neb_ap3

June 14, 2022

José Geraldo Fernandes

Sistemas Fuzzy Adaptativos

1 Implementação:

Arquitetura ANFIS:

[1]:

```
class anfis:
    def __init__(self, n, m):
        self.n = n
        self.c = np.random.randn(m, self.n)
        self.s = np.random.randn(m, self.n)
        self.P = np.random.randn(m, self.n)
        self.q = np.random.randn(self.n)
        self.log = None

    def _forward(self, x, regra):
        mu = np.zeros(shape = x.shape)
        y = self.q[regra]
        for j, x_j in enumerate(x):
            arg = (x_j - self.c[j, regra]) / self.s[j, regra]
            mu[j] = np.exp(-1 / 2 * arg**2)
            y += self.P[j, regra] * x_j
        w = np.product(mu, axis = 0)
        return w, y

    def forward(self, x):
        w = np.zeros(self.n)
        y = np.zeros(self.n)
        for regra in range(self.n):
            w[regra], y[regra] = self._forward(x, regra)
        b = np.sum(w)
        yhat = np.sum(y * w) / b
        return yhat, y, w, b

    def fit(self, X, y, max_epochs = 100, alpha = 0.01):
        self.log = []
        for epoch in tqdm(range(max_epochs)):
            for i, x_i in enumerate(X):
                yhat, Y, W, b = self.forward(x_i)
                de_dyhat = yhat - y[i]
                # print('erro: {}, y: {}, yhat: {}'.format(de_dyhat, y[i],␣
↪yhat))

                dyhat_dW = np.zeros(self.n)
                dyhat_dY = np.zeros(self.n)
                for regra in range(self.n):
                    dyhat_dW[regra] = (Y[regra] - yhat) / b
                    dyhat_dY[regra] = W[regra] / b

                    for j, x_j in enumerate(x_i):
                        dW_dc = W[regra] * (x_j - self.c[j, regra]) / self.s[j,␣
↪regra]**2
                        dW_ds = W[regra] * (x_j - self.c[j, regra])**2 / self.
↪s[j, regra]**3
                        dY_dP = x_j

                        # update
                        self.c[j, regra] = self.c[j, regra] - alpha * de_dyhat␣
↪* dyhat_dW[regra] * dW_dc
                        self.s[j, regra] = self.s[j, regra] - alpha * de_dyhat␣
↪* dyhat_dW[regra] * dW_ds
                        self.P[j, regra] = self.P[j, regra] - alpha * de_dyhat␣
↪* dyhat_dY[regra] * dY_dP
                    self.q[regra] = self.q[regra] - alpha * de_dyhat *␣
↪dyhat_dY[regra]
            self.log.append(self.mse(X, y))

    def mse(self, X, y):
        yhat = self.predict(X).reshape(-1, 1)
        return np.square(y - yhat).mean()

    def predict(self, X):
        W = np.zeros(shape = (X.shape[0], self.n))
        Y = np.zeros(shape = (X.shape[0], self.n))
        for i, x_i in enumerate(X):
            for regra in range(self.n):
                W[i, regra], Y[i, regra] = self._forward(x_i, regra)
        yhat = np.sum(Y * W, axis = 1) / np.sum(W, axis = 1)
        return yhat
```

Arquitetura NFN:

[2]:

```python
class nfn:
    def __init__(self, N = 100):
        self.w_i = None
        self.w_s = None
        self.N = N
        self.log = None
        self.delta = None
        self.minimo = None
        self.maximo = None

    def forward(self, x, index):
        y = 0
        mu = np.zeros(len(x))
        for j, x_j in enumerate(x):
            offset = (x_j // (2 * self.delta[j])) * 2 * self.delta[j]
            reta = (x_j - offset) / self.delta[j]
            if reta > 1:
                reta = reta - 1
            mu[j] = reta
            indice = index[j]
            try:
                w1 = self.w_i[j, indice]
            except IndexError:
                w1 = 0.5
            try:
                w2 = self.w_s[j, indice + 1]
            except IndexError:
                w2 = 0.5
            y += w1 * mu[j] + w2 * (1 - mu[j])
        return y, mu

    def fit(self, X, y, max_epochs = 100, alpha = 0.01):
        self.log = []
        m = X.shape[1]
        self.w_i = np.zeros(shape = (m, self.N))
        self.w_s = np.zeros(shape = (m, self.N))
        self.delta = np.zeros(m)
        self.minimo = np.zeros(m)
        self.maximo = np.zeros(m)
        for j in range(m):
            self.minimo[j] = np.min(X[:, j])
            self.maximo[j] = np.max(X[:, j])
            self.delta[j] = (self.maximo[j] - self.minimo[j]) / 2 / self.N
        for epoch in tqdm(range(max_epochs)):
            X, y = shuffle(X, y)
            for i, x_i in enumerate(X):
                index = []
                for j in range(m):
                    offset = (x_i[j] // (2 * self.delta[j])) * 2 * self.delta[j]
                    index.append(int((offset - self.minimo[j]) // (2 * self.
↪delta[j])))
                yhat, mu = self.forward(x_i, index)
                de_dyhat = yhat - y[i]

                if alpha == 'auto':
                    den = 0
                    for j in range(m):
                        den += mu[j]**2 + (1 - mu[j])**2
                    alpha = 1/den

                for j in range(m):
                    dyhat_dw = mu[j]
                    # update
                    indice = index[j]
                    try:
                        self.w_i[j, indice] = self.w_i[j, indice] - alpha *␣
↪de_dyhat * dyhat_dw
                    except IndexError:
                        pass
                    try:
                        self.w_s[j, indice + 1] = self.w_s[j, indice + 1] -␣
↪alpha * de_dyhat * (1 - dyhat_dw)
                    except IndexError:
                        pass
            self.log.append(self.mse(X, y))

    def mse(self, X, y):
        yhat = self.predict(X).reshape(-1, 1)
        return np.square(y - yhat).mean()

    def predict(self, X):
        m = X.shape[1]
        yhat = []
        for i, x_i in enumerate(X):
            index = []
            for j in range(m):
                offset = (x_i[j] // (2 * self.delta[j])) * 2 * self.delta[j]
                index.append(int((offset - self.minimo[j]) // (2 * self.
↪delta[j])))
            y, _ = self.forward(x_i, index)
            yhat.append(y)
        return np.array(yhat)
```

Bibliotecas

[3]:

```
import numpy as np
import matplotlib.pyplot as plt
from tqdm import tqdm
from sklearn.utils import shuffle
import pandas as pd
from sklearn import preprocessing
```

2 Problema 1

[4]:

