Московский государственный технический университет им. Н.Э. Баумана
Факультет «Информатика и системы управления»
Кафедра «Системы обработки информации и управления»

Отчет
Лабораторная работа № 6
По курсу «Проектирование интеллектуальных систем»

Вариант 9

ИСПОЛНИТЕЛЬ:
Попов Илья Андреевич
Группа ИУ5-23М

_____

"__"_____2022 г.

ПРЕПОДАВАТЕЛЬ:
Канев А.И.

_____

"__"_____2022 г.

Москва 2022

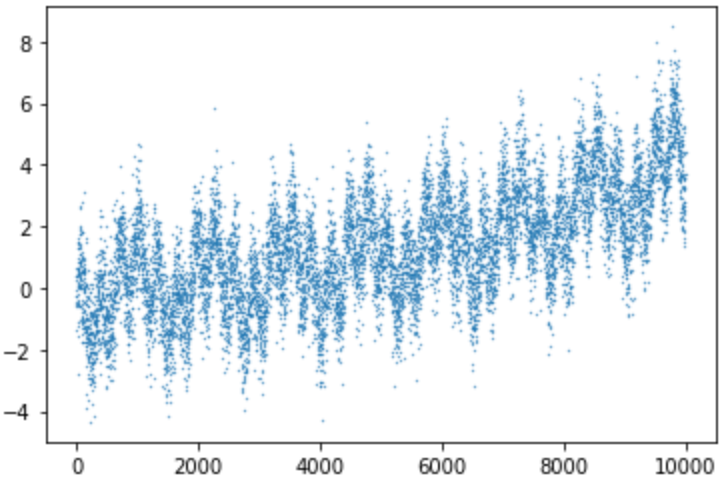
Задание

Необходимо сгенерировать синтетические данные и обучить на них модель авторегрессии, модель авторегрессии с методом главных компонент, модель LSTM.

Обучить рекуррентную нейронную сеть на реальных данных погоды по варианту.

Выполнение

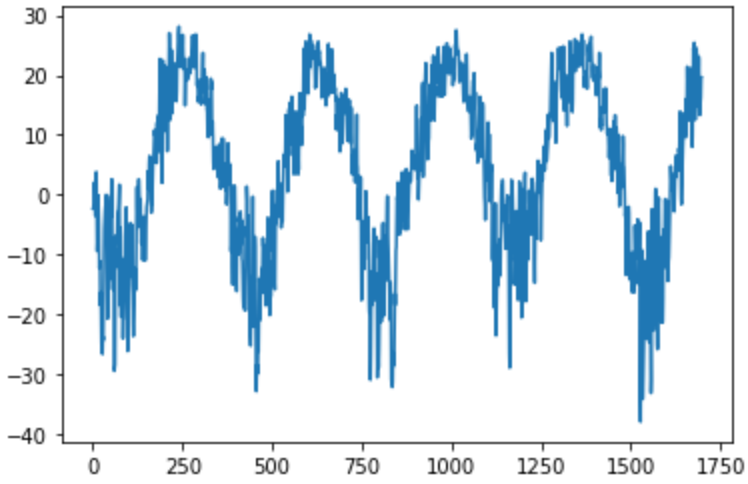
```
[ ]  X = np.arange(10000)
     y = np.sin(X/50)-np.sin(X/200)+(2*X/X.size)**2
     y += np.random.normal(scale=1.0, size=y.size)
     plt.scatter(X, y[:10000], s=0.1)

     df = pd.Series(y)
     df = df.diff().dropna()
```



Генерация синтетических данных для модели авторегрессии

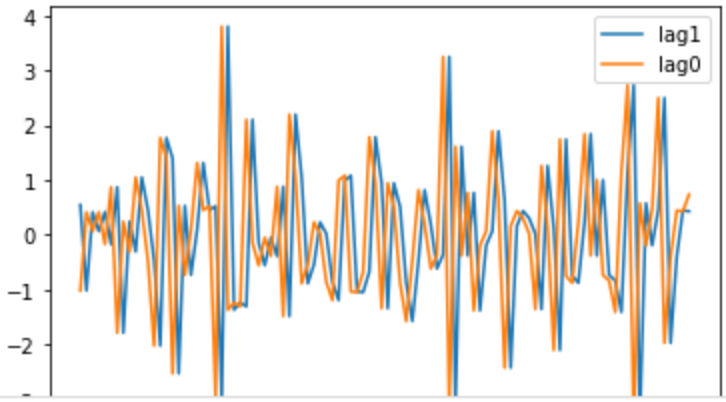
```
X_train = X[:X.shape[0]*9//10].copy()
y_train = y[:y.shape[0]*9//10].copy()
#y_train -= X_train[:, -1:]

X_test = X[X.shape[0]*9//10:].copy()
y_test = y[y.shape[0]*9//10:].copy()
#y_test -= X_test[:, -1:]

plt.plot(np.cumsum(y_test))
```

```
[<matplotlib.lines.Line2D at 0x7f4e8592c9d0>]
```




```
[ ]  plt.plot(X_test[-100:,-1], label='lag1')
     plt.plot(y_test[-100:], label='lag0')
     plt.legend()
```

