

Stress - strain

```
In [1]: from scipy.optimize import root
from numpy import sign
import numpy as np
import matplotlib.pyplot as plt

def Y_F(x):
    a, eps_0, n = 200e6, 0.0003, 0.2
    return a*(eps_0+x)**n

E = 2e9

def simulate(eps,E,Y_F):
    sig = np.zeros_like(eps)
    _lambda = np.zeros_like(eps)
    Y = np.ones_like(eps)*Y_F(0)

    for i in range(eps.shape[0]-1):
        deps = (eps[i+1] - eps[i])
        if abs(sig[i] + E*deps) - Y[i] <= 0:
            dlambd = 0
        else:
            def eq(dlambd):
                ret = abs(sig[i] + E*deps)
                ret -= E*dlambd
                ret -= Y_F(_lambda[i] + dlambd)
                return ret
            dlambd = root(eq,1e-8).x

        _lambda[i+1] = _lambda[i] + dlambd
        Y[i+1] = Y_F(_lambda[i+1])
        sig[i+1] = sig[i] + E*(deps - dlambd*sign(sig[i] + E*deps))

    return sig
```

$$u_k = \sum_{n=1}^N \frac{a_k}{n^p} \cdot \sin\left(\frac{2\pi \cdot n \cdot k}{N}\right), \quad k \in \overline{1, T}$$

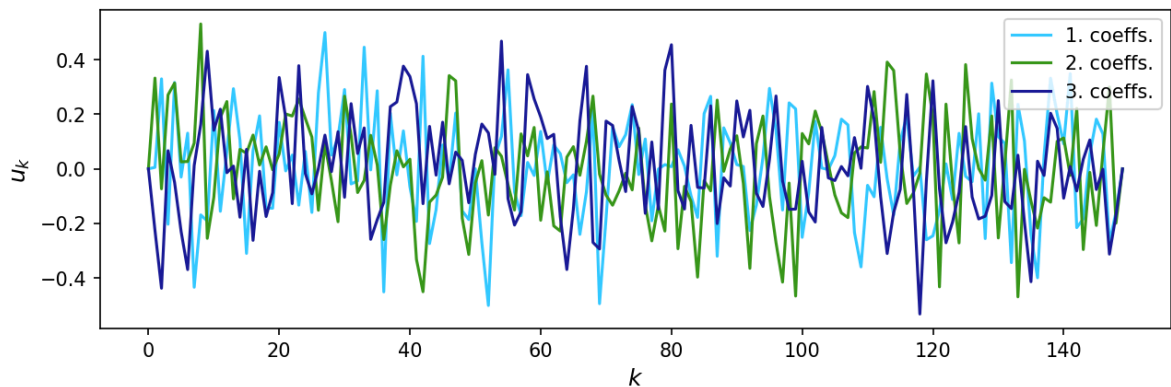
```
In [11]: def generate_random_signal(T,N,p=1.5,u_max=0.2):
    k = np.linspace(0, T, T)
    a = np.random.randn(N)
    u = np.zeros_like(k)
    for n in range(1, N + 1):
        u += (a[n - 1] / n**p) * np.sin(2*np.pi*n*k/N)
    u = np.array(u)
```

```
u = u/(np.max(u) - np.min(u))*u_max
return u
```

```
In [12]: plt.figure(figsize=(10,3),dpi=150)

colors = ['#33caff','#39971a','#1a1d97']

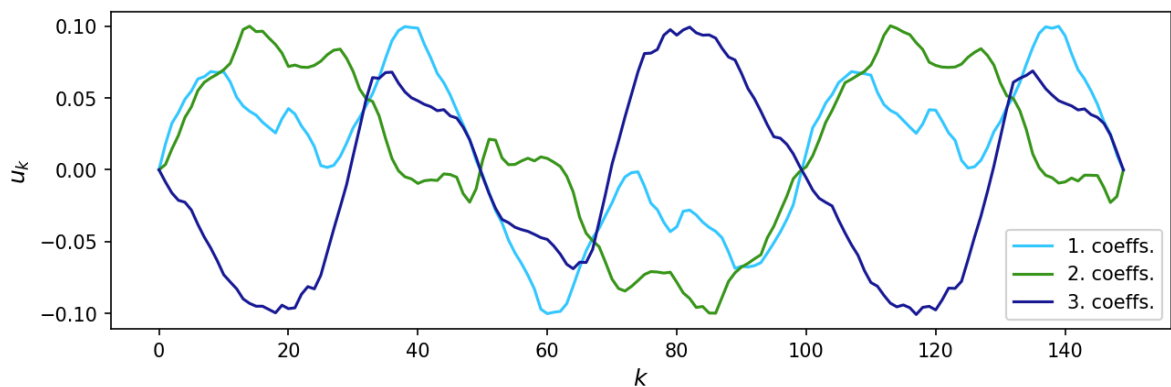
for i,color in enumerate(colors):
    u = generate_random_signal(T=150,N=300,p=0,u_max=1)
    plt.plot(u,label=f'{i+1}. coeffs.',color=color)
plt.ylabel(r'$u_k$',fontsize=12)
plt.xlabel(r'$k$',fontsize=12)
plt.legend()
plt.show()
```



```
In [13]: plt.figure(figsize=(10,3),dpi=150)

colors = ['#33caff','#39971a','#1a1d97']

for i,color in enumerate(colors):
    u = generate_random_signal(T=150,N=100,p=1.7,u_max=0.2)
    plt.plot(u,label=f'{i+1}. coeffs.',color=color)
plt.ylabel(r'$u_k$',fontsize=12)
plt.xlabel(r'$k$',fontsize=12)
plt.legend()
plt.show()
```



```
In [15]: # Simulation props
T = 150
n_samples = 200
```

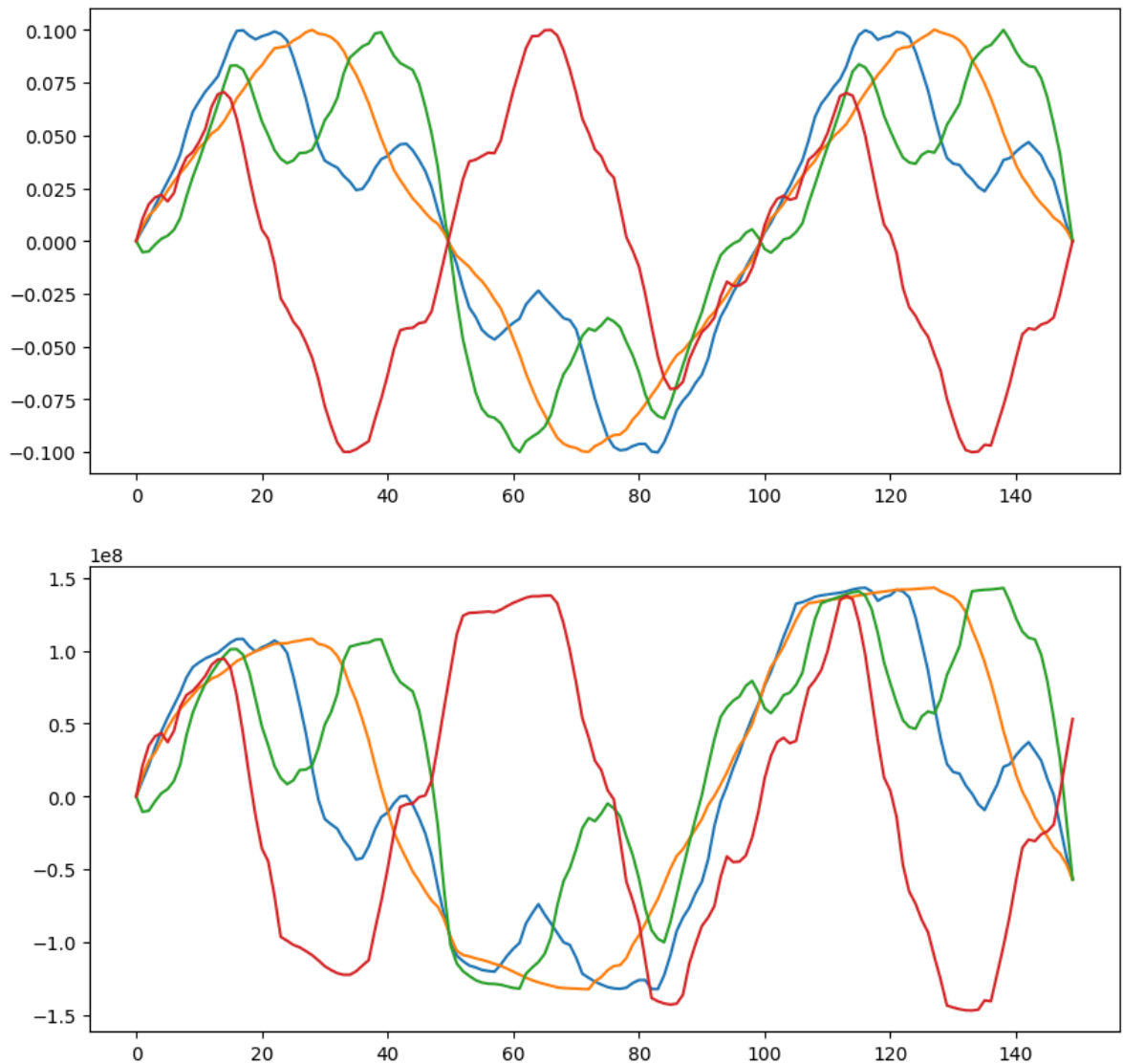
```
# Data gen props
N = 100
p = 2
u_max = 0.2

y_list, u_list = [], []
for _ in range(n_samples):
    u = generate_random_signal(T, N, p, u_max)
    y = simulate(u, E, Y_F)
    y_list += [y]
    u_list += [u]

y_list, u_list = np.array(y_list), np.array(u_list)
y_list.shape
```

Out[15]: (200, 150)

```
In [19]: fig, ax = plt.subplots(2, figsize=(10, 10))
for y, u in list(zip(y_list, u_list))[:4]:
    ax[0].plot(u)
    ax[1].plot(y)
plt.show()
```



