Московский авиационный институт (национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №2 по курсу «Нейроинформатика»

Студент: К. О. Вахрамян
Преподаватель: Н. П. Аносова
Группа: М8О-406Б

Дата:
Оценка:
Подпись:

Москва, 2021

Лабораторная работа №2

Задача: Целью работы является исследование свойств линейной нейронной сети и алгоритмов ее обучения, применение сети в задачах аппроксимации и фильтрации.

Основные этапы работы:

1. Использовать линейную нейронную сеть с задержками для аппроксимации функции. В качестве метода обучения использовать адаптацию.
2. Использовать линейную нейронную сеть с задержками для аппроксимации функции и выполнения многошагового прогноза.
3. Использовать линейную нейронную сеть в качестве адаптивного фильтра для подавления помех. Для настройки весовых коэффициентов использовать метод наименьших квадратов.

Вариант 2:

P	T
$x = \sin\left(\frac{1}{2}t^2 - 5t\right), \quad t \in [0, 2.2],\ h = 0.01$ $x = \sin\left(-3t^2 + 5t + 10\right), \quad t \in [0, 2.5],\ h = 0.01$	$y = \frac{1}{3}\sin\left(-3t^2 + 5t - 3\right)$

1 Описание

Опишем класс сети ADALINE

```
class ADALINE:
    def __init__(self, steps = 50, lr = 0.0001, stop_err=0.0):
        self.steps = steps
        self.w = None
        self.rate = lr
        self.stop_err = stop_err

    def fit(self, X, y):
        X_t = np.append(X, np.ones((X.shape[0], 1)), axis = 1)
        y_t = np.array(y)
        if self.w is None:
            self.w = np.random.random((X_t.shape[1], y_t.shape[1]))

        for _ in range(self.steps):
            for i in range(X_t.shape[0]):
                e = y_t[i] - X_t[i].dot(self.w)
                self.w += self.rate * X_t[i].reshape(X_t.shape[1], 1).dot(e.reshape(1,
                    y_t.shape[1]))

            mse = ((y_t - X_t.dot(self.w))**2).mean()
            if mse < self.stop_err:
                break

        return self

    def set_steps(self, steps):
        self.steps = steps

    def set_learning_rate(self, rate):
        self.rate = rate


    def predict(self, X):
        X_t = np.append(X, np.ones((X.shape[0], 1)), axis = 1)
        return X_t.dot(self.w)

    def weights(self):
        return self.w[:-1]

    def bias(self):
        return self.w[-1]


    def score(self, X, y):
        X_t = np.append(X, np.ones((X.shape[0], 1)), axis = 1)
        y_t = np.array(y)
```

```
46          return ((y_t - X_t.dot(self.w))**2).mean()**0.5
```

Линия задержки с отводами

```
1  class TDL:
2      def __init__(self, D = 1, pad_zeros=True):
3          self.depth = D
4          self.padding = pad_zeros
5          self.queue = np.zeros(D)
6
7      def fit(self, X, Y = None):
8          if self.padding:
9              in_arr = np.append(np.zeros(self.depth - 1), X)
10             result = np.zeros((len(X) - 1, self.depth))
11             if Y is None:
12                 Y = X[-len(X) + 1:]
13             else:
14                 Y = Y[-len(X) + 1:]
15         else:
16             if len(X) < self.depth:
17                 return None
18             in_arr = np.array(X)
19             result = np.zeros((len(X) - self.depth, self.depth))
20             if Y is None:
21                 Y = X[-len(X) + self.depth:]
22             else:
23                 Y = Y[-len(X) + self.depth:]
24
25         for i in range(in_arr.shape[0] - self.depth):
26             result[i] = in_arr[i:i + self.depth]
27
28         return result, Y
29
30     def tdl_init(self, values):
31         self.queue = np.append(np.zeros(1), np.array(values))
32
33
34     def tdl_init_zeros(self):
35         self.queue = np.zeros(self.depth)
36
37
38     def predict(self, X):
39         in_arr = np.append(self.queue[1:], X)
40         result = np.zeros((len(X), self.depth))
41
42         for i in range(in_arr.shape[0] - self.depth + 1):
43             result[i] = in_arr[i:i + self.depth]
44         self.queue = in_arr[-self.depth:]
45         return result
```