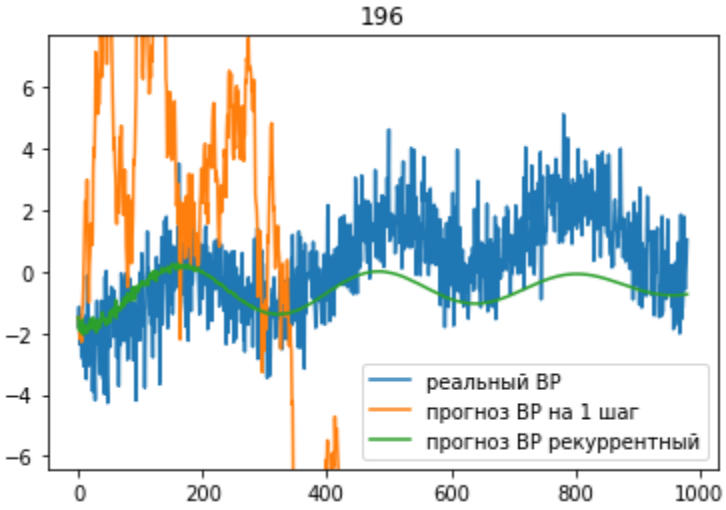
```
<matplotlib.legend.Legend at 0x7f905bee2450>
```



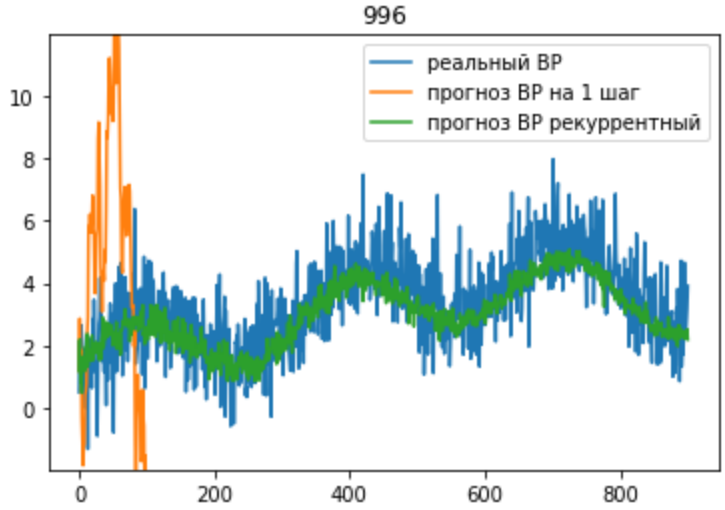
Формирование тренировочной и тестовой выборок

```
stats = []
series = []

windows = np.arange(1, 1001, 5)
for w in tqdm(windows):
    WINDOW_SIZE = w
    X = []
    df.rolling(WINDOW_SIZE+1).apply(getWindows)
    X = np.array(X)
    y = X[:, WINDOW_SIZE:].copy()
    X = X[:, :WINDOW_SIZE].copy()

    X_train = X[:X.shape[0]*9//10].copy()
    y_train = y[:y.shape[0]*9//10].copy()
    #y_train -= X_train[:, -1:]

    X_test = X[X.shape[0]*9//10:].copy()
    y_test = y[y.shape[0]*9//10:].copy()
    #y_test -= X_test[:, -1:]

    lr = LinearRegression().fit(X_train, y_train)

    #preds = lr.predict(X_test)
    #preds = recursive_predict(lr, X_test)
    tmp = []
    p = [lr.predict(X_test), recursive_predict(lr, X_test)]
    for preds in p:
        mae = mean_absolute_error(y_test, preds)
        r2 = r2_score(y_test, preds)
        tss = time_series_score(y_test, preds, X_test)
        tmp.extend([mae, r2*100, tss*100])
    stats.append(tmp)
    series.append(preds)
    if (len(stats)%10)==0:
        plt.plot(np.cumsum(y_test), label='реальный ВР')
        plt.plot(np.cumsum(p[0]), label='прогноз ВР на 1 шаг')
        plt.plot(np.cumsum(p[1]), label='прогноз ВР рекуррентный')
        plt.title(str(WINDOW_SIZE))
        plt.ylim(np.cumsum(y_test).min()*1.5, np.cumsum(y_test).max()*1.5)
        plt.legend(loc=0)
        plt.show()
```