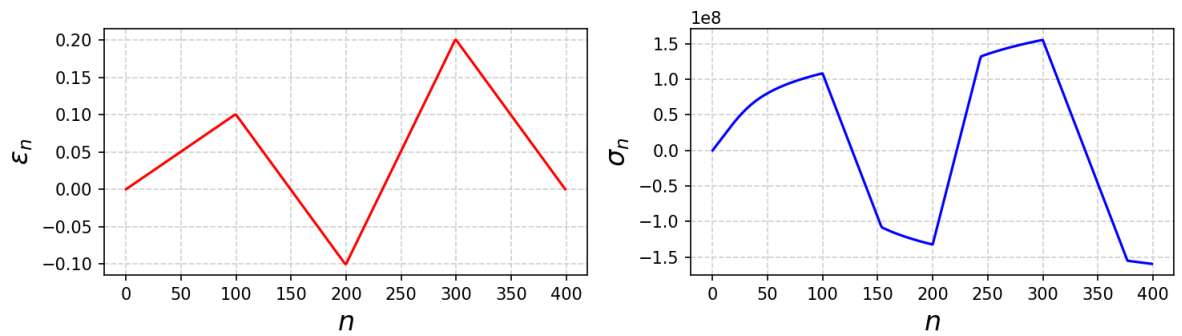
```
In [20]: np.save('y_list.npy',y_list)
         np.save('u_list.npy',u_list)
```

Benchmark (totally different)

```
In [6]: u = np.concatenate([np.linspace(0,0.1,100),np.linspace(0.1,-0.1,100),np.linspace(-0.1,0,100)])
        y = simulate(u,E,Y_F)

fig, ax = plt.subplots(1,2,figsize=(10,3),dpi=150,sharex=True)
ax[0].plot(u,c='red')
ax[1].plot(y,c='blue')
ax[0].grid(ls='--',c='lightgray')
ax[1].grid(ls='--',c='lightgray')
ax[0].set_xlabel(r'$n$', fontsize=16)
ax[1].set_xlabel(r'$n$', fontsize=16)
ax[0].set_ylabel(r'$\varepsilon_n$', fontsize=16)
ax[1].set_ylabel(r'$\sigma_n$', fontsize=16)
```

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



```
In [23]: np.save('y_benchmark.npy', y)  
         np.save('u_benchmark.npy', u)
```

0000\MLP\dir\model.py

```

1  import numpy as np
2  import os
3  from keras.models import Sequential
4  from keras.layers import Dense
5  import json
6  import keras
7  from keras.optimizers import Adam
8
9  class MLP_Direct:
10
11     data_path = 'data'
12
13     def __init__(self, n_ws, p, q):
14         self.n_ws = n_ws
15         base_dir = os.path.dirname(__file__)
16         self.path = os.path.join(base_dir, 'models', f'{n_ws}-{p}-{q}')
17
18
19     if not os.path.exists(self.path):
20         self.info = {}
21         self.info['eval'] = {}
22         self._init_model(p, q)
23         self.epochs = 0
24     else:
25         with open(os.path.join(self.path, 'info.json'), 'r') as f:
26             self.info = json.load(f)
27         self.epochs = len(self.info['history']['loss'])
28         self.y_max = self.info['y']['max']
29         self.y_min = self.info['y']['min']
30         self.u_max = self.info['u']['max']
31         self.u_min = self.info['u']['min']
32
33         model_path = os.path.join(self.path, f'model_{self.epochs}.h5')
34         self.model = keras.models.load_model(model_path)
35
36     def reset_weights(self, epochs):
37         model_path = os.path.join(self.path, f'model_{epochs}.h5')
38         self.model = keras.models.load_model(model_path)
39
40     def fit(self, epochs):
41         self._init_data()
42         I_train, O_train = self._preprocess(self.y_train, self.u_train, self.n_ws)
43         I_val, O_val = self._preprocess(self.y_val, self.u_val, self.n_ws)
44         history = self.model.fit(
45             I_train, O_train,
46             epochs=epochs,
47             batch_size=self.n_steps-self.n_ws,
48             validation_data=(I_val, O_val)

```

```

49         )
50         self.epochs += epochs
51         if 'history' in self.info.keys():
52             self.info['history']['loss'] += history.history['loss']
53             self.info['history']['mae'] += history.history['mae']
54             self.info['history']['val_loss'] += history.history['val_loss']
55             self.info['history']['val_mae'] += history.history['val_mae']
56         else:
57             self.info['history'] = history.history
58         self._evaluate()
59         self._save()
60
61     def simulate(self, y_true, u):
62
63         y_true = (y_true - self.y_min) / (self.y_max - self.y_min)
64         u = (u - self.u_min) / (self.u_max - self.u_min)
65
66         y_pred = [y_true[i] for i in range(0, self.n_ws)]
67         for k in range(self.n_ws, y_true.shape[0]):
68             I = np.array([y_pred[k-i] for i in range(1, self.n_ws+1)] + [u[k-i] for i in
range(self.n_ws+1)])
69             O = self.model.predict(I)
70             y_pred.append(O[0][0])
71
72         y_pred = np.array(y_pred)
73
74         return y_pred * (self.y_max - self.y_min) + self.y_min
75
76     def _save(self):
77         self.model.save(os.path.join(self.path, f'model_{self.epochs}.h5'))
78         with open(os.path.join(self.path, 'info.json'), 'w') as f:
79             json.dump(self.info, f, indent=4)
80
81     def _evaluate(self):
82         self.info['eval'][str(self.epochs)] = {}
83
84         MSE, MAE = [], []
85         for j in range(self.y_test.shape[0]):
86
87             # rescale the data
88             y_true = self.y_test[j] * (self.y_max - self.y_min) + self.y_min
89             u = self.u_test[j] * (self.u_max - self.u_min) + self.u_min
90
91             # calculate predicted output
92             y_pred = self.simulate(y_true, u)
93
94             # rescale
95             y_true = (y_true - self.y_min) / (self.y_max - self.y_min)
96             y_pred = (y_pred - self.y_min) / (self.y_max - self.y_min)
97

```

```

98         # Metrics
99         MSE += [np.mean((y_true - y_pred) ** 2)]
100         MAE += [np.mean(np.abs(y_true - y_pred))]
101
102     # save test info
103     self.info['eval'][str(self.epochs)]['mse_test'] = np.mean(MSE)
104     self.info['eval'][str(self.epochs)]['mae_test'] = np.mean(MAE)
105
106     # load benchmark data
107     y_true = np.load(os.path.join(self.data_path, 'y_benchmark.npy'))
108     u = np.load(os.path.join(self.data_path, 'u_benchmark.npy'))
109
110     # calculate predicted output
111     y_pred = self.simulate(y_true, u)
112
113     # rescale
114     y_true = (y_true - self.y_min) / (self.y_max - self.y_min)
115     y_pred = (y_pred - self.y_min) / (self.y_max - self.y_min)
116
117     # save benchmark info
118     self.info['eval'][str(self.epochs)]['mse_benchmark'] = np.mean((y_true - y_pred) ** 2)
119     self.info['eval'][str(self.epochs)]['mae_benchmark'] = np.mean(np.abs(y_true - y_pred))
120
121     def _init_model(self, p, q):
122         self.model = Sequential()
123         self.model.add(Dense(p, activation="relu", input_shape=(self.n_ws*2 + 1, ),
kernel_initializer='he_normal'))
124         for _ in range(q-1):
125             self.model.add(Dense(p, activation="relu", kernel_initializer='he_normal'))
126         self.model.add(Dense(1))
127         self.model.compile(optimizer=Adam(learning_rate=1e-3), loss='mse', metrics=['mae'])
128
129     def _init_data(self):
130         y_list = np.load(os.path.join(self.data_path, 'y_list.npy'))
131         u_list = np.load(os.path.join(self.data_path, 'u_list.npy'))
132         self.n_steps = y_list.shape[1]
133         self.n_meas = y_list.shape[0]
134
135         # Split into train-valid-test
136         y_train, y_val, y_test = self._split(y_list)
137         u_train, u_val, u_test = self._split(u_list)
138
139         # Min-max scaler
140         self.y_train, self.y_val, self.y_test, self.y_max, self.y_min =
self._minmaxscaler(y_train, y_val, y_test)
141         self.u_train, self.u_val, self.u_test, self.u_max, self.u_min =
self._minmaxscaler(u_train, u_val, u_test)
142
143         # save the min-max values
144         self.info['y'] = {'max' : self.y_max, 'min' : self.y_min,}
145         self.info['u'] = {'max' : self.u_max, 'min' : self.u_min,}

```

```
146
147     def _split(self,a_list):
148         return a_list[:165], a_list[165:190], a_list[190:]
149
150     def _minmaxscaler(self,a_train, a_val, a_test):
151         a_max = a_train.flatten().max()
152         a_min = a_train.flatten().min()
153
154         a_train = (a_train - a_min)/(a_max - a_min)
155         a_val   = (a_val   - a_min)/(a_max - a_min)
156         a_test  = (a_test  - a_min)/(a_max - a_min)
157
158         return a_train, a_val, a_test, a_max, a_min
159
160     def _preprocess(self,y_list,u_list,n_ws):
161         I, O = [],[]
162         for y, u in zip(y_list, u_list):
163             for k in range(self.n_ws,y.shape[0]):
164                 I += [[y[k-i] for i in range(1,n_ws+1)] + [u[k-i] for i in
range(self.n_ws+1)]]
165                 O += [[y[k]]]
166         I, O = np.array(I), np.array(O)
167         return I, O
168
169
170
171
172
173
174
175
```

0000\MLP\incr\model.py

```

1  import numpy as np
2  import os
3  from keras.models import Sequential
4  from keras.layers import Dense
5  import matplotlib.pyplot as plt
6  import json
7  import keras
8  from keras.optimizers import Adam
9
10
11 class MLP_Increment:
12
13     data_path = 'data'
14
15     def __init__(self, n_ws, p, q):
16         self.n_ws = n_ws
17         base_dir = os.path.dirname(__file__)
18         self.path = os.path.join(base_dir, 'models', f'{n_ws}-{p}-{q}')
19
20         if not os.path.exists(self.path):
21             self.info = {}
22             self.info['eval'] = {}
23             self._init_model(p, q)
24             self.epochs = 0
25         else:
26             with open(os.path.join(self.path, 'info.json'), 'r') as f:
27                 self.info = json.load(f)
28                 self.epochs = len(self.info['history']['loss'])
29                 self.dy_max = self.info['dy']['max']
30                 self.dy_min = self.info['dy']['min']
31                 self.y_max = self.info['y']['max']
32                 self.y_min = self.info['y']['min']
33                 self.u_max = self.info['u']['max']
34                 self.u_min = self.info['u']['min']
35
36             model_path = os.path.join(self.path, f'model_{self.epochs}.h5')
37             self.model = keras.models.load_model(model_path)
38
39     def fit(self, epochs):
40         self._init_data()
41         I_train, O_train = self._preprocess(self.dy_train, self.y_train, self.u_train)
42         I_val, O_val = self._preprocess(self.dy_val, self.y_val, self.u_val)
43
44         history = self.model.fit(
45             I_train, O_train,
46             epochs=epochs,
47             batch_size=self.n_steps-self.n_ws,
48             validation_data=(I_val, O_val)

