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 akaike(N, mse, m):
  aic = N*np.log(mse) + 2*m
  aic_c = aic + (2*m*(m+1))/(N - m - 1)
  return aic, aic_c

def models_frame():
  return pd.DataFrame(columns=[
    'model',
    'na','nb','nc','nd','nf','nk',
    'Ji','Jv',
    'AICi','AICCi','AICv','AICCv',
    'A','B','C','D','F',
    'G','H','zG','pG','kG','zH','pH','kH',
    'yp','delay',
    'ei', 'ev',
  ])

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]
```

```
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)
```

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

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

            y_p_i, delay_i = predict(u_i, y_i, G, H)
            y_p_v, delay_v = predict(u_v, y_v, G, H)
            assert(delay_i == delay_v)

            J_p_i = mean_squared_error(y_i[delay_i:], y_p_i)
            J_p_v = mean_squared_error(y_v[delay_v:], y_p_v)

            aic_i, aicc_i = akaike(len(u_i), J_p_i, na + nb + 1 + nc)
            aic_v, aicc_v = akaike(len(u_v), J_p_v, na + nb + 1 + nc)

            e_i = y_i[delay_i:] - y_p_i
            e_v = y_v[delay_v:] - y_p_v

            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_v],
              'Ji': [J_p_i],
              'Jv': [J_p_v],
              'AICi': [aic_i],
              'AICCi': [aicc_i],
              'AICv': [aic_v],
              'AICCv': [aicc_v],
              'delay': [delay_v],
              'ei': [e_i],
              'ev': [e_v],
            })], ignore_index=True)

  return models

def oe(u_i, y_i, u_v, y_v, nb_range, nf_range, nk_range):
  models = pd.DataFrame()
  for nb in nb_range:
    for nf in nf_range:
      for nk in nk_range:
        id = pysid.oe(nb=nb, nf=nf, nk=nk, u=u_i, y=y_i)
        B = id.B[0][0]
        F = id.F[0][0]

        assert(F[0] == 1)

```

```
        G = transferFunction(B, F)
        H = transferFunction([1], [1])

        y_p_i, delay_i = predict(u_i, y_i, G, H)
        y_p_v, delay_v = predict(u_v, y_v, G, H)
        assert(delay_i == delay_v)

        J_p_i = mean_squared_error(y_i[delay_i:], y_p_i)
        J_p_v = mean_squared_error(y_v[delay_v:], y_p_v)

        aic_i, aicc_i = akaike(len(u_i), J_p_i, nb + 1 + nf)
        aic_v, aicc_v = akaike(len(u_v), J_p_v, nb + 1 + nf)

        e_i = y_i[delay_i:] - y_p_i
        e_v = y_v[delay_v:] - y_p_v

        models = pd.concat([models, pd.DataFrame({
          'model': 'OE',
          'nb': [nb],
          'nf': [nf],
          'nk': [nk],
          'B': [B],
          'F': [F],
          '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_v],
          'Ji': [J_p_i],
          'Jv': [J_p_v],
          'AICi': [aic_i],
          'AICCi': [aicc_i],
          'AICv': [aic_v],
          'AICCv': [aicc_v],
          'delay': [delay_v],
          'ei': [e_i],
          'ev': [e_v],
        })], ignore_index=True)

  return models

def bj(u_i, y_i, u_v, y_v, nb_range, nc_range, nd_range, nf_range, nk_range):
  models = pd.DataFrame()
  for nb in nb_range:
    for nc in nc_range:
      for nd in nd_range:
        for nf in nf_range:
          for nk in nk_range:
            try:
              id = pysid.bj(nb=nb, nc=nc, nd=nd, nf=nf, nk=nk, u=u_i, y=y_i)
              B = id.B[0][0]
              C = id.C[0]
              D = id.D[0]
              F = id.F[0][0]

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

```
            assert(D[0] == 1)
            assert(F[0] == 1)

            G = transferFunction(B, F)
            H = transferFunction(C, D)

            y_p_i, delay_i = predict(u_i, y_i, G, H)
            y_p_v, delay_v = predict(u_v, y_v, G, H)
            assert(delay_i == delay_v)

            J_p_i = mean_squared_error(y_i[delay_i:], y_p_i)
            J_p_v = mean_squared_error(y_v[delay_v:], y_p_v)

            aic_i, aicc_i = akaike(len(u_i), J_p_i, nb + 1 + nc + nd + nf)
            aic_v, aicc_v = akaike(len(u_v), J_p_v, nb + 1 + nc + nd + nf)

            e_i = y_i[delay_i:] - y_p_i
            e_v = y_v[delay_v:] - y_p_v

            models = pd.concat([models, pd.DataFrame({
              'model': 'BJ',
              'nb': [nb],
              'nc': [nc],
              'nd': [nd],
              'nf': [nf],
              'nk': [nk],
              'B': [B],
              'C': [C],
              'D': [D],
              'F': [F],
              '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_v],
              'Ji': [J_p_i],
              'Jv': [J_p_v],
              'AICi': [aic_i],
              'AICCi': [aicc_i],
              'AICv': [aic_v],
              'AICCv': [aicc_v],
              'delay': [delay_v],
              'ei': [e_i],
              'ev': [e_v],
            })])
          except Exception as e:
            # display(str(e))
            models = pd.concat([models, pd.DataFrame({
              'model': 'bj',
              'nb': [nb],
              'nc': [nc],
              'nd': [nd],
              'nf': [nf],
              'nk': [nk],
            })], ignore_index=True)

```

```
  return models

def display_models(df, columns, precision, qty):
  df = df.copy() # probably unnecessary, but safety first

  with np.printoptions(precision=precision):
    for collumn in ['A', 'B', 'C', 'D', 'F', 'zG', 'pG', 'zH', 'pH']:
      if collumn in df:
        df[collumn] = df[collumn].astype(str)

  with pd.option_context('display.precision', precision):
    display(df[columns].fillna('-').head(qty))

```