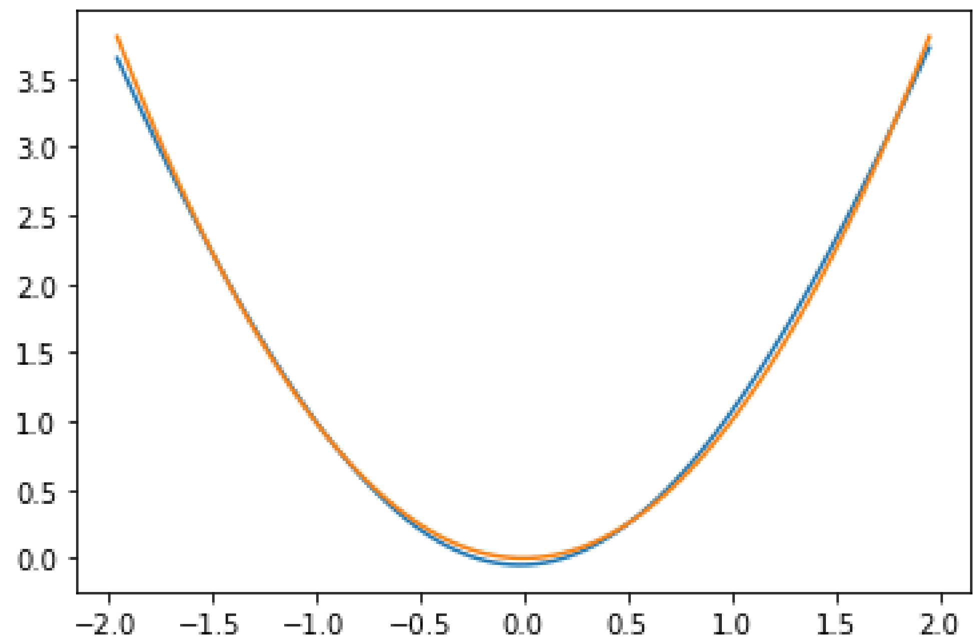
```
# input
N = 100
X_test = np.linspace(-1.95, 1.95, N).reshape(-1, 1)
y_test = X_test ** 2

# shuffle
X_train = np.random.uniform(low = -2, high = 2, size = 9*N).reshape(-1, 1)
y_train = X_train ** 2
```

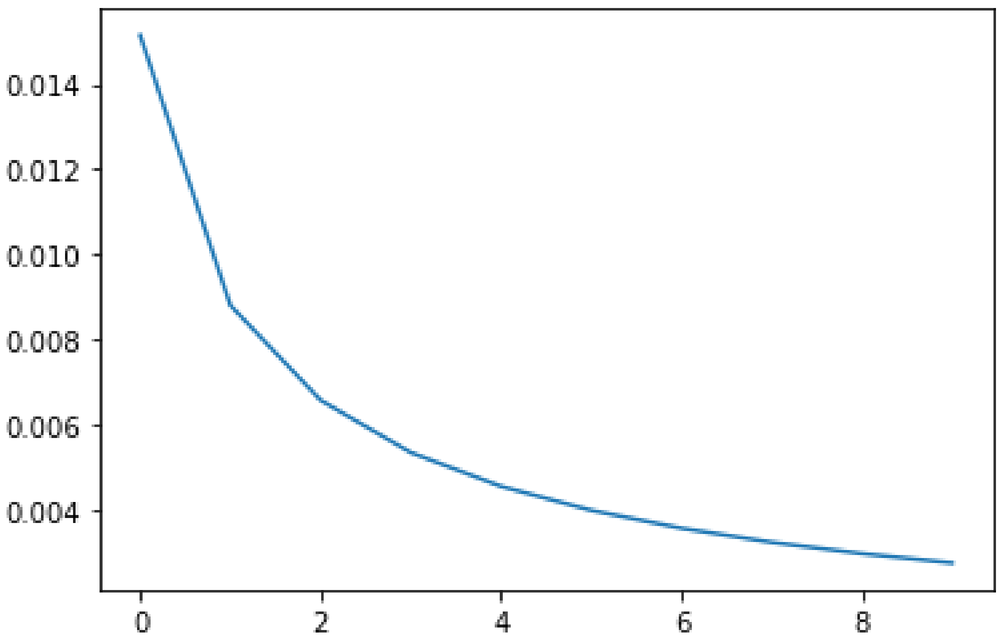
2.1 ANFIS

[5]:

```
# anfis
n = 2
model = anfis(n = n, m = X_train.shape[1])
model.fit(X_train, y_train, alpha = 0.1, max_epochs = 10)

# eval
yhat = model.predict(X_test).reshape(-1, 1)
mse = model.mse(X_test, y_test)
epm = (np.abs(y_test - yhat) / yhat).mean()
print('mse: {}, epm: {}'.format(mse, epm))

# plot
plt.figure()
xx, yy = zip(*sorted(zip(X_test, yhat)));
plt.plot(xx, yy)
xx, yy = zip(*sorted(zip(X_test, y_test)));
plt.plot(xx, yy)

# log
plt.figure()
plt.plot(model.log);
```

```
100%|
                         | 10/10 [00:01<00:00,  8.07it/s]

mse: 0.0026139738748035385, epm: -0.1796331381959637
```