Описание модели авторегрессии

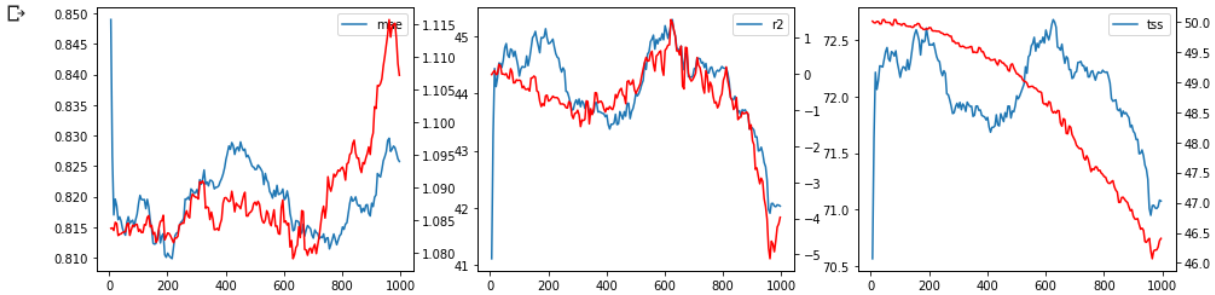


Обучение (начало)



Обучение завершено

```
fig, ax = plt.subplots(1, 3, figsize=(16, 4))

ax[0].plot(windows[1:], np.array(stats)[1:,0], label='mae')
ax[1].plot(windows[1:], np.array(stats)[1:,1], label='r2')
ax[2].plot(windows[1:], np.array(stats)[1:,2], label='tss')

ax[0].twinx().plot(windows[1:], np.array(stats)[1:,3], c='r', label='mae rec')
ax[1].twinx().plot(windows[1:], np.array(stats)[1:,4], c='r', label='r2 rec')
ax[2].twinx().plot(windows[1:], np.array(stats)[1:,5], c='r', label='tss rec')

for ax_ in ax:
    ax_.legend()
```

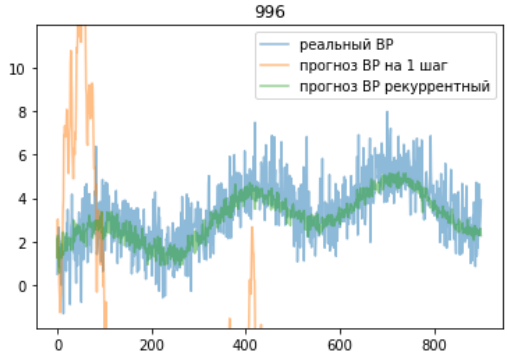


Результат

```python
def time_series_score_pca(y_true, y_pred, X_true, pca):
    true_values = y_true#pca.inverse_transform(X_true)[:, -1]
    pred_values = y_pred#pca.inverse_transform(X_true)[:, -1]

    last_value = pca.inverse_transform(X_true)[:,-1].copy()
    variance_dummy = ((last_value - true_values)**2).mean()
    variance_pred = ((pred_values - true_values)**2).mean()
    #print(variance_dummy, variance_pred)
    r2 = 1 - variance_pred/variance_dummy
    #r2_adj = 1 - (1-r2)*(y_true.size - 1)/(y_true.size - X_true.shape[1] - 1)
    return r2

def recursive_predict_pca(model, X, pca):
    preds = []
    length = X.shape[0]
    last_vals = pca.inverse_transform(X)[0].copy()
    for i in range(length):
        preds.append(model.predict(pca.transform([last_vals])))
        last_vals[:-1] = last_vals[1:]
        last_vals[-1:] = preds[-1]
    return np.array(preds).reshape(-1)
    # preds = []
    # length = X.shape[0]
    # last_vals = pca.inverse_transform(X)[0].copy()
    # for i in range(length):
    #     preds.append(model.predict(pca.transform([last_vals])))
    #     tmp = pca.transform([last_vals])[0, -1]
    #     last_vals[:-1] = last_vals[1:]
    #     last_vals[-1:] = tmp + preds[-1]
    # return np.array(preds).reshape(-1)
```