```

```

49         )
50         self.epochs += epochs
51
52         if 'history' in self.info.keys():
53             self.info['history']['loss'] += history.history['loss']
54             self.info['history']['mae'] += history.history['mae']
55             self.info['history']['val_loss'] += history.history['val_loss']
56             self.info['history']['val_mae'] += history.history['val_mae']
57         else:
58             self.info['history'] = history.history
59         self._evaluate()
60         self._save()
61
62     def reset_weights(self, epochs):
63         model_path = os.path.join(self.path, f'model_{epochs}.h5')
64         self.model = keras.models.load_model(model_path)
65
66     def simulate(self, y_true, u):
67
68         y_true = (y_true - self.y_min) / (self.y_max - self.y_min)
69         u = (u - self.u_min) / (self.u_max - self.u_min)
70
71         y_pred = [y_true[i] for i in range(0, self.n_ws)]
72         for k in range(self.n_ws, y_true.shape[0]):
73             I = np.array([[y_pred[k-i] for i in range(1, self.n_ws+1)] + [u[k-i] for i in
range(self.n_ws+1)]])
74             O = self.model.predict(I)
75             # rescale
76             dy = O[0][0] * (self.dy_max - self.dy_min) + self.dy_min
77             y = y_pred[-1] * (self.y_max - self.y_min) + self.y_min
78             # add
79             y = y + dy
80             # scale
81             y = (y - self.y_min) / (self.y_max - self.y_min)
82             y_pred.append(y)
83
84         y_pred = np.array(y_pred)
85
86         return y_pred * (self.y_max - self.y_min) + self.y_min
87
88     def _save(self):
89         self.model.save(os.path.join(self.path, f'model_{self.epochs}.h5'))
90         with open(os.path.join(self.path, 'info.json'), 'w') as f:
91             json.dump(self.info, f, indent=4)
92
93     def _evaluate(self):
94         self.info['eval'][str(self.epochs)] = {}
95
96         MSE, MAE = [], []
97         for j in range(self.y_test.shape[0]):

```

```

98
99     # rescale the data
100     y_true = self.y_test[j] * (self.y_max - self.y_min) + self.y_min
101     u = self.u_test[j] * (self.u_max - self.u_min) + self.u_min
102
103     # calculate predicted output
104     y_pred = self.simulate(y_true,u)
105
106     # scale
107     y_true = (y_true - self.y_min) / (self.y_max - self.y_min)
108     y_pred = (y_pred - self.y_min) / (self.y_max - self.y_min)
109
110     # Metrics
111     MSE += [np.mean((y_true - y_pred) ** 2)]
112     MAE += [np.mean(np.abs(y_true - y_pred))]
113
114     # save test info
115     self.info['eval'][str(self.epochs)][ 'mse_test' ] = np.mean(MSE)
116     self.info['eval'][str(self.epochs)][ 'mae_test' ] = np.mean(MAE)
117
118     # load benchmark data
119     y_true = np.load(os.path.join(self.data_path, 'y_benchmark.npy'))
120     u = np.load(os.path.join(self.data_path, 'u_benchmark.npy'))
121
122     # calculate predicted output
123     y_pred = self.simulate(y_true,u)
124
125     # scale
126     y_true = (y_true - self.y_min) / (self.y_max - self.y_min)
127     y_pred = (y_pred - self.y_min) / (self.y_max - self.y_min)
128
129     # save benchmark info
130     self.info['eval'][str(self.epochs)][ 'mse_benchmark' ] = np.mean((y_true - y_pred) ** 2)
131     self.info['eval'][str(self.epochs)][ 'mae_benchmark' ] = np.mean(np.abs(y_true - y_pred))
132
133     def _init_model(self,p,q):
134         self.model = Sequential()
135         self.model.add(Dense(p, activation="relu", input_shape=(self.n_ws*2 + 1,),
kernel_initializer='he_normal'))
136         for _ in range(q-1):
137             self.model.add(Dense(p, activation="relu", kernel_initializer='he_normal'))
138         self.model.add(Dense(1))
139         self.model.compile(optimizer=Adam(learning_rate=1e-3), loss='mse', metrics=['mae'])
140
141     def _init_data(self):
142         y_list = np.load(os.path.join(self.data_path, 'y_list.npy'))
143         u_list = np.load(os.path.join(self.data_path, 'u_list.npy'))
144         self.n_steps = y_list.shape[1]
145         self.n_meas = y_list.shape[0]
146         dy_list = np.array([[y[i]-y[i-1] for i in range(1,self.n_steps)]for y in y_list])

```

```

147
148     # Split into train-valid-test
149     dy_train, dy_val, dy_test = self._split(dy_list)
150     y_train, y_val, y_test = self._split(y_list)
151     u_train, u_val, u_test = self._split(u_list)
152
153     # Min-max scaler
154     self.dy_train, self.dy_val, self.dy_test, self.dy_max, self.dy_min =
self._minmaxscaler(dy_train, dy_val, dy_test)
155     self.y_train, self.y_val, self.y_test, self.y_max, self.y_min =
self._minmaxscaler(y_train, y_val, y_test)
156     self.u_train, self.u_val, self.u_test, self.u_max, self.u_min =
self._minmaxscaler(u_train, u_val, u_test)
157
158     # save the min-max values
159     self.info['dy'] = { 'max' : self.dy_max, 'min' : self.dy_min}
160     self.info['y'] = { 'max' : self.y_max, 'min' : self.y_min,}
161     self.info['u'] = { 'max' : self.u_max, 'min' : self.u_min,}
162
163     def _split(self,a_list):
164         return a_list[:165], a_list[165:190], a_list[190:]
165
166     def _minmaxscaler(self,a_train, a_val, a_test):
167         a_max = a_train.flatten().max()
168         a_min = a_train.flatten().min()
169
170         a_train = (a_train - a_min)/(a_max - a_min)
171         a_val = (a_val - a_min)/(a_max - a_min)
172         a_test = (a_test - a_min)/(a_max - a_min)
173
174         return a_train, a_val, a_test, a_max, a_min
175
176     def _preprocess(self,dy_list,y_list,u_list):
177         I, O = [],[]
178         for dy, y, u in zip(dy_list, y_list, u_list):
179             for k in range(self.n_ws,y.shape[0]):
180                 I += [[y[k-i] for i in range(1,self.n_ws+1)] + [u[k-i] for i in
range(self.n_ws+1)]]
181                 O += [[dy[k-1]]]
182         I, O = np.array(I), np.array(O)
183         return I, O
184

```

0000\LSTM\dir\model.py

```

1  import numpy as np
2  import os
3  import json
4  import keras
5  from keras.models import Sequential
6  from keras.layers import Dense, LSTM
7  from keras.optimizers import Adam
8
9
10 class LSTM_Direct:
11
12     data_path = 'data'
13
14     def __init__(self, n_ws, p, q):
15         base_dir = os.path.dirname(__file__)
16         self.path = os.path.join(base_dir, 'models', f'{n_ws}-{p}-{q}')
17         self.n_ws = n_ws
18
19         if not os.path.exists(self.path):
20             self.info = {}
21             self.info['eval'] = {}
22             self._init_model(p, q)
23             self.epochs = 0
24         else:
25             with open(os.path.join(self.path, 'info.json'), 'r') as f:
26                 self.info = json.load(f)
27             self.epochs = len(self.info['history']['loss'])
28             self.y_max = self.info['y']['max']
29             self.y_min = self.info['y']['min']
30             self.u_max = self.info['u']['max']
31             self.u_min = self.info['u']['min']
32
33             model_path = os.path.join(self.path, f'model_{self.epochs}.h5')
34             self.model = keras.models.load_model(model_path)
35
36     def fit(self, epochs):
37         self._init_data()
38         I_train, O_train = self._preprocess(self.y_train, self.u_train)
39         I_val, O_val = self._preprocess(self.y_val, self.u_val)
40         history = self.model.fit(
41             I_train, O_train,
42             epochs=epochs,
43             batch_size=1,
44             validation_data=(I_val, O_val)
45         )
46         self.epochs += epochs
47         if 'history' in self.info.keys():
48             self.info['history']['loss'] += history.history['loss']

```

```

49         self.info['history']['mae'] += history.history['mae']
50         self.info['history']['val_loss'] += history.history['val_loss']
51         self.info['history']['val_mae'] += history.history['val_mae']
52     else:
53         self.info['history'] = history.history
54     self._evaluate()
55     self._save()
56
57     def simulate(self, y_true, u):
58         u = (u - self.u_min) / (self.u_max - self.u_min)
59         I = np.array([[u[k-i] for i in range(self.n_ws+1)] for k in
range(self.n_ws, u.shape[0])])
60         I = I.reshape(1, I.shape[0], I.shape[1])
61         O = self.model.predict(I, batch_size=1)[0, :, 0]
62
63         y_pred = [y_true[i] for i in range(0, self.n_ws)]
64         y_pred += [y for y in O * (self.y_max - self.y_min) + self.y_min]
65         return np.array(y_pred)
66
67     def reset_weights(self, epochs):
68         model_path = os.path.join(self.path, f'model_{epochs}.h5')
69         self.model = keras.models.load_model(model_path)
70
71     def _save(self):
72         self.model.save(os.path.join(self.path, f'model_{self.epochs}.h5'))
73         with open(os.path.join(self.path, 'info.json'), 'w') as f:
74             json.dump(self.info, f, indent=4)
75
76     def _evaluate(self):
77         self.info['eval'][str(self.epochs)] = {}
78         I_test, O_test = self._preprocess(self.y_test, self.u_test)
79         MSE, MAE = self.model.evaluate(I_test, O_test, batch_size=1, verbose=1)
80
81         # save test info
82         self.info['eval'][str(self.epochs)]['mse_test'] = np.mean(MSE)
83         self.info['eval'][str(self.epochs)]['mae_test'] = np.mean(MAE)
84
85         # load benchmark data
86         y_true = np.load(os.path.join(self.data_path, 'y_benchmark.npy'))
87         u = np.load(os.path.join(self.data_path, 'u_benchmark.npy'))
88
89         # calculate predicted output
90         y_pred = self.simulate(y_true, u)
91
92         # rescale
93         y_true = (y_true - self.y_min) / (self.y_max - self.y_min)
94         y_pred = (y_pred - self.y_min) / (self.y_max - self.y_min)
95
96         # save benchmark info
97         self.info['eval'][str(self.epochs)]['mse_benchmark'] = np.mean((y_true - y_pred) ** 2)