3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

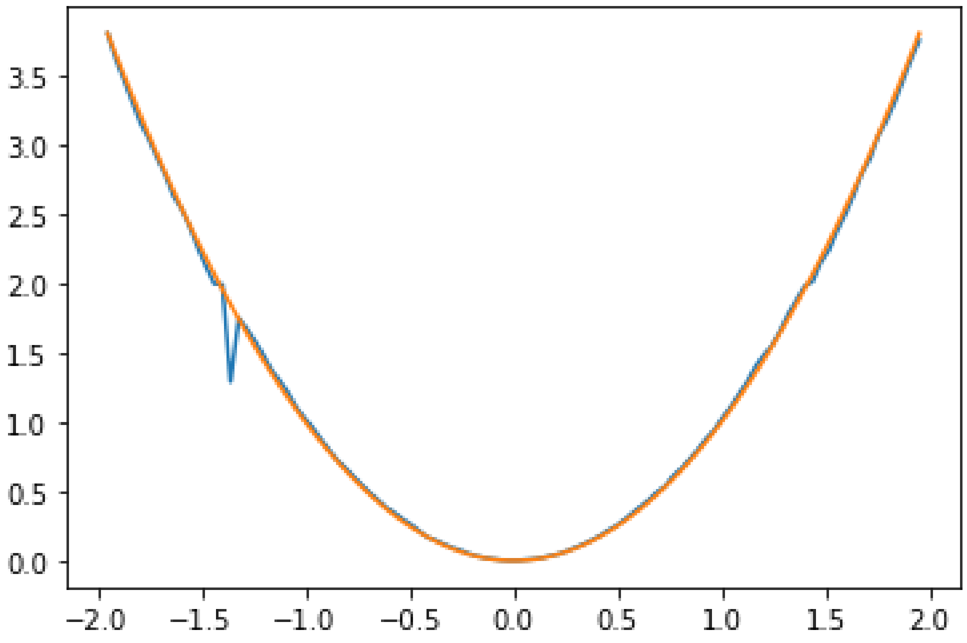

0.014
0.012
0.010
0.008
0.006
0.004
0
2
4
6
8

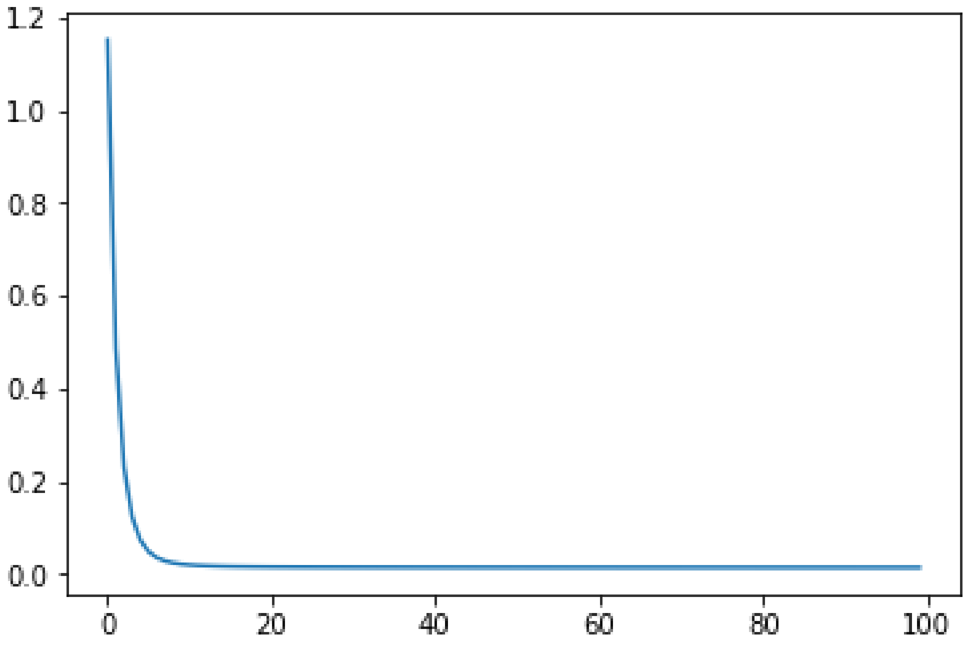
2.2 NFN

[6]:

```
# nfn
model = nfn(N = 100)
model.fit(X_train, y_train, alpha = 0.1, max_epochs = 100)

# eval
yhat = model.predict(X_test).reshape(-1, 1)
mse = model.mse(X_test, y_test)
epm = (np.abs(y_test - yhat) / yhat).mean()
print('mse: {}, epm: {}'.format(mse, epm))

# plot
plt.figure()
xx, yy = zip(*sorted(zip(X_test, yhat)))
plt.plot(xx, yy);
xx, yy = zip(*sorted(zip(X_test, y_test)))
plt.plot(xx, yy);

# log
plt.figure()
plt.plot(model.log);
```

```
100%|
                   | 100/100 [00:02<00:00, 38.96it/s]

mse: 0.003821442595344005, epm: 0.05469431044872585
```





3 Problema 2

[7]:

```python
def f(x):
    return (1 + x[0]**(0.5) + x[1]**(-1) + x[2]**(-1.5))**2

# input
X_train = np.zeros(shape = (216, 3))
ind = 0
for i in range(1, 7):
    for j in range(1, 7):
        for k in range(1, 7):
            X_train[ind, 0] = i
            X_train[ind, 1] = j
            X_train[ind, 2] = k
            ind += 1
# shuffle
random = np.random.permutation(X_train.shape[0])
X = X_train[random]
y_train = np.array([f(x) for x in X_train]).reshape(-1, 1)
X_test = np.zeros(shape = (125, 3))
ind = 0
for i in range(1, 6):
    for j in range(1, 6):
        for k in range(1, 6):
            X_test[ind, 0] = i + 0.5
            X_test[ind, 1] = j + 0.5
            X_test[ind, 2] = k + 0.5
            ind += 1
y_test = np.array([f(x) for x in X_test]).reshape(-1, 1)
```

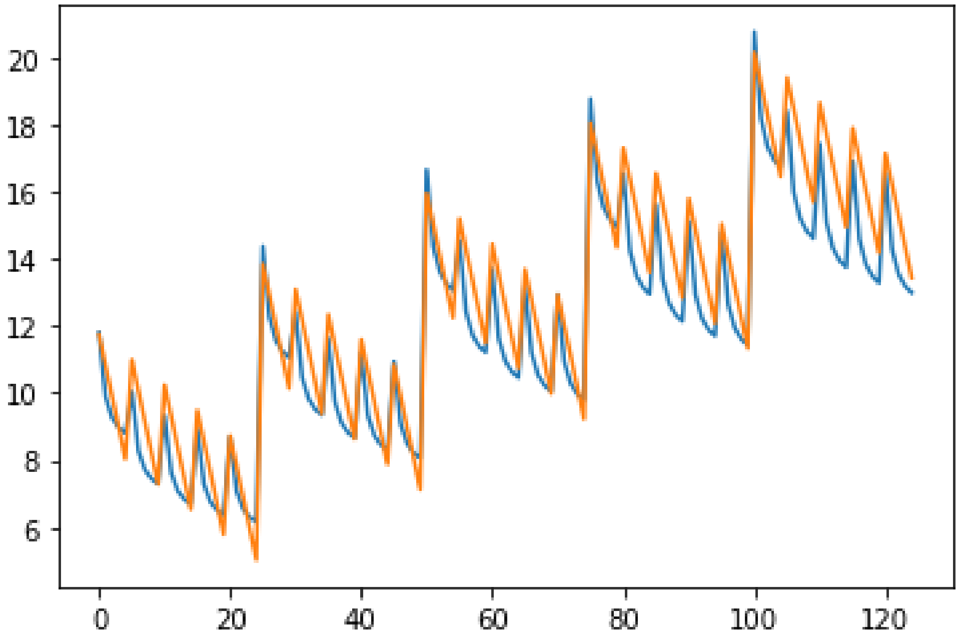
3.1 ANFIS

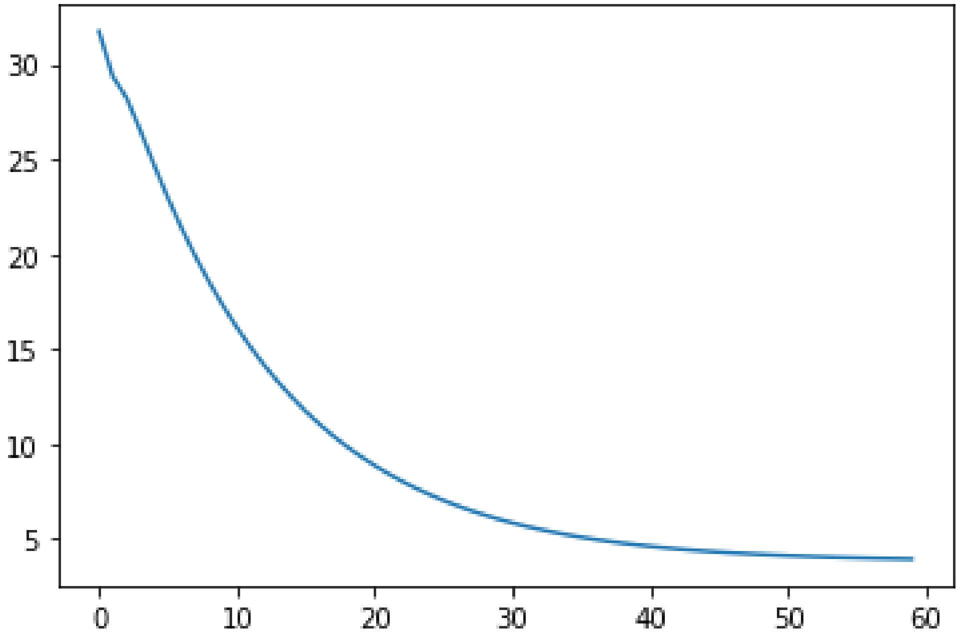
[8]:

```
# anfis
n = 8
model = anfis(n = n, m = X_train.shape[1])
model.fit(X_train, y_train, alpha = 0.01, max_epochs = 60)

# report
yhat = model.predict(X_test).reshape(-1, 1)
mse = model.mse(X_test, y_test)
epm = (np.abs(y_test - yhat) / yhat).mean()
print('mse: {}, epm: {}'.format(mse, epm))

# plot
plt.figure()
plt.plot(y_test);
plt.plot(yhat);

# log
plt.figure()
plt.plot(model.log);
```

```
100%|
                      | 60/60 [00:11<00:00,  5.37it/s]
```

```
mse: 1.3112993803605837, epm: 0.07533537447702036
```