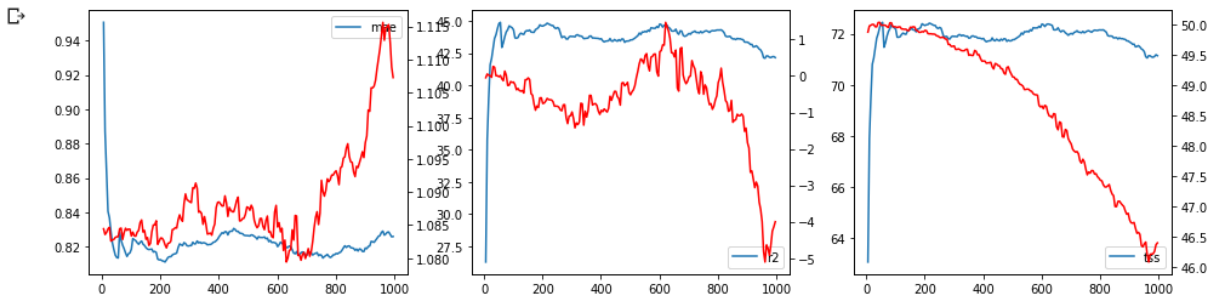
Определение модели авторегрессии с методом главных компонент

```
[0.8259581463363723, 42.15316395743631, 71.13621134963365, 1.107286271888006, -4.000831605756727, 46.4001654518037]
```



Обучение модели

```
fig, ax = plt.subplots(1, 3, figsize=(16, 4))

ax[0].plot(windows[1:], np.array(stats)[1:,0], label='mae')
ax[1].plot(windows[1:], np.array(stats)[1:,1], label='r2')
ax[2].plot(windows[1:], np.array(stats)[1:,2], label='tss')

ax[0].twinx().plot(windows[1:], np.array(stats)[1:,3], c='r', label='mae rec')
ax[1].twinx().plot(windows[1:], np.array(stats)[1:,4], c='r', label='r2 rec')
ax[2].twinx().plot(windows[1:], np.array(stats)[1:,5], c='r', label='tss rec')

for ax_ in ax:
    ax_.legend()
```



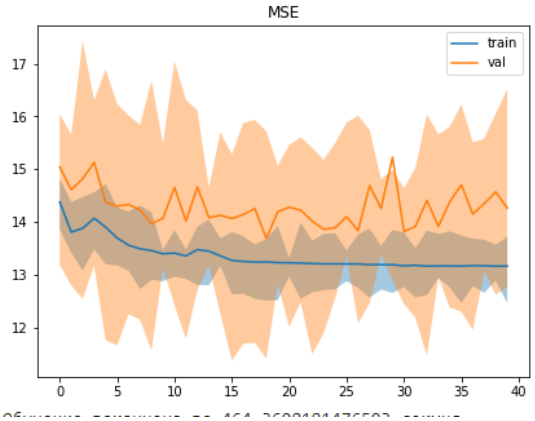
Результат обучения

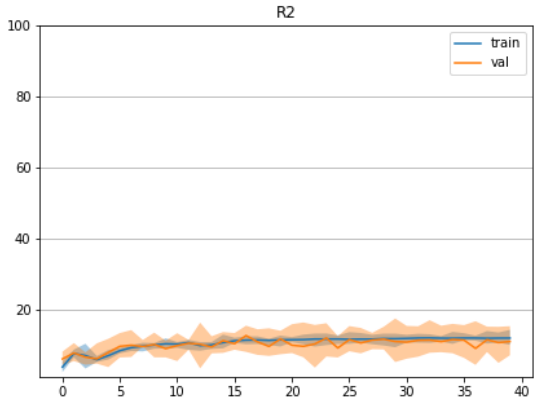
```
class LSTM(nn.Module):

    def __init__(self, num_classes, input_size, hidden_size, num_layers):
        super(LSTM, self).__init__()

        self.num_classes = num_classes
        self.num_layers = num_layers
        self.input_size = input_size
        self.hidden_size = hidden_size

        self.lstm = nn.LSTM(input_size=input_size, hidden_size=hidden_size,
                            num_layers=num_layers, batch_first=True)

        self.fc = nn.Linear(hidden_size, num_classes)

    def forward(self, x):
        h_0 = torch.autograd.Variable(torch.zeros(
            self.num_layers, x.size(0), self.hidden_size))

        c_0 = torch.autograd.Variable(torch.zeros(
            self.num_layers, x.size(0), self.hidden_size))

        # Propagate input through LSTM
        ula, (h_out, _) = self.lstm(x, (h_0, c_0))

        h_out = h_out.view(-1, self.hidden_size)

        out = self.fc(h_out)

        return out
```