```

```

98         self.info['eval'][str(self.epochs)]['mae_benchmark'] = np.mean(np.abs(y_true - y_pred))
99
100     def _init_data(self):
101         y_list = np.load(os.path.join(self.data_path, 'y_list.npy'))
102         u_list = np.load(os.path.join(self.data_path, 'u_list.npy'))
103         self.n_steps = y_list.shape[1]
104         self.n_meas = y_list.shape[0]
105
106         # Split into train-valid-test
107         y_train, y_val, y_test = self._split(y_list)
108         u_train, u_val, u_test = self._split(u_list)
109
110         # Min-max scaler
111         self.y_train, self.y_val, self.y_test, self.y_max, self.y_min =
self._minmaxscaler(y_train, y_val, y_test)
112         self.u_train, self.u_val, self.u_test, self.u_max, self.u_min =
self._minmaxscaler(u_train, u_val, u_test)
113
114         # save the min-max values
115         self.info['y'] = {'max' : self.y_max, 'min' : self.y_min,}
116         self.info['u'] = {'max' : self.u_max, 'min' : self.u_min,}
117
118     def _init_model(self, p, q):
119         self.model = Sequential()
120         self.model.add(LSTM(p, activation="tanh", input_shape=
(None, self.n_ws+1), return_sequences=True))
121         for _ in range(q-1):
122             self.model.add(LSTM(p, activation="tanh", return_sequences=True))
123         self.model.add(Dense(1))
124         self.model.compile(optimizer=Adam(learning_rate=1e-3), loss='mse', metrics=['mae'])
125
126     def _split(self, a_list):
127         return a_list[:165], a_list[165:190], a_list[190:]
128
129     def _minmaxscaler(self, a_train, a_val, a_test):
130         a_max = a_train.flatten().max()
131         a_min = a_train.flatten().min()
132
133         a_train = (a_train - a_min)/(a_max - a_min)
134         a_val = (a_val - a_min)/(a_max - a_min)
135         a_test = (a_test - a_min)/(a_max - a_min)
136
137         return a_train, a_val, a_test, a_max, a_min
138
139     def _preprocess(self, y_list, u_list):
140         I, O = [], []
141         for y, u in zip(y_list, u_list):
142             I_tmp, O_tmp = [], []
143             for k in range(self.n_ws, u.shape[0]):
144                 I_tmp += [[u[k-i] for i in range(self.n_ws+1)]]
145                 O_tmp += [[y[k]]]

```

10/31/24, 8:17 PM

model.py

```
146         I += [I_tmp]
147         O += [O_tmp]
148     I, O = np.array(I), np.array(O)
149     return I, O
```

0000\LSTM\incr\model.py

```

1  import numpy as np
2  import os
3  import json
4  import keras
5  from keras.models import Sequential
6  from keras.layers import Dense, LSTM
7  from keras.optimizers import Adam
8
9
10 class LSTM_Increment:
11
12     data_path = 'data'
13
14     def __init__(self, n_ws, p, q):
15         base_dir = os.path.dirname(__file__)
16         self.path = os.path.join(base_dir, 'models', f'{n_ws}-{p}-{q}')
17         self.n_ws = n_ws
18
19         if not os.path.exists(self.path):
20             self.info = {}
21             self.info['eval'] = {}
22             self._init_model(p, q)
23             self.epochs = 0
24         else:
25             with open(os.path.join(self.path, 'info.json'), 'r') as f:
26                 self.info = json.load(f)
27             self.epochs = len(self.info['history']['loss'])
28             self.dy_max = self.info['dy']['max']
29             self.dy_min = self.info['dy']['min']
30             self.y_max = self.info['y']['max']
31             self.y_min = self.info['y']['min']
32             self.u_max = self.info['u']['max']
33             self.u_min = self.info['u']['min']
34
35             model_path = os.path.join(self.path, f'model_{self.epochs}.h5')
36             self.model = keras.models.load_model(model_path)
37
38     def fit(self, epochs):
39         self._init_data()
40         I_train, O_train = self._preprocess(self.dy_train, self.u_train)
41         I_val, O_val = self._preprocess(self.dy_val, self.u_val)
42         history = self.model.fit(
43             I_train, O_train,
44             epochs=epochs,
45             batch_size=1,
46             validation_data=(I_val, O_val)
47         )
48         self.epochs += epochs

```

```

49     if 'history' in self.info.keys():
50         self.info['history']['loss'] += history.history['loss']
51         self.info['history']['mae'] += history.history['mae']
52         self.info['history']['val_loss'] += history.history['val_loss']
53         self.info['history']['val_mae'] += history.history['val_mae']
54     else:
55         self.info['history'] = history.history
56     self._evaluate()
57     self._save()
58
59     def simulate(self, y_true, u):
60
61         u = (u - self.u_min) / (self.u_max - self.u_min)
62
63         I = np.array([u[k-i] for i in range(self.n_ws+1)] for k in
64 range(self.n_ws, u.shape[0]))
65         I = I.reshape(1, I.shape[0], I.shape[1])
66
67         O = self.model.predict(I, batch_size=1)[0, :, 0]
68         dy_pred = 0 * (self.dy_max - self.dy_min) + self.dy_min
69
70         y_pred = [0] + [y_true[i] for i in range(0, self.n_ws)]
71
72         for dy in dy_pred:
73             y_pred += [y_pred[-1] + dy]
74
75         return np.array(y_pred[1:])
76
77     def reset_weights(self, epochs):
78         model_path = os.path.join(self.path, f'model_{epochs}.h5')
79         self.model = keras.models.load_model(model_path)
80
81     def _save(self):
82         self.model.save(os.path.join(self.path, f'model_{self.epochs}.h5'))
83         with open(os.path.join(self.path, 'info.json'), 'w') as f:
84             json.dump(self.info, f, indent=4)
85
86     def _evaluate(self):
87         self.info['eval'][str(self.epochs)] = {}
88
89         MSE, MAE = [], []
90         for j in range(self.u_test.shape[0]):
91             # rescale
92             y_true = self.y_test[j] * (self.y_max - self.y_min) + self.y_min
93             u = self.u_test[j] * (self.u_max - self.u_min) + self.u_min
94
95             # calculate predicted output
96             y_pred = self.simulate(y_true, u)
97
98             # scale

```

```

98         y_true = (y_true - self.y_min) / (self.y_max - self.y_min)
99         y_pred = (y_pred - self.y_min) / (self.y_max - self.y_min)
100
101         # Metrics
102         MSE += [np.mean((y_true - y_pred) ** 2)]
103         MAE += [np.mean(np.abs(y_true - y_pred))]
104
105         # save test info
106         self.info['eval'][str(self.epochs)]['mse_test'] = np.mean(MSE)
107         self.info['eval'][str(self.epochs)]['mae_test'] = np.mean(MAE)
108
109         # load benchmark data
110         y_true = np.load(os.path.join(self.data_path, 'y_benchmark.npy'))
111         u = np.load(os.path.join(self.data_path, 'u_benchmark.npy'))
112
113         # calculate predicted output
114         y_pred = self.simulate(y_true, u)
115
116         # rescale
117         y_true = (y_true - self.y_min) / (self.y_max - self.y_min)
118         y_pred = (y_pred - self.y_min) / (self.y_max - self.y_min)
119
120         # save benchmark info
121         self.info['eval'][str(self.epochs)]['mse_benchmark'] = np.mean((y_true - y_pred) ** 2)
122         self.info['eval'][str(self.epochs)]['mae_benchmark'] = np.mean(np.abs(y_true - y_pred))
123
124     def _init_data(self):
125         y_list = np.load(os.path.join(self.data_path, 'y_list.npy'))
126         u_list = np.load(os.path.join(self.data_path, 'u_list.npy'))
127         self.n_steps = y_list.shape[1]
128         self.n_meas = y_list.shape[0]
129         dy_list = np.array([[y[i] - y[i-1] for i in range(1, self.n_steps)] for y in y_list])
130
131         # Split into train-valid-test
132         dy_train, dy_val, dy_test = self._split(dy_list)
133         y_train, y_val, y_test = self._split(y_list)
134         u_train, u_val, u_test = self._split(u_list)
135
136         # Min-max scaler
137         self.dy_train, self.dy_val, self.dy_test, self.dy_max, self.dy_min =
self._minmaxscaler(dy_train, dy_val, dy_test)
138         self.y_train, self.y_val, self.y_test, self.y_max, self.y_min =
self._minmaxscaler(y_train, y_val, y_test)
139         self.u_train, self.u_val, self.u_test, self.u_max, self.u_min =
self._minmaxscaler(u_train, u_val, u_test)
140
141         # save the min-max values
142         self.info['dy'] = {'max': self.dy_max, 'min': self.dy_min}
143         self.info['y'] = {'max': self.y_max, 'min': self.y_min}
144         self.info['u'] = {'max': self.u_max, 'min': self.u_min}
145

```

```
146     def _init_model(self,p,q):
147         self.model = Sequential()
148         self.model.add(LSTM(p, activation="tanh", input_shape=
(None,self.n_ws+1),return_sequences=True))
149         for _ in range(q-1):
150             self.model.add(LSTM(p, activation="tanh",return_sequences=True))
151         self.model.add(Dense(1))
152         self.model.compile(optimizer=Adam(learning_rate=1e-3), loss='mse', metrics=['mae'])
153
154     def _split(self,a_list):
155         return a_list[:165], a_list[165:190], a_list[190:]
156
157     def _minmaxscaler(self,a_train, a_val, a_test):
158         a_max = a_train.flatten().max()
159         a_min = a_train.flatten().min()
160
161         a_train = (a_train - a_min)/(a_max - a_min)
162         a_val = (a_val - a_min)/(a_max - a_min)
163         a_test = (a_test - a_min)/(a_max - a_min)
164
165         return a_train, a_val, a_test, a_max, a_min
166
167     def _preprocess(self,dy_list,u_list):
168         I, O = [],[]
169         for dy, u in zip(dy_list, u_list):
170             I_tmp, O_tmp = [],[]
171             for k in range(self.n_ws,u.shape[0]):
172                 I_tmp += [[u[k-i] for i in range(self.n_ws+1)]]
173                 O_tmp += [[dy[k-1]]]
174             I += [I_tmp]
175             O += [O_tmp]
176         I, O = np.array(I), np.array(O)
177         return I, O
```