Класс, который объединяет в себе сеть и линия задержки с отводами:

```
class Filtrator:
    def __init__(self, D = 1, pad_zeros = False, steps = 50, l_r=0.001, stop=0.0):
        self.tdl = TDL(D, pad_zeros)
        self.linlr = ADALINE(steps, l_r, stop)
        self.tld_initialized = pad_zeros
        self.last_predict = None

    def fit(self, X, Y = None):
        X1, Y1 = self.tdl.fit(X, Y)
        Y1 = np.array(Y1).reshape(len(Y1), 1)
        self.linlr.fit(X1, Y1)
        return self

    def tdl_init(self, values):
        self.tdl.tdl_init(values)
        self.tld_initialized = True

    def tdl_init_zeros(self):
        self.tdl.tdl_init_zeros()
        self.tld_initialized = True

    def predict(self, x):
        ans = self.linlr.predict(self.tdl.predict(x)).ravel()
        self.last_predict = ans[-1]
        return ans

    def display(self):
        return self.tdl.display() + self.linlr.display()

    staticmethod
    def score_value(Y_t, Y_p):
        return ((Y_t - Y_p)**2).mean()**0.5

    def gen_values(self, num, inpt = None):
        if inpt is not None:
            self.last_predict = inpt
        for i in range(num):
            yield self.predict(np.array([self.last_predict]))[0]
```

Задание 1

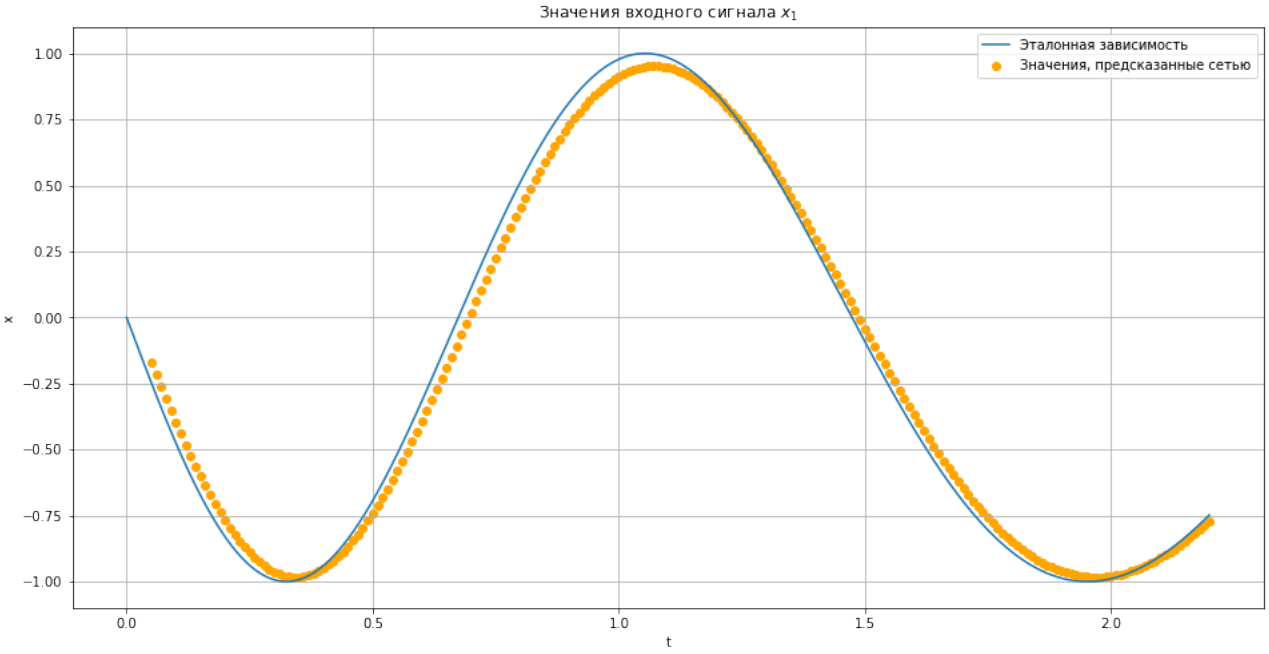
Построим обучающее множество. В качестве входного множества используем значения первого входного сигнала на заданном интервале

```
T = np.append(np.arange(*t1, h1), t1[1])
X = x1(T)
```

Обучим сеть. Скорость обучения 0.01, задержка 5. Число циклов в адаптации - 50.

```
D = 5
steps = 50
learn_rate = 0.01

model = Filtrator(D, False, steps, learn_rate).fit(X)
```