Определение модели LSTM

```
print('Обучение закончено за %s секунд' % passed)
```

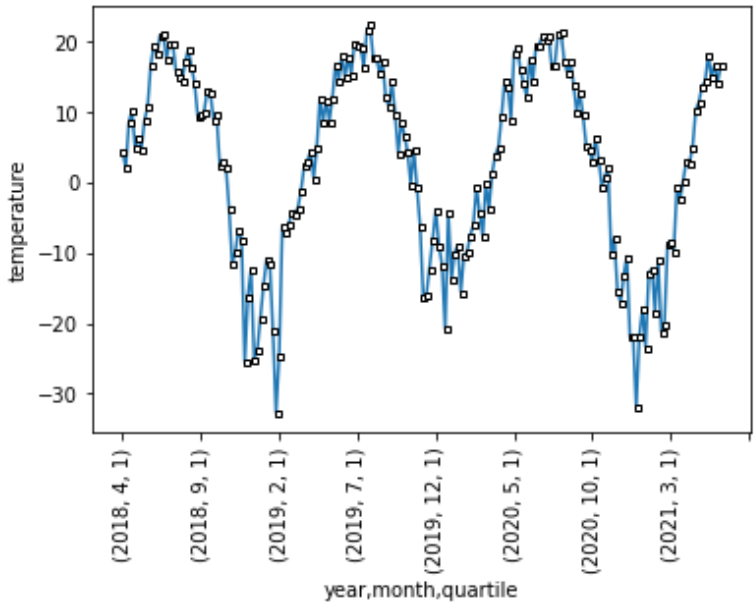
```
Эпоха: 40
Лучший коэффициент детерминации: 13.688967432294573
Текущий коэффициент детерминации: 14.262728486742292
100% 600/600 [00:15<00:00, 38.19it/s]
```





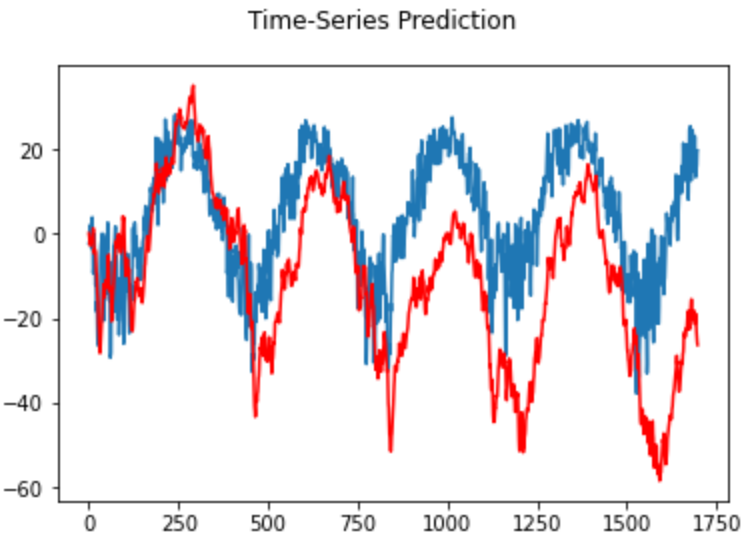
Обучение

```python
# По варианту проделать те же шаги для набора данных с погодой за полвека
df = pd.read_csv('climate/climate_novosib.csv', sep='\t')
df = df[df.year>1972].copy()
df['quartile'] = df.day//7
df.groupby(['year', 'month', 'quartile']).\
   Temperature.\
   aggregate('mean').\
   iloc[-4*12*4:].\
   plot(rot=90, ylabel='temperature',
        marker='s',
        markersize=3,
        markerfacecolor='white',
        markeredgecolor='k')
df = df.Temperature.diff().dropna()
```



Данные по погоде в Новосибирске

```
model.eval()
train_predict = model(tensor_X_test.view(-1, WINDOW_SIZE, 1))

data_predict = train_predict.data.numpy()#*MAX_VAL
dataY_plot = tensor_y_test.data.numpy()#*MAX_VAL

plt.plot(np.cumsum(dataY_plot))
plt.plot(np.cumsum(-data_predict), color='r')
plt.suptitle('Time-Series Prediction')
plt.show()
```



Погода, предсказанная моделью и реальная погода