0000\develop.py

```
1 from MLP.dir.model import MLP_Direct
2 from MLP.incr.model import MLP_Increment
3 from LSTM.dir.model import LSTM_Direct
4 from LSTM.incr.model import LSTM_Increment
5
6
7 default_range = [2,3,5,8]
8 model_list = [MLP_Direct, MLP_Increment, LSTM_Direct, LSTM_Increment]
9
10 for n_ws in default_range:
11     for p in default_range:
12         for q in default_range:
13             for MODEL in model_list:
14                 dyn_sys = MODEL(n_ws,p,q)
15                 dyn_sys.fit(50)
16                 dyn_sys.fit(50)
17
18
```

```
In [6]: from MLP.dir.model import MLP_Direct
from MLP.incr.model import MLP_Increment
from LSTM.dir.model import LSTM_Direct
from LSTM.incr.model import LSTM_Increment
import numpy as np
import os
import matplotlib.pyplot as plt
import json
import pandas as pd
import seaborn as sns
```

```
In [36]: nn_type = 'MLP'
pred_type = 'dir'

n_ws_range = [2,3,5,8]
p_range     = [2,3,5,8]
q_range     = [2,3,5,8]
```

```
In [38]: nn_type = 'MLP'
pred_type = 'incr'

n_ws_range = [2,3,5,8]
p_range     = [2,3,5,8]
q_range     = [2,3,5,8]
```

```
In [2]: nn_type = 'LSTM'
pred_type = 'dir'

n_ws_range = [0,2,3,5,8,10]
p_range     = [2,3,5,8,10]
q_range     = [2]
```

```
In [1]: nn_type = 'LSTM'
pred_type = 'incr'

n_ws_range = [0,2,3,5,8,10]
p_range     = [2,3,5,8,10]
q_range     = [2]
```

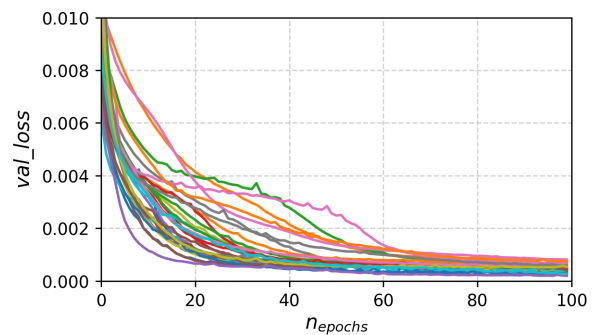
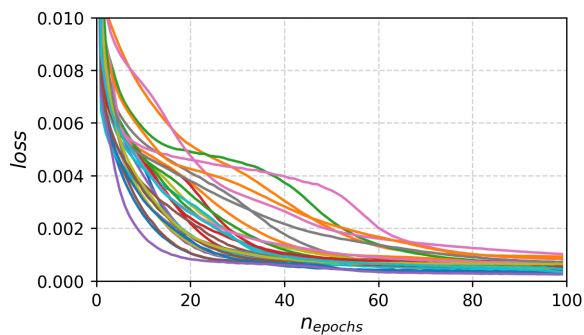
Convergence

```
In [50]: fig, ax = plt.subplots(1,2,figsize=(10,3),dpi=250)
model_names = os.listdir(f'{nn_type}/{pred_type}')
for model_name in os.listdir(f'{nn_type}/{pred_type}/models'):
    n_ws, p, q = map(int, model_name.split('-'))
    if n_ws in n_ws_range and p in p_range and q in q_range:
        with open(f'{nn_type}/{pred_type}/models/{model_name}/info.json','r') as f:
            info = json.load(f)
            ax[0].plot(info['history']['loss'],label=model_name)
            ax[1].plot(info['history']['val_loss'],label=model_name)
```

```

ax[0].set_xlabel(r'$n_{epochs}$', fontsize=12)
ax[1].set_xlabel(r'$n_{epochs}$', fontsize=12)
ax[0].set_ylabel(r'$loss$', fontsize=12)
ax[1].set_ylabel(r'$val\_loss$', fontsize=12)
ax[0].set_xlim(0,100)
ax[1].set_xlim(0,100)
ax[0].set_ylim(0,0.01)
ax[1].set_ylim(0,0.01)
ax[0].grid(ls='--',c='lightgray')
ax[1].grid(ls='--',c='lightgray')
plt.tight_layout()
plt.show()

```



Max MSE (100 epoch)

```

In [10]: columns = [
    'k+1',
    'p',
    'q',
    'mse_test',
    'mse_benchmark',
]

df = pd.DataFrame(columns=columns)

model_names = os.listdir(f'{nn_type}/{pred_type}')
for model_name in os.listdir(f'{nn_type}/{pred_type}/models'):
    n_ws, p, q = map(int, model_name.split('-'))
    if n_ws in n_ws_range and p in p_range and q in q_range:
        with open(f'{nn_type}/{pred_type}/models/{model_name}/info.json', 'r') as f:
            info = json.load(f)
            for epochs in ['50', '100']:
                if '100' in info['eval'].keys():
                    data = {
                        'k+1': n_ws,
                        'p': p,
                        'q': q,
                        'epochs': epochs,
                        'mse_test': info['eval'][epochs]['mse_test'],
                        'mse_benchmark': info['eval'][epochs]['mse_benchmark'],
                    }
                    new_row = pd.DataFrame([data])
                    df = pd.concat([df, new_row], ignore_index=True)

```

```

print(f'Best {nn_type}_{pred_type} on TEST')
idx = df['mse_test'].idxmin()
display(df.loc[idx])

print(f'Best {nn_type}_{pred_type} on BENCHMARK')
idx = df['mse_benchmark'].idxmin()
display(df.loc[idx])

```

C:\Users\Zsoci\AppData\Local\Temp\ipykernel_15296\1479325465.py:28: FutureWarning: The behavior of DataFrame concatenation with empty or all-NA entries is deprecated. In a future version, this will no longer exclude empty or all-NA columns when determining the result dtypes. To retain the old behavior, exclude the relevant entries before the concat operation.

```
df = pd.concat([df, new_row], ignore_index=True)
```

```

Best LSTM_incr on TEST
k+1                5
p                  8
q                  2
mse_test           0.002224
mse_benchmark      0.364714
epochs             100
Name: 49, dtype: object
Best LSTM_incr on BENCHMARK
k+1                10
p                  3
q                  2
mse_test           0.012621
mse_benchmark      0.032555
epochs             100
Name: 15, dtype: object

```

Visual evaluations

```

In [12]: plot_params = [(5,8,2,100,'red'),(10,3,2,100,'magenta')]

fig,ax = plt.subplots(1,2,figsize=(9,3),dpi=300)

i = 10
y_list = [np.load('data/y_list.npy')[i], np.load('data/y_benchmark.npy')]
u_list = [np.load('data/u_list.npy')[i], np.load('data/u_benchmark.npy')]

for i,(y_true,u) in enumerate(zip(y_list,u_list)):
    ax[i].plot(y_true,color='#00bd17',label='true')

    for n_ws,p,q,epochs,color in plot_params:

        # INIT MODEL
        if pred_type == 'dir' and nn_type == 'MLP':
            dyn_sys = MLP_Direct(n_ws,p,q)
        if pred_type == 'incr' and nn_type == 'MLP':
            dyn_sys = MLP_Increment(n_ws,p,q)
        if pred_type == 'dir' and nn_type == 'LSTM':

```

```

dyn_sys = LSTM_Direct(n_ws,p,q)
if pred_type == 'incr' and nn_type == 'LSTM':
    dyn_sys = LSTM_Increment(n_ws,p,q)
dyn_sys.reset_weights(epochs)

y_pred = dyn_sys.simulate(y_true,u)
ax[i].plot(y_pred,color=color,ls='--',label=f'({n_ws-1},{p},{q})')

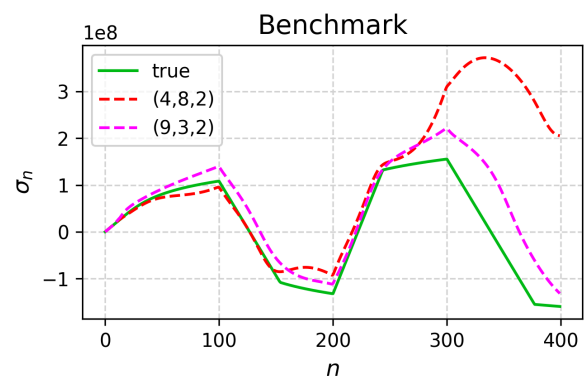
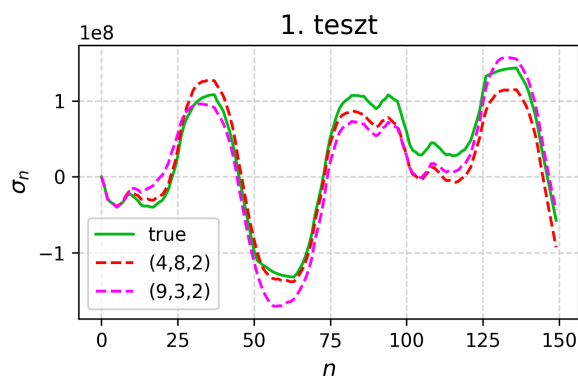
ax[0].set_ylabel(r'\sigma_n$', fontsize=12)
ax[0].set_title(f'{i}. teszt', fontsize=14)
ax[0].legend()
ax[1].set_ylabel(r'\sigma_n$', fontsize=12)
ax[1].set_title('Benchmark', fontsize=14)
ax[1].legend()
ax[0].grid(ls='--',c='lightgray')
ax[1].grid(ls='--',c='lightgray')
ax[0].set_xlabel(r'$n$', fontsize=12)
ax[1].set_xlabel(r'$n$', fontsize=12)
plt.tight_layout()
plt.show()

```

```

1/1 [=====] - 1s 697ms/step
1/1 [=====] - 1s 710ms/step
1/1 [=====] - 1s 784ms/step
1/1 [=====] - 1s 766ms/step