$RMSE = 0.08583976009552448$

Задание 2

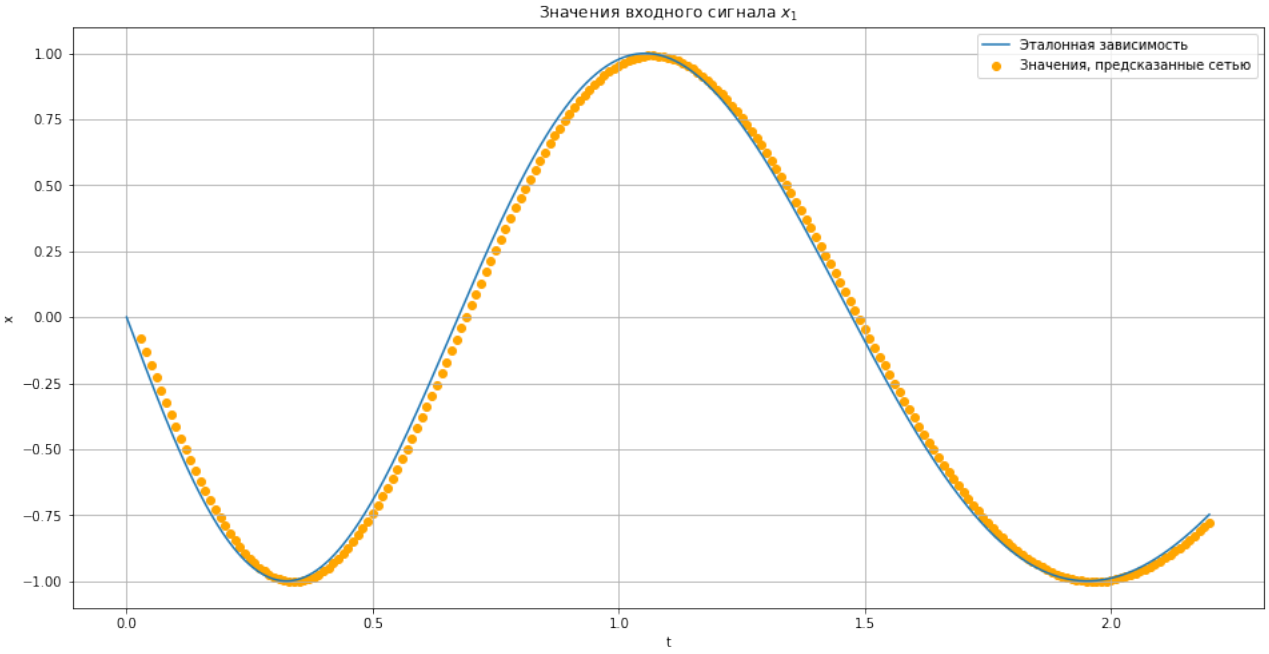
Построить обучающее множество: в качестве входного множества использовать значения первого входного сигнала на заданном интервале

```
T = np.append(np.arange(*t1, h1), t1[1])
X = x1(T)
```

Обучаем сеть. Скорость обучения 0.001, задержка 3. Число эпох в обучении - 600.

```
D = 3
steps = 600
learn_rate = 0.001
stop_val = 10e-6

model = Filtrator(D, False, steps, learn_rate, stop_val).fit(X)
```