20
18
16
14
12
10
8
6
0
20
40
60
80
100
120


30
25
20
15
10
5
0
10
20
30
40
50
60

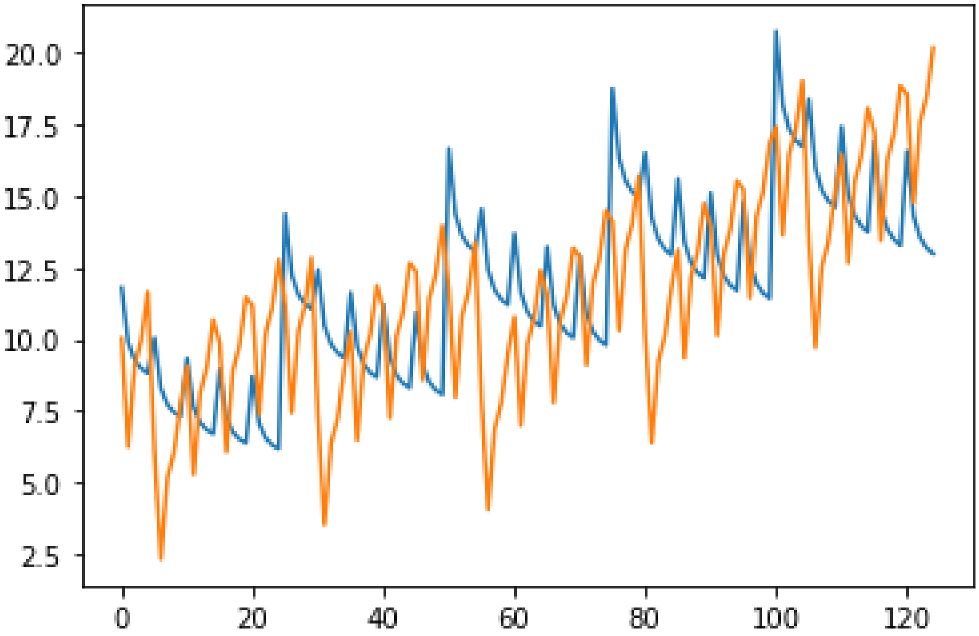
3.2 NFN

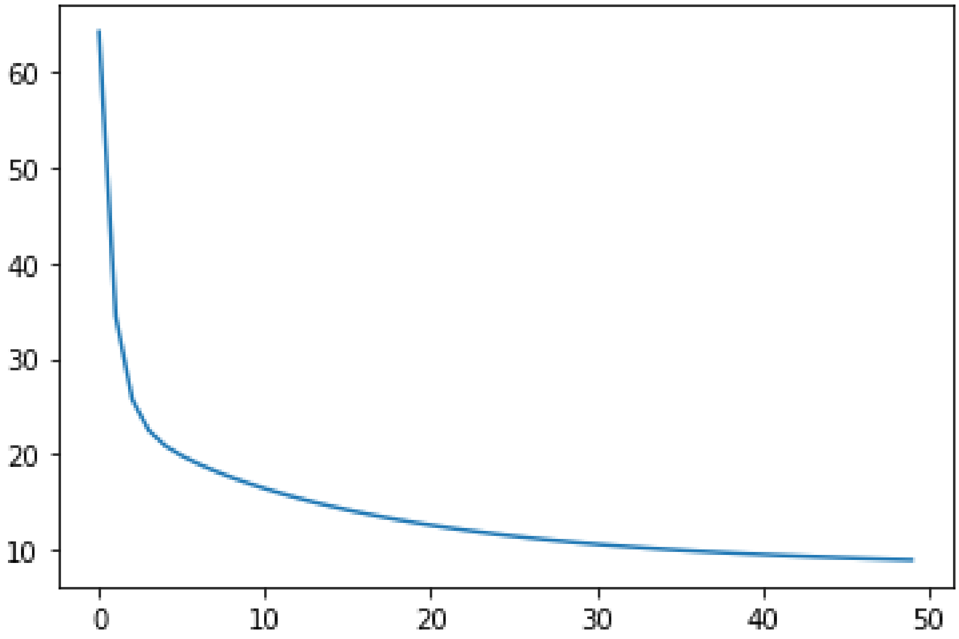
[9]:

```
# nfn
model = nfn(N = 4)
model.fit(X_train, y_train, alpha = 0.01, max_epochs = 50)

# eval
yhat = model.predict(X_test).reshape(-1, 1)
mse = model.mse(X_test, y_test)
epm = (np.abs(y_test - yhat) / (yhat + 0.1)).mean()
print('mse: {}, epm: {}'.format(mse, epm))

# plot
plt.figure()
plt.plot(y_test);
plt.plot(yhat);

# log
plt.figure()
plt.plot(model.log);
```

```
100%|
                     | 50/50 [00:00<00:00, 61.93it/s]
```

```
mse: 11.350998710345769, epm: 0.29420402152777964
```





4 Problema 3

[10]:
```
def g(x):
    num = x[0] * x[1] * x[2] * x[4] * (x[2] - 1) + x[3]
    den = 1 + x[2]**2 + x[3]**2
    return num / den

# input
N = 1000
k = np.arange(N)
u = np.sin(2*np.pi * k / 250)
u[k>500] = 0.8 * u[k>500] + 0.2 * np.sin(2*np.pi * k[k>500] / 25)

y = np.zeros(N)
X = np.zeros(shape = (N - 6, 5))
for k in range(2, N - 1):
    x = np.array([y[k], y[k-1], y[k-2], u[k], u[k-1]])
    y[k + 1] = g(x)
    if k > 4:
        X[k - 5] = x
y = np.delete(y, obj = [0, 1, 2, 3, 4, -1], axis = 0)

# kfold
k = 10
size = len(X)
```

```
index = list(range(size))
np.random.shuffle(index)
step = round(size / k)
kfolds = [index[i:i+step] for i in range(0, size, step)]

k = 0
kfold = kfolds[k]
fold = np.ones(size, bool)
fold[kfold] = False

X_test = X[np.invert(fold), :]
X_train = X[fold, :]
y_test = y[np.invert(fold)]
y_train = y[fold]
```