```



Statistical evaluation

MLP

```

In [39]: epochs = '100'

colors = {
    '0' : 'gray',
    '2' : 'red',
    '3' : 'orange',
    '5' : 'green',
    '8' : 'blue',
}

```

```

model_names = os.listdir(f'{nn_type}/{pred_type}')
mse_test_list = []
mse_benchmark_list = []
n_ws_list, p_list, q_list = [], [], []
for model_name in os.listdir(f'{nn_type}/{pred_type}/models'):
    n_ws, p, q = map(int, model_name.split('-'))
    with open(f'{nn_type}/{pred_type}/models/{model_name}/info.json', 'r') as f:
        info = json.load(f)
        if epochs in info['eval'].keys():
            if n_ws in n_ws_range and p in p_range and q in q_range:
                mse_test = info['eval'][epochs]['mse_test']
                mse_benchmark = info['eval'][epochs]['mse_benchmark']
                if mse_test < 0.5 and mse_benchmark < 1:
                    mse_test_list += [mse_test]
                    mse_benchmark_list += [mse_benchmark]
                    n_ws_list += [n_ws]
                    p_list += [p]
                    q_list += [q]

df = pd.DataFrame({'(k+1)': n_ws_list, 'p': p_list, 'q': q_list, 'mse_test': mse_t

def boxplot(x,y,df,ax):
    # Adjust the figure size
    sns.boxplot(x=x, y=y, data=df, ax=ax, showmeans=False, showfliers=False,
                meanprops={"marker": "o", "markerfacecolor": "white", "markeredgecolor": "black", "markersize": 100},
                boxprops=dict(alpha=0.7), # Make the boxes slightly transparent
                flierprops=dict(marker='o', color='red', alpha=0.6) # Customize outliers
    )
    ax.set_xlabel('')
    ax.set_ylabel('')

fig,ax = plt.subplots(3,2,figsize=(8, 12),dpi=250)

for i,x in enumerate(['(k+1)', 'p', 'q']):
    for j,y in enumerate(['mse_test', 'mse_benchmark']):
        boxplot(x=x,y=y,df=df, ax=ax[i][j])
        ylabel = r'$MSE_{benchmark}$' if y == 'mse_benchmark' else r'$MSE_{test}$'
        ax[i][j].set_ylabel(ylabel,fontsize=12)
        ax[i][j].set_xlabel(x,fontsize=12)

plt.suptitle(f"{'Direkt' if pred_type == 'dir' else 'Inkrementális'} {nn_type} háló",
             fontsize=24,
             weight='bold')
plt.tight_layout(rect=[0, 0, 1, 0.98])
plt.show()

mse_test_list = np.array(mse_test_list)
mse_benchmark_list = np.array(mse_benchmark_list)
n_ws_list = np.array(n_ws_list)
p_list = np.array(p_list)
q_list = np.array(q_list)
n_params = (n_ws_list*2+1)*p_list + p_list + (p_list*p_list + p_list) * (q_list-1)
color_list = np.array([None]*n_params.shape[0])

```

```

fig, ax = plt.subplots(1,2,figsize=(7,2),dpi=350,sharex=True)
for j,metric in enumerate(['mse_test','mse_benchmark']):
    ax[j].scatter(n_params,df[metric],c='black',marker='.')
    ylabel = r'$MSE_{benchmark}$' if metric == 'mse_benchmark' else r'$MSE_{test}$'
    ax[j].set_ylabel(ylabel,fontsize=10)
    ax[j].set_xlabel(r'$n_{params}$',fontsize=10)
    ax[j].set_xlim(0,1000)

plt.tight_layout(rect=[0, 0, 1, 0.98])
plt.show()

fig, ax = plt.subplots(3,2,figsize=(7,4.5),dpi=350,sharex=True)

for i,param in enumerate(['(k+1)','p','q']):
    for j,metric in enumerate(['mse_test','mse_benchmark']):
        means = []
        for num in colors.keys():
            idx = np.where(df[param]==int(num))
            color_list[idx] = colors[num]
            if idx[0].shape[0] != 0:
                ax[i][j].scatter(n_params[idx].mean(),np.array(df[metric])[idx].mean(),
                                color=colors[num],label=f'{param} = {num}')
                means += [[n_params[idx].mean(),np.array(df[metric])[idx].mean()]]

        ax[i][j].scatter(n_params,df[metric],c=color_list,marker='.')
        ax[i][j].legend()
        ylabel = r'$MSE_{benchmark}$' if metric == 'mse_benchmark' else r'$MSE_{test}$'
        ax[i][j].set_ylabel(ylabel,fontsize=10)
        ax[-1][j].set_xlabel(r'$n_{params}$',fontsize=10)
        ax[i][j].set_xlim(0,1000)

        means = np.array(means)

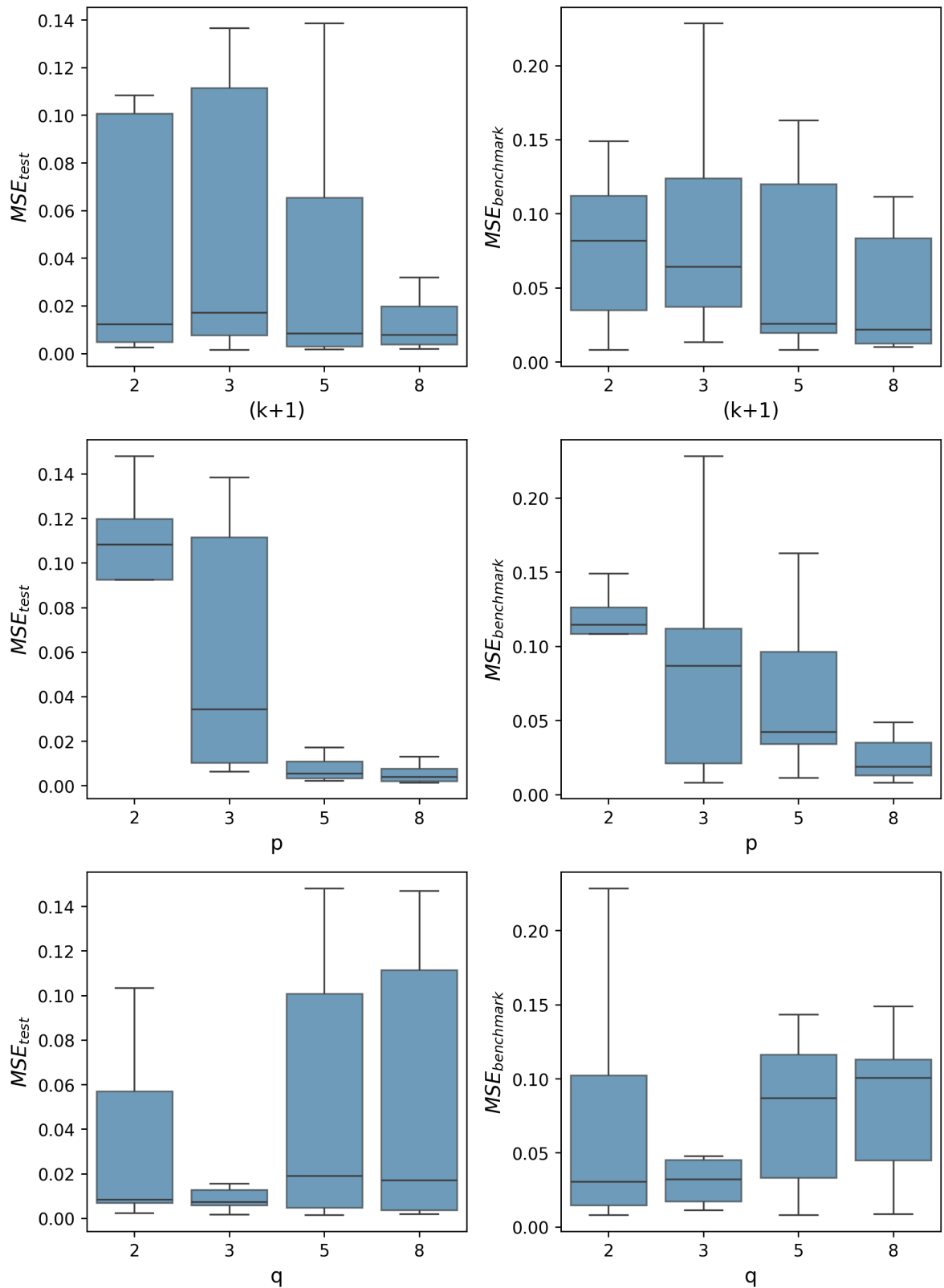
        ax[i][j].plot(means[:,0],means[:,1],color='black')

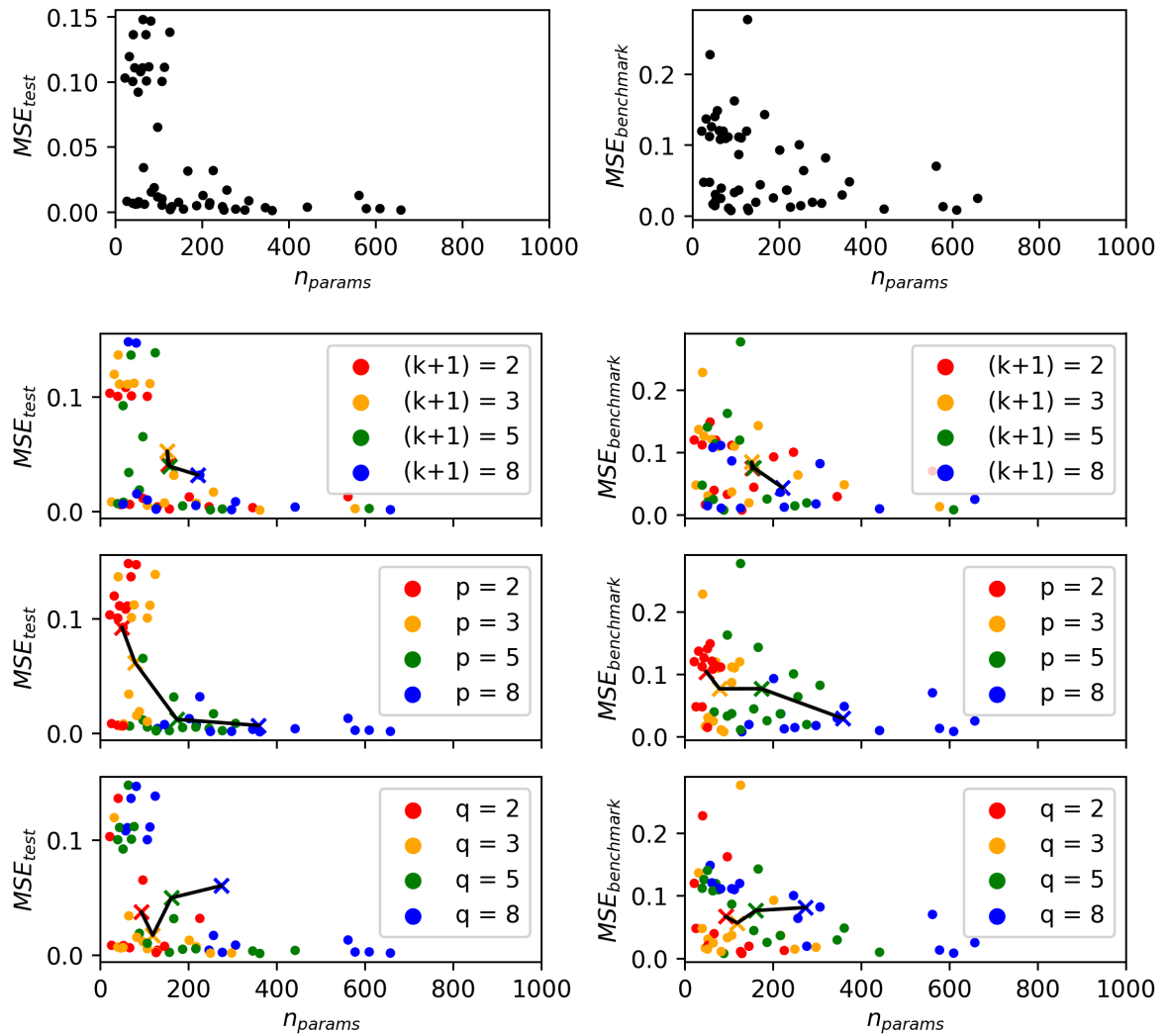
#plt.suptitle(f"'Direkt' if pred_type == 'dir' else 'Inkrementális' {nn_type} hál
#             fontsize=24,
#             weight='bold')
plt.tight_layout(rect=[0, 0, 1, 0.98])
plt.show()