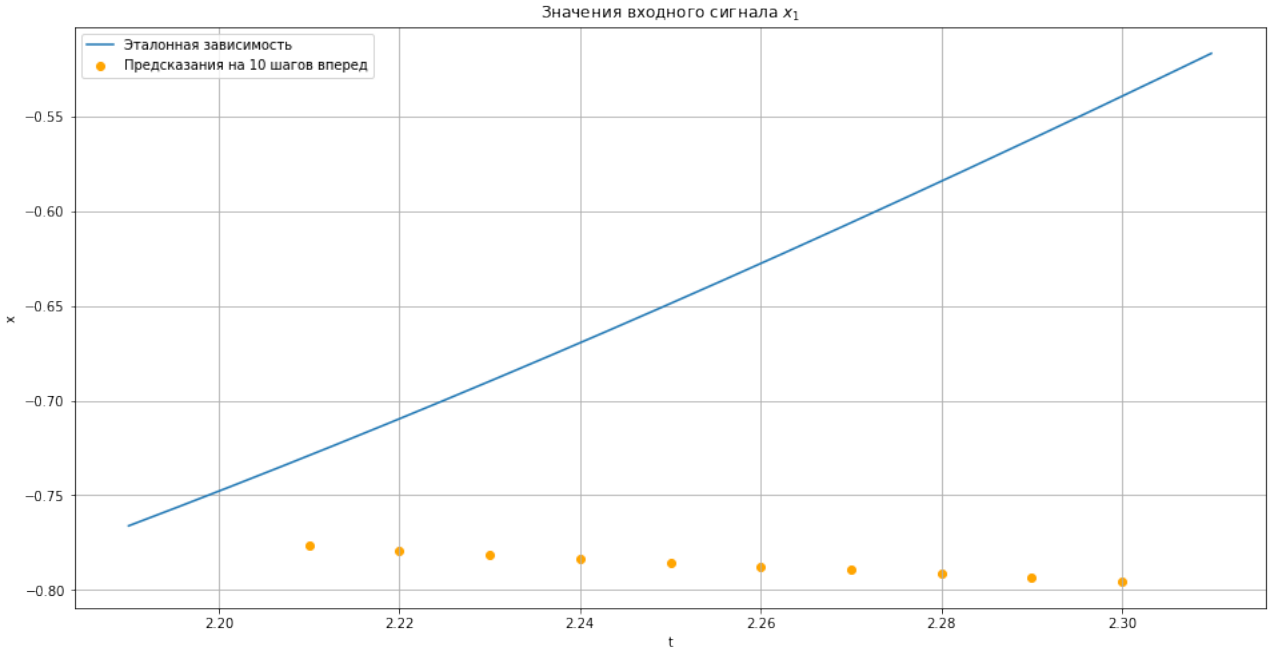


$RMSE = 0.03982275893458956$

Сделаем предсказание за пределы обучающего множества на $K = 10$ шагов вперед и посмотрим на результат.

```
K = 10

X_pred = np.array(list(model.gen_values(K)))
```



$RMSE = 0.16384976421466238$

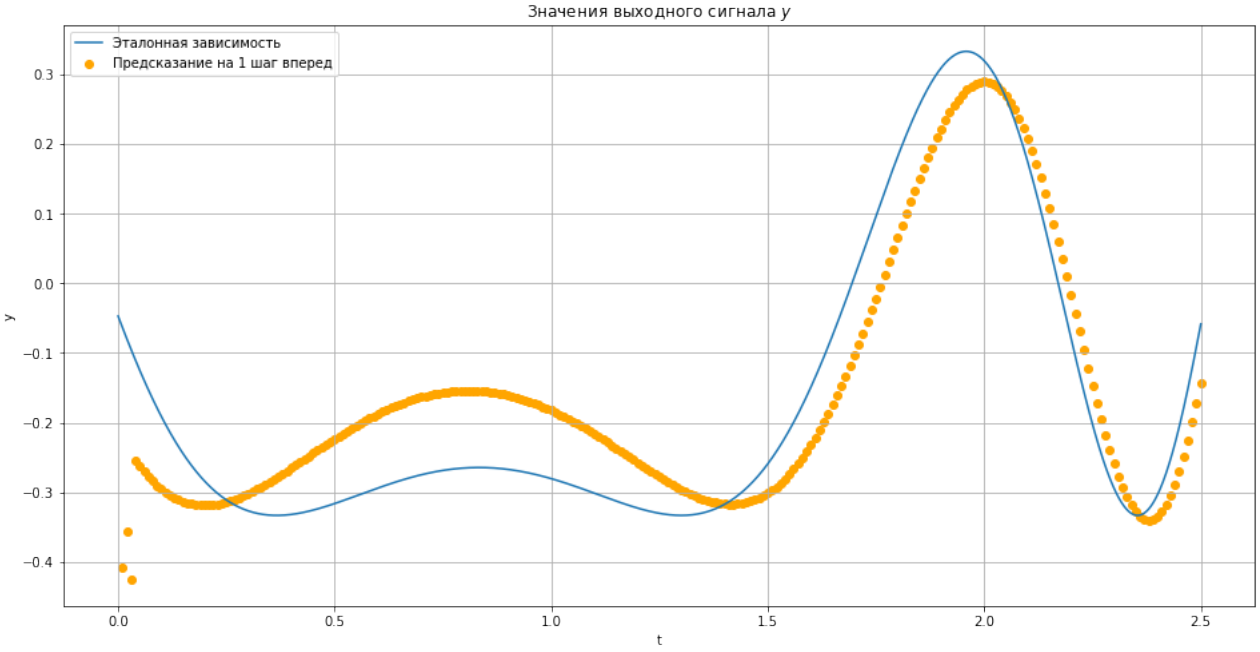
Задание 3

Построим и обучим линейную сеть, которая является адаптивным линейным фильтром. Задачей фильтра является моделирование источника шума, чтобы в последующем удалить помехи из полезного сигнала

```
T = np.append(np.arange(*t2, h2), t2[1])
X = x2(T)
Y = y(T)
```

```
D = 4
steps = 500
learn_rate = 0.0025
stop_val = 10e-6

model = Filtrator(D, True, steps, learn_rate, stop_val).fit(X, Y)
```


Значения выходного сигнала y
Эталонная зависимость
Предсказание на 1 шаг вперед
y
t
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
−0.4
0.0
0.5
1.0
1.5
2.0
2.5

2 Выводы

Алгоритм LMS можно использовать для аппроксимации нелинеинои функции, а также для прогнозирования значении функции, при нелинеинои зависимости.