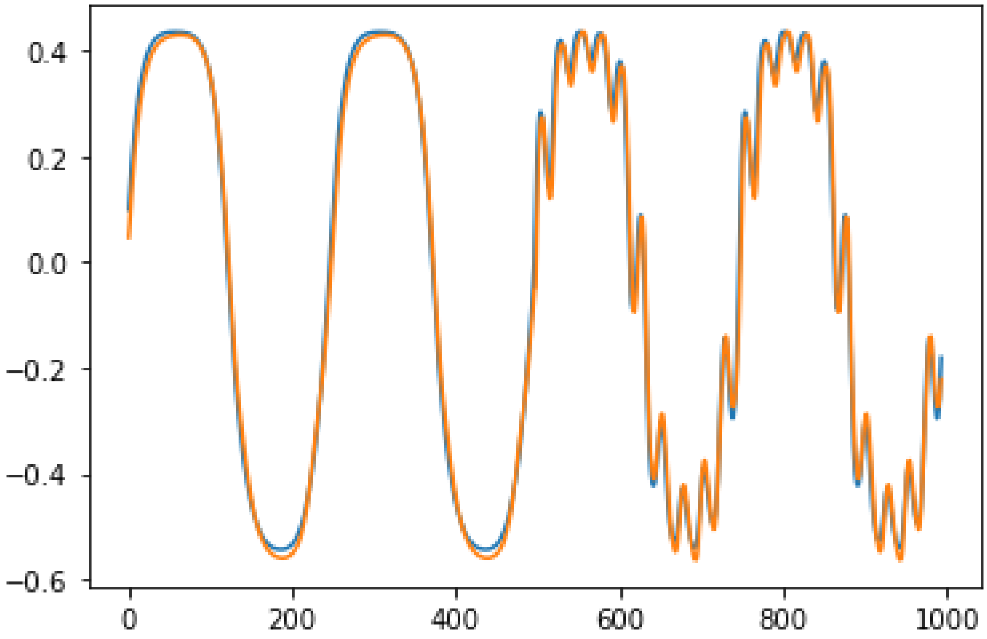
4.1 ANFIS

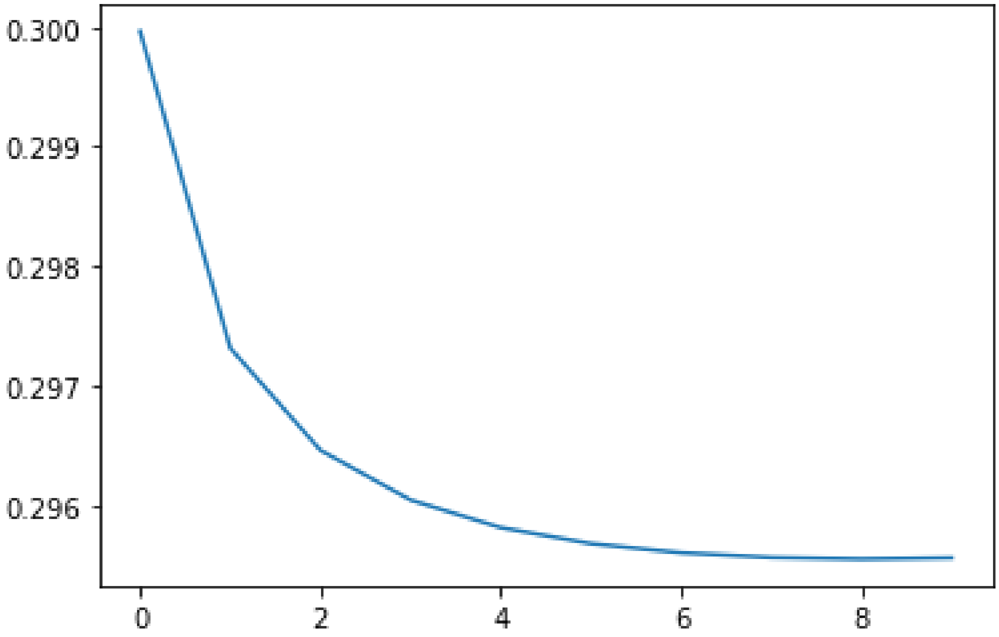
[11]:

```
# anfis
n = 15
model = anfis(n = n, m = X_train.shape[1])
model.fit(X_train, y_train, max_epochs = 10, alpha = 0.01)

# report
yhat = model.predict(X).reshape(-1, 1)
mse = model.mse(X_test, y_test)
epm = (np.abs(y - yhat) / yhat).mean()
print('mse: {}, epm: {}'.format(mse, epm))

# plot
plt.figure()
plt.plot(y);
plt.plot(yhat);

# log
plt.figure()
plt.plot(model.log);
```

```
100%|
                    | 10/10 [00:10<00:00,  1.06s/it]
```

```
mse: 0.27937340177103465, epm: -1.4741521729584146
```


0.4
0.2
0.0
−0.2
−0.4
−0.6
0
200
400
600
800
1000

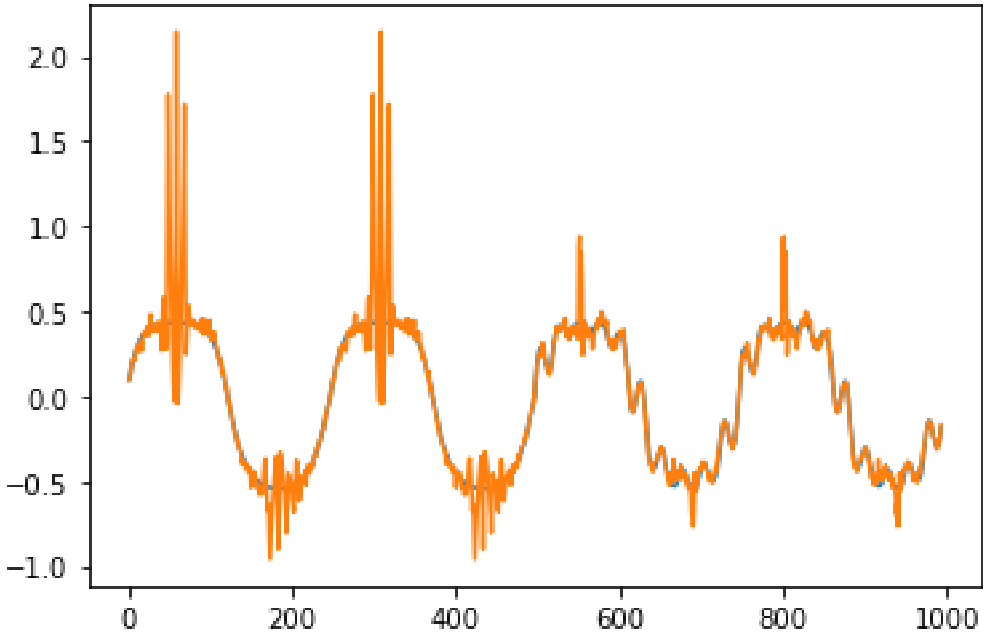

0.300
0.299
0.298
0.297
0.296
0
2
4
6
8

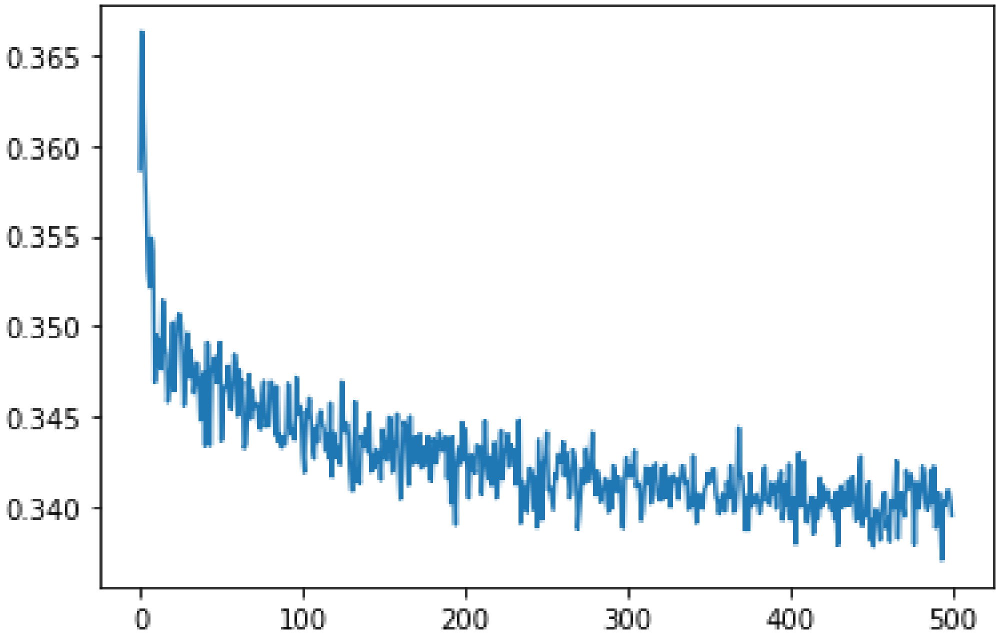
4.2 NFN

```
[23]: # nfn
model = nfn(N = 25)
model.fit(X_train, y_train, alpha = 0.01, max_epochs = 500)

# report
yhat = model.predict(X).reshape(-1, 1)
mse = model.mse(X_test, y_test)
epm = (np.abs(y - yhat) / yhat).mean()
print('mse: {}, epm: {}'.format(mse, epm))

# plot
plt.figure()
plt.plot(y);
plt.plot(yhat);

# log
plt.figure()
plt.plot(model.log);
```

```
100%|
                              | 500/500 [00:30<00:00, 16.24it/s]
```

```
mse: 0.40646736618225693, epm: -0.16977716154736042
```





