv': [aicc_v],
159         'delay': [delay_v],
160         'ei': [e_i],
161         'ev': [e_v],
162     })], ignore_index=True)
163
164     return models
165
166 def oe(u_i, y_i, u_v, y_v, nb_range, nf_range, nk_range):
167     models = pd.DataFrame()
168     for nb in nb_range:
169         for nf in nf_range:
170             for nk in nk_range:
171                 id = pysid.oe(nb=nb, nf=nf, nk=nk, u=u_i, y=y_i)
172                 B = id.B[0][0]
173                 F = id.F[0][0]
174
175                 assert(F[0] == 1)
176

```

```

177     G = transferFunction(B, F)
178     H = transferFunction([1], [1])
179
180     y_p_i, delay_i = predict(u_i, y_i, G, H)
181     y_p_v, delay_v = predict(u_v, y_v, G, H)
182     assert(delay_i == delay_v)
183
184     J_p_i = mean_squared_error(y_i[delay_i:], y_p_i)
185     J_p_v = mean_squared_error(y_v[delay_v:], y_p_v)
186
187     aic_i, aicc_i = akaike(len(u_i), J_p_i, nb + 1 + nf)
188     aic_v, aicc_v = akaike(len(u_v), J_p_v, nb + 1 + nf)
189
190     e_i = y_i[delay_i:] - y_p_i
191     e_v = y_v[delay_v:] - y_p_v
192
193     models = pd.concat([models, pd.DataFrame({
194         'model': 'OE',
195         'nb': [nb],
196         'nf': [nf],
197         'nk': [nk],
198         'B': [B],
199         'F': [F],
200         'G': [G],
201         'zG': [G.zeros()],
202         'pG': [G.poles()],
203         'kG': [G.dcgain()],
204         'H': [H],
205         'zH': [H.zeros()],
206         'pH': [H.poles()],
207         'kH': [H.dcgain()],
208         'yp': [y_p_v],
209         'Ji': [J_p_i],
210         'Jv': [J_p_v],
211         'AICi': [aic_i],
212         'AICCi': [aicc_i],
213         'AICv': [aic_v],
214         'AICcv': [aicc_v],
215         'delay': [delay_v],
216         'ei': [e_i],
217         'ev': [e_v],
218     })], ignore_index=True)
219
220     return models
221
222 def bj(u_i, y_i, u_v, y_v, nb_range, nc_range, nd_range, nf_range, nk_range):
223     models = pd.DataFrame()
224     for nb in nb_range:
225         for nc in nc_range:
226             for nd in nd_range:
227                 for nf in nf_range:
228                     for nk in nk_range:
229                         try:
230                             id = pysid.bj(nb=nb, nc=nc, nd=nd, nf=nf, nk=nk, u=u_i, y=y_i)
231                             B = id.B[0][0]
232                             C = id.C[0]
233                             D = id.D[0]
234                             F = id.F[0][0]
235
236                             assert(C[0] == 1)

```

```

237     assert(D[0] == 1)
238     assert(F[0] == 1)
239
240     G = transferFunction(B, F)
241     H = transferFunction(C, D)
242
243     y_p_i, delay_i = predict(u_i, y_i, G, H)
244     y_p_v, delay_v = predict(u_v, y_v, G, H)
245     assert(delay_i == delay_v)
246
247     J_p_i = mean_squared_error(y_i[delay_i:], y_p_i)
248     J_p_v = mean_squared_error(y_v[delay_v:], y_p_v)
249
250     aic_i, aicc_i = akaike(len(u_i), J_p_i, nb + 1 + nc + nd + nf)
251     aic_v, aicc_v = akaike(len(u_v), J_p_v, nb + 1 + nc + nd + nf)
252
253     e_i = y_i[delay_i:] - y_p_i
254     e_v = y_v[delay_v:] - y_p_v
255
256     models = pd.concat([models, pd.DataFrame({
257         'model': 'BJ',
258         'nb': [nb],
259         'nc': [nc],
260         'nd': [nd],
261         'nf': [nf],
262         'nk': [nk],
263         'B': [B],
264         'C': [C],
265         'D': [D],
266         'F': [F],
267         'G': [G],
268         'zG': [G.zeros()],
269         'pG': [G.poles()],
270         'kG': [G.dcgain()],
271         'H': [H],
272         'zH': [H.zeros()],
273         'pH': [H.poles()],
274         'kH': [H.dcgain()],
275         'yp': [y_p_v],
276         'Ji': [J_p_i],
277         'Jv': [J_p_v],
278         'AICi': [aic_i],
279         'AICCi': [aicc_i],
280         'AICv': [aic_v],
281         'AICcv': [aicc_v],
282         'delay': [delay_v],
283         'ei': [e_i],
284         'ev': [e_v],
285     })])
286 except Exception as e:
287     # display(str(e))
288     models = pd.concat([models, pd.DataFrame({
289         'model': 'bj',
290         'nb': [nb],
291         'nc': [nc],
292         'nd': [nd],
293         'nf': [nf],
294         'nk': [nk],
295     })], ignore_index=True)
296

```

```
297     return models
298
299 def display_models(df, columns, precision, qty):
300     df = df.copy() # probably unnecessary, but safety first
301
302     with np.printoptions(precision=precision):
303         for collumn in ['A', 'B', 'C', 'D', 'F', 'zG', 'pG', 'zH', 'pH']:
304             if collumn in df:
305                 df[collumn] = df[collumn].astype(str)
306
307     with pd.option_context('display.precision', precision):
308         display(df[columns].fillna('-').head(qty))
309
```