```

Inkrementális MLP hálózat

MSE eloszlás a paraméterek mentén





In [120...

```

p_fix = 8
q_fix = 2

plots = {
    '50': {
        'c' : '#FFD700',
        'n_params': [],
        'mse_test': [],
        'mse_benchmark': [],
    },
    '100': {
        'c' : "#00FF00",
        'n_params': [],
        'mse_test': [],
        'mse_benchmark': [],
    },
    '150': {
        'c' : "#00FFFF",
        'n_params': [],
        'mse_test': [],
        'mse_benchmark': [],
    },
}

```

```

    '200': {
        'c' : "#0000FF",
        'n_params': [],
        'mse_test': [],
        'mse_benchmark': [],
    }
}

for model_name in os.listdir(f'{nn_type}/{pred_type}/models'):
    n_ws, p, q = map(int, model_name.split('-'))
    if p == p_fix and q == q_fix:
        with open(f'{nn_type}/{pred_type}/models/{model_name}/info.json', 'r') as f:
            info = json.load(f)
            print(n_ws, p, q, f":{info['eval'].keys()}")
            n_params = (n_ws*2+1)*p + p + (p*p + p) * (q-1) + (p + 1)*1
            for epochs in info['eval'].keys():
                plots[epochs]['n_params'] += [n_params]
                plots[epochs]['mse_test'] += [info['eval'][epochs]['mse_test']]
                plots[epochs]['mse_benchmark'] += [info['eval'][epochs]['mse_benchmark']]

fig, ax = plt.subplots(1,2,figsize=(8,3),dpi=150,sharex=True)

for epochs in plots.keys():
    for i, metric in enumerate(['mse_test', 'mse_benchmark']):
        x = [n for n, _ in sorted(zip(plots[epochs]['n_params'], plots[epochs][metric]))]
        y = [mse for _, mse in sorted(zip(plots[epochs]['n_params'], plots[epochs][metric]))]
        x,y = np.array(x), np.array(y)
        ax[i].plot(x,np.log10(y),c=plots[epochs]['c'],marker='o',label=epochs)
        ylabel = r'log$(MSE_{benchmark})$' if metric == 'mse_benchmark' else r'log$(MSE_{test})$'
        ax[i].set_ylabel(ylabel,fontsize=12)
        ax[i].set_xlabel(r'$\#{params}$', fontsize=12)
        ax[i].legend(loc='upper left')
        ax[i].grid(ls='--',c='lightgray')

ax[0].set_ylim(-4,0)
ax[1].set_ylim(-4,0)

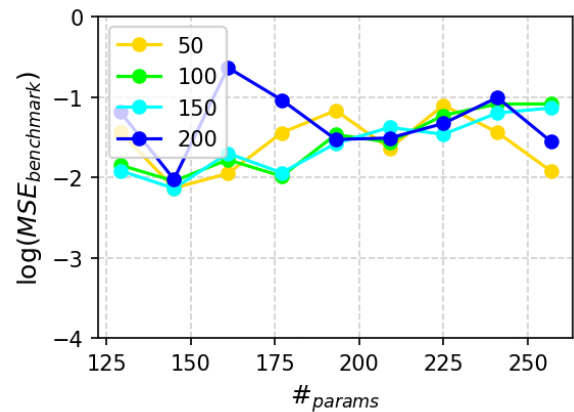
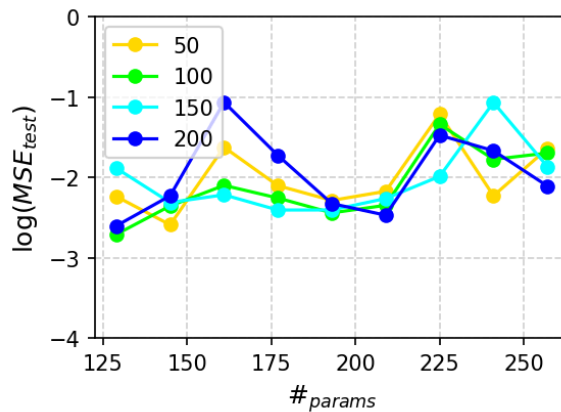
plt.tight_layout(rect=[0, 0, 1, 0.98])
plt.show()

```

```

10 8 2 :dict_keys(['50', '100', '150', '200'])
2 8 2 :dict_keys(['50', '100', '150', '200'])
3 8 2 :dict_keys(['50', '100', '150', '200'])
4 8 2 :dict_keys(['50', '100', '150', '200'])
5 8 2 :dict_keys(['50', '100', '150', '200'])
6 8 2 :dict_keys(['50', '100', '150', '200'])
7 8 2 :dict_keys(['50', '100', '150', '200'])
8 8 2 :dict_keys(['50', '100', '150', '200'])
9 8 2 :dict_keys(['50', '100', '150', '200'])

```



LSTM

```
In [71]: epochs = '100'

colors = {
    '0' : 'gray',
    '2' : 'red',
    '3' : 'orange',
    '5' : 'green',
    '8' : 'blue',
    '10' : 'magenta',
}

model_names = os.listdir(f'{nn_type}/{pred_type}')
mse_test_list = []
mse_benchmark_list = []
n_ws_list, p_list, q_list = [], [], []
for model_name in os.listdir(f'{nn_type}/{pred_type}/models'):
    n_ws, p, q = map(int, model_name.split('-'))
    with open(f'{nn_type}/{pred_type}/models/{model_name}/info.json', 'r') as f:
        info = json.load(f)
        if epochs in info['eval'].keys():
            if n_ws in n_ws_range and p in p_range and q in q_range:
                mse_test = info['eval'][epochs]['mse_test']
                mse_benchmark = info['eval'][epochs]['mse_benchmark']
                if mse_test < 5 and mse_benchmark < 5:
                    mse_test_list += [mse_test]
                    mse_benchmark_list += [mse_benchmark]
                    n_ws_list += [n_ws]
                    p_list += [p]
                    q_list += [q]

df = pd.DataFrame({'(k+1)': n_ws_list, 'p': p_list, 'q': q_list, 'mse_test': mse_t

mse_test_list = np.array(mse_test_list)
mse_benchmark_list = np.array(mse_benchmark_list)
n_ws_list = np.array(n_ws_list)
p_list = np.array(p_list)
q_list = np.array(q_list)
n_params = 4*(p_list*(n_ws_list+1)+p_list*p_list+p_list)
```

```

n_params += (q_list-1)*4*(p_list*p_list+p_list*p_list+p_list)
n_params += (p_list+1)
color_list = np.array([None]*n_params.shape[0])

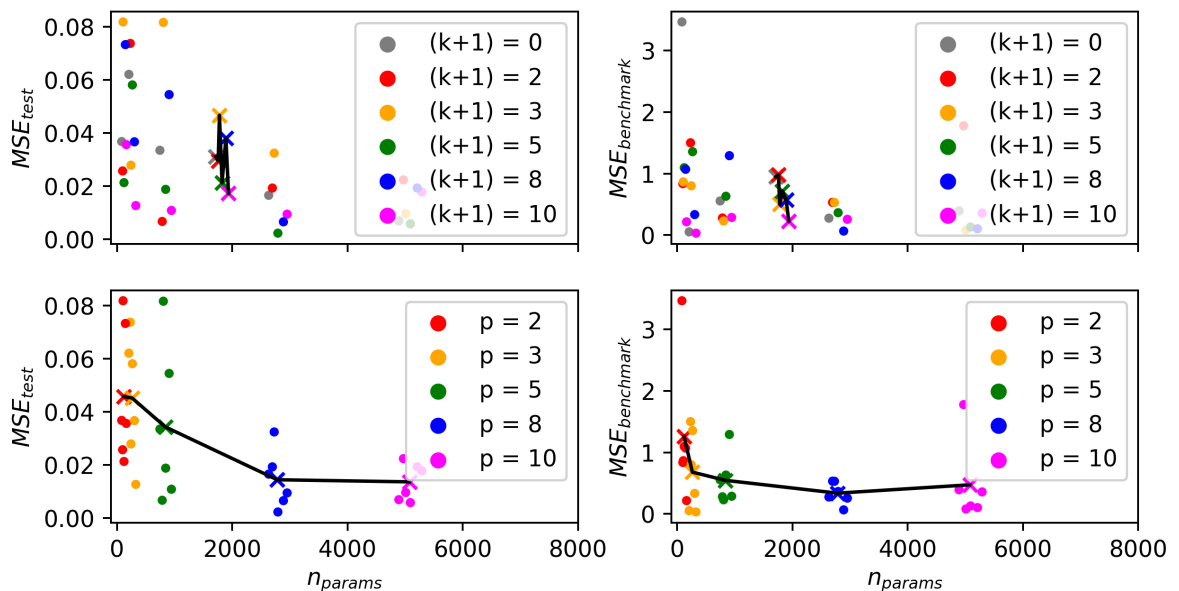
fig, ax = plt.subplots(2,2,figsize=(8,4),dpi=350,sharex=True)

for i,param in enumerate(['(k+1)', 'p']):
    for j,metric in enumerate(['mse_test', 'mse_benchmark']):
        means = []
        for num in colors.keys():
            idx = np.where(df[param]==int(num))
            color_list[idx] = colors[num]
            if idx[0].shape[0] != 0:
                ax[i][j].scatter(n_params[idx].mean(),np.array(df[metric])[idx].mean())
                ax[i][j].scatter([],[],color=colors[num],label=f'{param} = {num}')
                means += [[n_params[idx].mean(),np.array(df[metric])[idx].mean()]]

        ax[i][j].scatter(n_params,df[metric],c=color_list,marker='.')
        ax[i][j].legend()
        ylabel = r'$MSE_{benchmark}$' if metric == 'mse_benchmark' else r'$MSE_{test}$'
        ax[i][j].set_ylabel(ylabel,fontsize=10)
        ax[-1][j].set_xlabel(r'$n_{params}$',fontsize=10)
        ax[i][j].set_xlim(-100,8000)
        means = np.array(means)
        ax[i][j].plot(means[:,0],means[:,1],color='black')

plt.show()