5 Problema 4

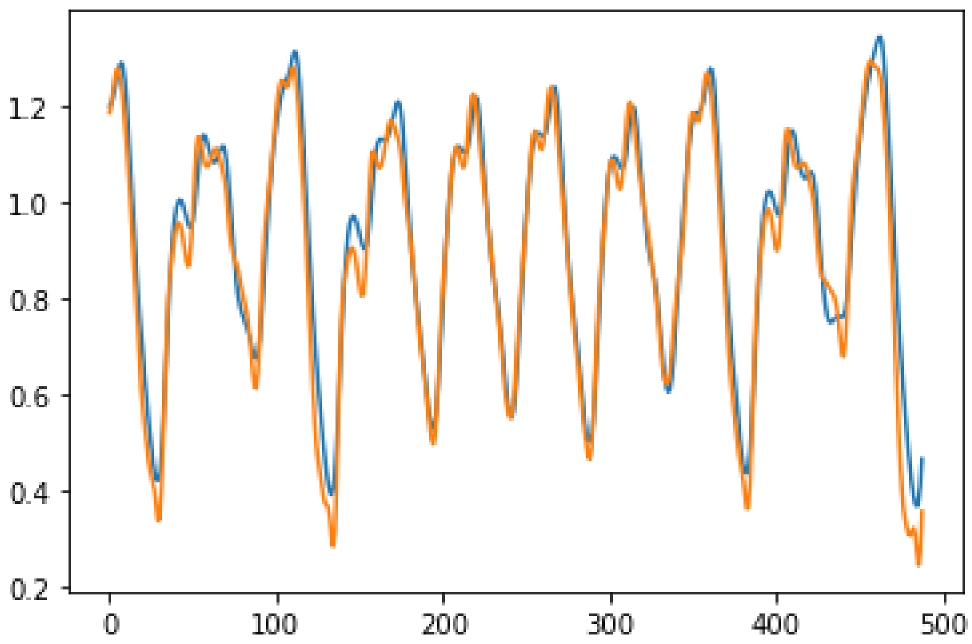
[24]:
```
def mackey_glass(N = 1000):
    b = 0.1
    c = 0.2
    tau = 17

    y = [0.9697, 0.9699, 0.9794, 1.0003, 1.0319, 1.0703, 1.1076, 1.1352, 1.1485,
     1.1482, 1.1383, 1.1234, 1.1072, 1.0928, 1.0820, 1.0756, 1.0739, 1.0759]

    for n in range(17,N+99):
        y.append(y[n] - b*y[n] + c*y[n-tau]/(1+y[n-tau]**10))
    y = y[100:]
    return np.array(y)

# input
N = 1000
data = mackey_glass(N)
y = np.zeros(N - 18 - 6)
X = np.zeros(shape = (N - 18 - 6, 4))
i = 0
for t in range(18, N - 6):
    x = np.array([data[t - 18], data[t - 12], data[t - 6], data[t]])
    X[i] = x
    y[i] = data[t + 6]
    i += 1

# train test
X_train, X_test = np.split(X, 2)
y_train, y_test = np.split(y, 2)
```

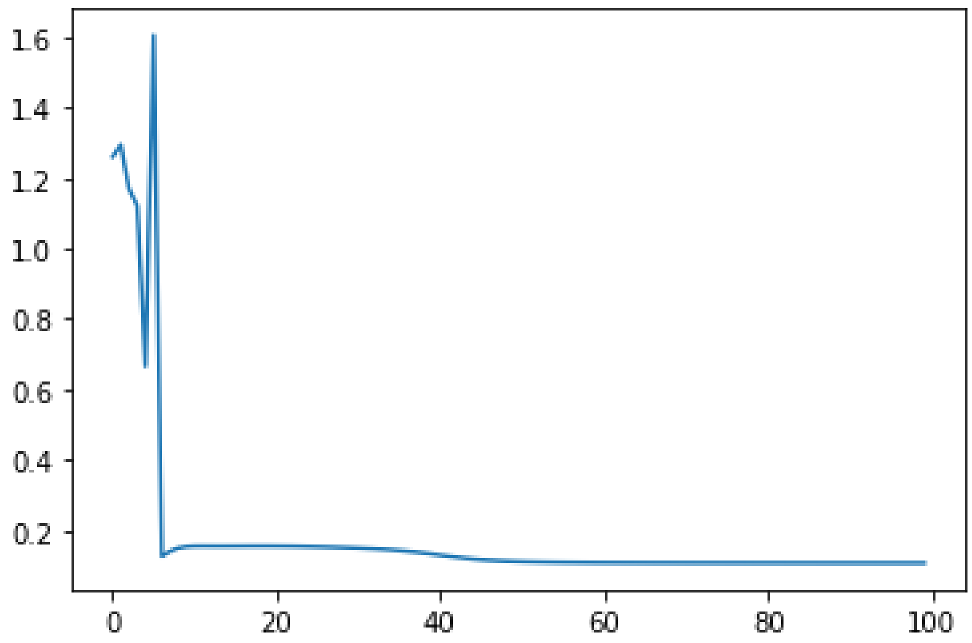
5.1 ANFIS

[26]:

```
# anfis
n = 12
model = anfis(n = n, m = X_train.shape[1])
model.fit(X_train, y_train, max_epochs = 100, alpha = 0.01)

# report
yhat = model.predict(X_test).reshape(-1, 1)
mse = model.mse(X_test, y_test)
epm = (np.abs(y_test - yhat) / yhat).mean()
print('mse: {}, epm: {}'.format(mse, epm))

# plot
plt.plot(y_test);
plt.plot(yhat);

# log
plt.figure()
plt.plot(model.log);
```

```
100%|
                      | 100/100 [00:37<00:00,  2.70it/s]

mse: 0.12685415259878896, epm: 0.41020933628998696
```


