Python\functions.py

```python
import numpy as np
import pandas as pd
import pysid
import control

def mean_squared_error(x, y):
  return np.square(x - y).mean()

def transferFunction(num, den):
  z = control.TransferFunction.z
  num_z = 0
  den_z = 0
  for i, n in enumerate(num):
    num_z += n * z**(-i)
  for i, d in enumerate(den):
    den_z += d * z**(-i)
  return control.minreal(num_z/den_z, verbose=False)

def predict(u, y, G, H):
  L_u = control.minreal(G/H,     verbose=False)
  L_y = control.minreal(1 - 1/H, verbose=False)

  delay = int(max(len(L_u.den[0][0]), len(L_y.den[0][0])) - 1)
  assert(delay >= 1)

  y_u = control.forced_response(sys=L_u, U=u, return_x=False)[1]
  y_y = control.forced_response(sys=L_y, U=y, return_x=False)[1]
  assert(len(y_u) == len(y_y))

  return y_u[delay:] + y_y[delay:], delay

def models_frame():
  return pd.DataFrame(columns=
['model','na','nb','nc','nd','nf','nk','Jp','A','B','C','D','F','G','H','zG','pG',

def arx(u_i, y_i, u_v, y_v, na_range, nb_range, nk_range):
  models = pd.DataFrame()
  for na in na_range:
    for nb in nb_range:
      for nk in nk_range:
        id = pysid.arx(na=na, nb=nb, nk=nk, u=u_i, y=y_i)
        A = id.A[0][0]
        B = id.B[0][0]

        assert(A[0] == 1)

        G = transferFunction(B, A)
        H = transferFunction([1], A)

        y_p, delay = predict(u_v, y_v, G, H)
        J_p = mean_squared_error(y_v[delay:], y_p)

        models = pd.concat([models, pd.DataFrame({
          'model': 'arx',
          'na': [na],
          'nb': [nb],
```

```
            'nk': [nk],
            'A': [A],
            'B': [B],
            'G': [G],
            'zG': [G.zeros()],
            'pG': [G.poles()],
            'kG': [G.dcgain()],
            'H': [H],
            'zH': [H.zeros()],
            'pH': [H.poles()],
            'kH': [H.dcgain()],
            'yp': [y_p],
            'Jp': [J_p],
            'delay': [delay],
          })])

  return models

def armax(u_i, y_i, u_v, y_v, na_range, nb_range, nc_range, nk_range):
  models = pd.DataFrame()
  for na in na_range:
    for nb in nb_range:
      for nc in nc_range:
        for nk in nk_range:
          id = pysid.armax(na=na, nb=nb, nc=nc, nk=nk, u=u_i, y=y_i)
          A = id.A[0][0]
          B = id.B[0][0]
          C = id.C[0]

          assert(A[0] == 1)
          assert(C[0] == 1)

          G = transferFunction(B, A)
          H = transferFunction(C, A)

          y_p, delay = predict(u_v, y_v, G, H)
          J_p = mean_squared_error(y_v[delay:], y_p)

          models = pd.concat([models, pd.DataFrame({
            'model': 'armax',
            'na': [na],
            'nb': [nb],
            'nc': [nc],
            'nk': [nk],
            'A': [A],
            'B': [B],
            'C': [C],
            'G': [G],
            'zG': [G.zeros()],
            'pG': [G.poles()],
            'kG': [G.dcgain()],
            'H': [H],
            'zH': [H.zeros()],
            'pH': [H.poles()],
            'kH': [H.dcgain()],
            'yp': [y_p],
            'Jp': [J_p],
            'delay': [delay],
          })])

```

```
116   return models
117
118 def oe(u_i, y_i, u_v, y_v, nb_range, nf_range, nk_range):
119   models = pd.DataFrame()
120   for nb in nb_range:
121     for nf in nf_range:
122       for nk in nk_range:
123         id = pysid.oe(nb=nb, nf=nf, nk=nk, u=u_i, y=y_i)
124         B = id.B[0][0]
125         F = id.F[0][0]
126
127         assert(F[0] == 1)
128
129         G = transferFunction(B, F)
130         H = transferFunction([1], [1])
131
132         y_p, delay = predict(u_v, y_v, G, H)
133         J_p = mean_squared_error(y_v[delay:], y_p)
134
135         models = pd.concat([models, pd.DataFrame({
136           'model': 'oe',
137           'nb': [nb],
138           'nf': [nf],
139           'nk': [nk],
140           'B': [B],
141           'F': [F],
142           'G': [G],
143           'zG': [G.zeros()],
144           'pG': [G.poles()],
145           'kG': [G.dcgain()],
146           'H': [H],
147           'zH': [H.zeros()],
148           'pH': [H.poles()],
149           'kH': [H.dcgain()],
150           'yp': [y_p],
151           'Jp': [J_p],
152           'delay': [delay],
153         })])
154
155   return models
156
157 def bj(u_i, y_i, u_v, y_v, nb_range, nc_range, nd_range, nf_range, nk_range):
158   models = pd.DataFrame()
159   for nb in nb_range:
160     for nc in nc_range:
161       for nd in nd_range:
162         for nf in nf_range:
163           for nk in nk_range:
164             try:
165               id = pysid.bj(nb=nb, nc=nc, nd=nd, nf=nf, nk=nk, u=u_i, y=y_i)
166               B = id.B[0][0]
167               C = id.C[0]
168               D = id.D[0]
169               F = id.F[0][0]
170
171               assert(C[0] == 1)
172               assert(D[0] == 1)
173               assert(F[0] == 1)
174
175               G = transferFunction(B, F)
```

```
176                 H = transferFunction(C, D)
177 
178                 y_p, delay = predict(u_v, y_v, G, H)
179                 J_p = mean_squared_error(y_v[delay:], y_p)
180 
181                 models = pd.concat([models, pd.DataFrame({
182                   'model': 'bj',
183                   'nb': [nb],
184                   'nc': [nc],
185                   'nd': [nd],
186                   'nf': [nf],
187                   'nk': [nk],
188                   'B': [B],
189                   'C': [C],
190                   'D': [D],
191                   'F': [F],
192                   'G': [G],
193                   'zG': [G.zeros()],
194                   'pG': [G.poles()],
195                   'kG': [G.dcgain()],
196                   'H': [H],
197                   'zH': [H.zeros()],
198                   'pH': [H.poles()],
199                   'kH': [H.dcgain()],
200                   'yp': [y_p],
201                   'Jp': [J_p],
202                   'delay': [delay]
203                 })])
204               except Exception as e:
205                 # display(str(e))
206                 models = pd.concat([models, pd.DataFrame({
207                   'model': 'bj',
208                   'nb': [nb],
209                   'nc': [nc],
210                   'nd': [nd],
211                   'nf': [nf],
212                   'nk': [nk],
213                 })])
214 
215     return models
216 
```