```



In [70]: df

Out[70]:

	(k+1)	p	q	mse_test	mse_benchmark
0	0	10	2	0.006850	0.393578
1	0	3	2	0.062051	0.053294
2	0	5	2	0.033442	0.556136
3	0	8	2	0.016480	0.275712
4	10	10	2	0.017721	0.358666
5	10	2	2	0.035583	0.217055
6	10	3	2	0.012621	0.032555
7	10	5	2	0.010790	0.287889
8	10	8	2	0.009384	0.257645
9	2	2	2	0.025662	0.836582
10	2	5	2	0.006632	0.279225
11	2	8	2	0.019221	0.532648
12	3	10	2	0.009568	0.080965
13	3	2	2	0.081803	0.865940
14	3	3	2	0.027838	0.801917
15	3	5	2	0.081621	0.230240
16	3	8	2	0.032373	0.531472
17	5	10	2	0.005748	0.130285
18	5	5	2	0.018728	0.632195
19	5	8	2	0.002224	0.364714
20	8	10	2	0.019229	0.103553
21	8	3	2	0.036637	0.334364
22	8	8	2	0.006487	0.066535

Max MSE (all time)

```
In [11]: columns = [  
    'k+1',  
    'p',  
    'q',  
    'epochs',  
    'mse_test',  
    'mse_benchmark',  
]
```

```

df = pd.DataFrame(columns=columns)

model_names = os.listdir(f'{nn_type}/{pred_type}')
for model_name in os.listdir(f'{nn_type}/{pred_type}/models'):
    n_ws, p, q = map(int, model_name.split('-'))
    with open(f'{nn_type}/{pred_type}/models/{model_name}/info.json', 'r') as f:
        info = json.load(f)
        for epochs in info['eval'].keys():
            data = {
                'k+1': n_ws,
                'p': p,
                'q': q,
                'epochs': epochs,
                'mse_test': info['eval'][epochs]['mse_test'],
                'mse_benchmark': info['eval'][epochs]['mse_benchmark']
            }
            new_row = pd.DataFrame([data])
            df = pd.concat([df, new_row], ignore_index=True)

print(f'Best {nn_type}_{pred_type} on TEST')
idx = df['mse_test'].idxmin()
display(df.loc[idx])

print(f'Best {nn_type}_{pred_type} on BENCHMARK')
idx = df['mse_benchmark'].idxmin()
display(df.loc[idx])

```

Best LSTM_incr on TEST

C:\Users\Zsoci\AppData\Local\Temp\ipykernel_15296\2507134465.py:27: FutureWarning: The behavior of DataFrame concatenation with empty or all-NA entries is deprecated. In a future version, this will no longer exclude empty or all-NA columns when determining the result dtypes. To retain the old behavior, exclude the relevant entries before the concat operation.

```
df = pd.concat([df, new_row], ignore_index=True)
```

```

k+1      5
p         8
q         2
epochs   100
mse_test  0.002224
mse_benchmark  0.364714
Name: 63, dtype: object
Best LSTM_incr on BENCHMARK
k+1      10
p         3
q         2
epochs   100
mse_test  0.012621
mse_benchmark  0.032555
Name: 15, dtype: object

```

Model size

MLP

```
In [80]: n_ws = 2
p = 8
q = 2

from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import Adam
model = Sequential()
model.add(Dense(p, activation="relu", input_shape=(n_ws*2 + 1,), kernel_initializer='he_normal'))
for _ in range(q-1):
    model.add(Dense(p, activation="relu", kernel_initializer='he_normal'))
model.add(Dense(1))
model.compile(optimizer=Adam(learning_rate=1e-3), loss='mse', metrics=['mae'])
model.summary()

(2*n_ws+1)*p + p + (q-1)*(p*p+p) + (p*1+1)
```

Model: "sequential_5"

Layer (type)	Output Shape	Param #
dense_13 (Dense)	(None, 8)	48
dense_14 (Dense)	(None, 8)	72
dense_15 (Dense)	(None, 1)	9
Total params: 129 (516.00 Byte)		
Trainable params: 129 (516.00 Byte)		
Non-trainable params: 0 (0.00 Byte)		

Out[80]: 129

LSTM

```
In [21]: from keras.models import Sequential
from keras.layers import Dense, LSTM
from keras.optimizers import Adam

n_ws = 8
p = 3
q = 2

model = Sequential()
model.add(LSTM(p, activation="tanh", input_shape=(None,n_ws+1),return_sequences=True))
for _ in range(q-1):
    model.add(LSTM(p, activation="tanh",return_sequences=True))
model.add(Dense(1))
model.compile(optimizer=Adam(learning_rate=1e-3), loss='mse', metrics=['mae'])
model.summary()
```

```
4*(p*(n_ws+1) + p*p + p) + (q-1)*4*(p*p+p*p+p)+(p+1)
```

Model: "sequential_8"

Layer (type)	Output Shape	Param #
lstm_16 (LSTM)	(None, None, 3)	156
lstm_17 (LSTM)	(None, None, 3)	84
dense_8 (Dense)	(None, None, 1)	4
=====		
Total params: 244 (976.00 Byte)		
Trainable params: 244 (976.00 Byte)		
Non-trainable params: 0 (0.00 Byte)		

Out[21]: 244

General

Benchmark

```
In [ ]: plot_params = [
    ('MLP' , 'dir' ,150,(3,8,2) , '#d62728'),
    ('MLP' , 'incr',150,(8,8,3) , '#1f77b4'),
    ('LSTM', 'dir' ,50 ,(8,3,2), '#2ca02c'),
    ('LSTM', 'incr',100,(10,3,2) , '#9467bd'),
]
y_true = np.load('data/y_benchmark.npy')
u       = np.load('data/u_benchmark.npy')

plt.figure(figsize=(14,4),dpi=300)
plt.plot(y_true,color='#000000',label='true', linewidth=2)

for nn_type,pred_type,epochs,size,color in plot_params:

    n_ws,p,q = size

    if pred_type == 'dir' and nn_type == 'MLP':
        dyn_sys = MLP_Direct(n_ws,p,q)
    if pred_type == 'incr' and nn_type == 'MLP':
        dyn_sys = MLP_Increment(n_ws,p,q)
    if pred_type == 'dir' and nn_type == 'LSTM':
        dyn_sys = LSTM_Direct(n_ws,p,q)
    if pred_type == 'incr' and nn_type == 'LSTM':
        dyn_sys = LSTM_Increment(n_ws,p,q)
    dyn_sys.reset_weights(epochs)

    y_pred = dyn_sys.simulate(y_true,u)
    plt.plot(y_pred,color=color,ls='--',label=f'({n_ws-1},{p},{q})')
```

```
plt.ylabel(r'$\sigma_n$', fontsize=14)
plt.legend()
plt.grid(ls='--', c='lightgray')
plt.xlabel(r'$n$', fontsize=14)
```

Test

```
In [ ]: plot_params = [
    ('MLP', 'dir', 50, (5, 3, 2), '#d62728'),
    ('MLP', 'incr', 200, (8, 8, 3), '#1f77b4'),
    ('LSTM', 'dir', 100, (5, 10, 2), '#2ca02c'),
    ('LSTM', 'incr', 100, (5, 8, 2), '#9467bd'),
]

i = 10
y_true = np.load('data/y_list.npy')[i]
u = np.load('data/u_list.npy')[i]

plt.figure(figsize=(14, 4), dpi=300)
plt.plot(y_true, color='#000000', label='true', linewidth=2)

for nn_type, pred_type, epochs, size, color in plot_params:
    n_ws, p, q = size

    if pred_type == 'dir' and nn_type == 'MLP':
        dyn_sys = MLP_Direct(n_ws, p, q)
    if pred_type == 'incr' and nn_type == 'MLP':
        dyn_sys = MLP_Increment(n_ws, p, q)
    if pred_type == 'dir' and nn_type == 'LSTM':
        dyn_sys = LSTM_Direct(n_ws, p, q)
    if pred_type == 'incr' and nn_type == 'LSTM':
        dyn_sys = LSTM_Increment(n_ws, p, q)
    dyn_sys.reset_weights(epochs)

    y_pred = dyn_sys.simulate(y_true, u)
    plt.plot(y_pred, color=color, ls='--', label=f'({n_ws-1}, {p}, {q})')

plt.ylabel(r'$\sigma_n$', fontsize=14)
plt.legend()
plt.grid(ls='--', c='lightgray')
plt.xlabel(r'$n$', fontsize=14)
```