1.2
1.0
0.8
0.6
0.4
0.2
0
100
200
300
400
500


1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0
20
40
60
80
100

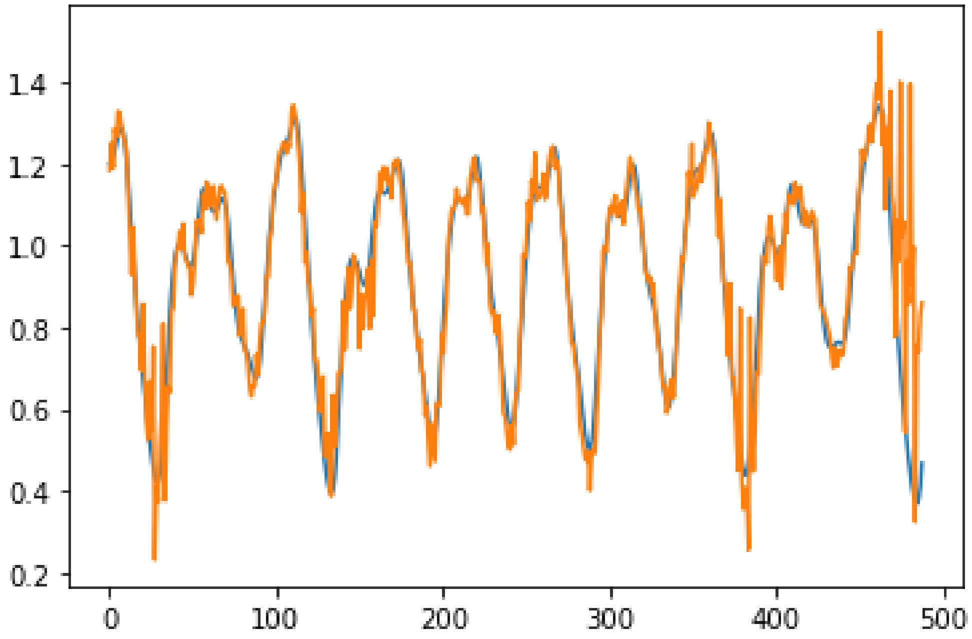
5.2 NFN

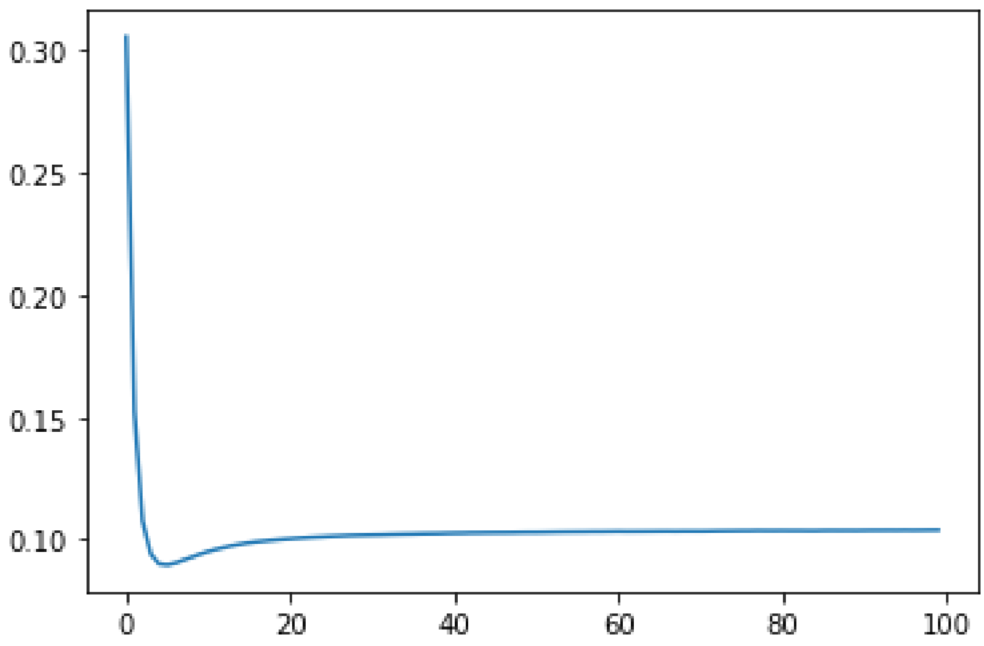
[36]:

```
# nfn
model = nfn(N = 20)
model.fit(X_train, y_train, alpha = 0.01, max_epochs = 100)

# report
yhat = model.predict(X_test).reshape(-1, 1)
mse = model.mse(X_test, y_test)
epm = (np.abs(y_test - yhat) / yhat).mean()
print('mse: {}, epm: {}'.format(mse, epm))

# plot
plt.plot(y_test);
plt.plot(yhat);

# log
plt.figure()
plt.plot(model.log);
```

```
100%|
                   | 100/100 [00:02<00:00, 39.31it/s]
```

```
mse: 0.11776255720181003, epm: 0.3478951281836752
```





6 Problema 5

https://archive.ics.uci.edu/ml/datasets/Concrete+Slump+Test

[37]:
```
# input
# https://archive.ics.uci.edu/ml/datasets/Concrete+Slump+Test
data = pd.read_csv('slump_test.data').values[:, 1:]

# train test
k = 10
size = len(data)
index = list(range(size))
np.random.shuffle(index)
step = round(size / k)
kfolds = [index[i:i+step] for i in range(0, size, step)]

k = 0
kfold = kfolds[k]
fold = np.ones(size, bool)
fold[kfold] = False

X = data[:, 0:-1]
y = data[:, -1]

X_test = data[np.invert(fold), 0:-1]
```

```
X_train = data[fold, 0:-1]
y_test = data[np.invert(fold), -1]
y_train = data[fold, -1]

# norm
scaler = preprocessing.StandardScaler().fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
X = scaler.transform(X)
```

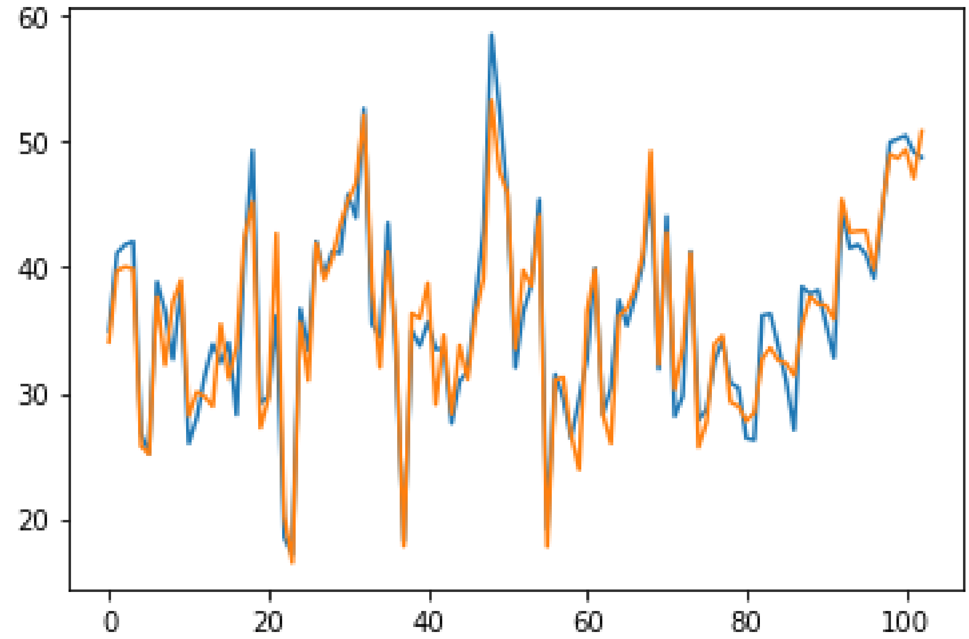
6.1 ANFIS

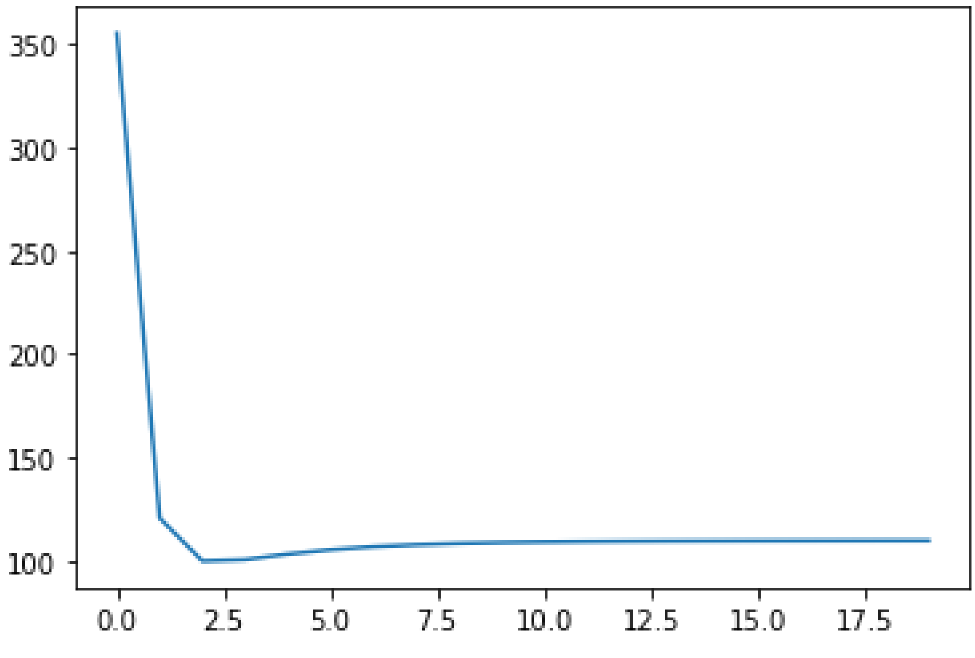
[38]:

```
# anfis
n = 32
model = anfis(n = n, m = X_train.shape[1])
model.fit(X_train, y_train, max_epochs = 20, alpha = 0.01)

# report
yhat = model.predict(X).reshape(-1, 1)
mse = model.mse(X_test, y_test)
epm = (np.abs(y - yhat) / yhat).mean()
print('mse: {}, epm: {}'.format(mse, epm))

# plot
plt.figure()
plt.plot(y);
plt.plot(yhat);

# log
plt.figure()
plt.plot(model.log);
```

```
100%|
                   | 20/20 [00:06<00:00,  2.86it/s]
```

```
mse: 197.10290913234897, epm: 0.2605227550864596
```

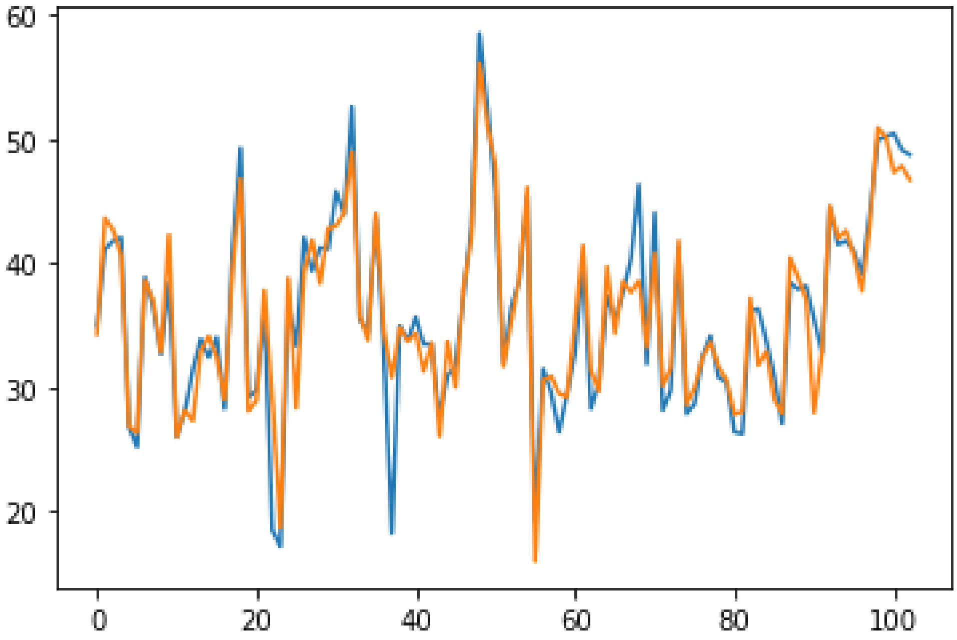

60
50
40
30
20
0
20
40
60
80
100

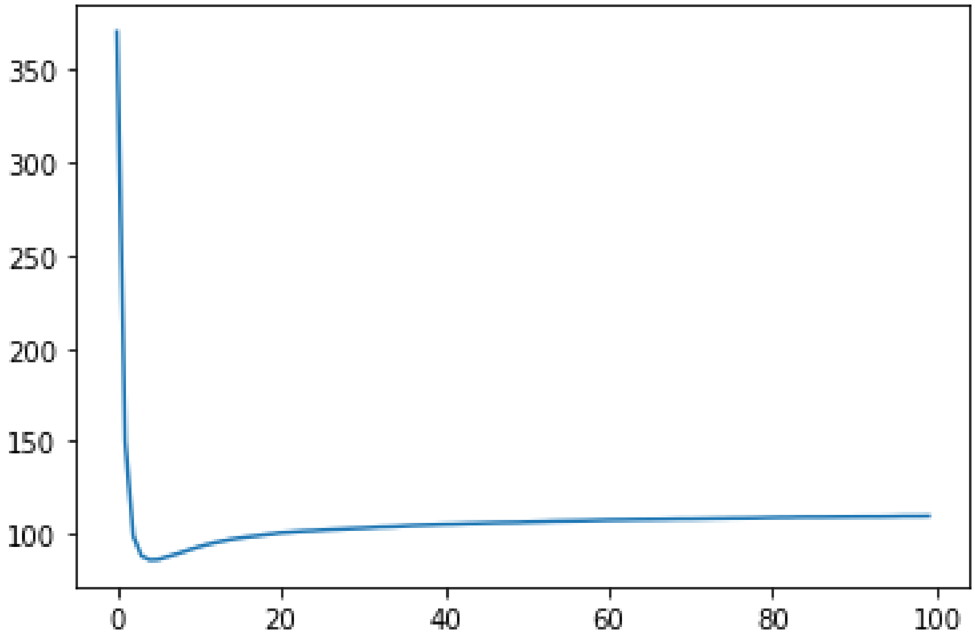

350
300
250
200
150
100
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

6.2 NFN

```
[39]: # nfn
model = nfn(N = 10)
model.fit(X_train, y_train, alpha = 0.01, max_epochs = 100)

# report
yhat = model.predict(X).reshape(-1, 1)
mse = model.mse(X_test, y_test)
epm = (np.abs(y - yhat) / yhat).mean()
print('mse: {}, epm: {}'.format(mse, epm))

# plot
plt.figure()
plt.plot(y);
plt.plot(yhat);

# log
plt.figure()
plt.plot(model.log);
```

```
100%|
                              | 100/100 [00:01<00:00, 99.23it/s]

mse: 129.55020087403355, epm: 0.24559001265626665
```





7 Comentário

Os modelos implementados parecem ser instáveis em função da alta frequência da série de previsão, principalmente o modelo NFN. Por outro lado, o custo computacional é muito menor, dos modelos NFN, visto o tempo de execução observável.