Лабораторная работа № 1

Суррогатное моделирование

Содержание

[1]:
```python
# Imports
import numpy as np
import matplotlib.pyplot as plt
```

[2]:
```python
# Styles
import matplotlib
matplotlib.rcParams['font.size'] = 14
cm = plt.cm.tab10  # Colormap

import seaborn
from IPython.display import Image
im_width = 800
```

[4]:
```python
def set_constants(obj_fun):
    '''Set bounds and optimum point'''

    fun_name = obj_fun.__name__
    if fun_name == 'ParSin':
        X_LIM = [0., 1.]
        F_LIM = [0, obj_fun(X_LIM[1])]
        X_OPT = 0.757

    elif fun_name == 'rosen':
        X_LIM = [-1.2, 1.2]
        F_LIM = [0, 680]
        X_OPT = [1., 1.]

    else:
        raise ValueError(f"Unknown objective function: '{fun_name}'")

    return np.array(X_LIM), np.array(F_LIM), np.array(X_OPT)
```

---

1. Одномерный случай

Парабола × синус (ParSin)

$$f(x) = (6x-2)^2 \cdot \sin(12x-4)$$

Глобальный минимум: $x = 0.757$, $f(x) = -6.021$
Локальный минимум: $x = 0.143$, $f(x) = -0.986$
Точка перегиба: $x = 0.333$, $f(x) = 0.0$

[5]:
```python
def ParSin(x):
    '''Parabola times sine'''
    return (6*x-2)**2 * np.sin(12*x-4)
ParSin.__name__ = 'ParSin'
```

[6]:
```
def graph_fun(fun, trajectory=[], figname='', noisy=False):
    '''Plot function'''
    seaborn.set_style('whitegrid')

    plt.figure(figsize=(8, 5))
    X_test = np.linspace(*X_LIM, 401)

    # function contours
    if noisy:
        plt.plot(X_test, fun(X_test), 'kx', alpha=.5, label='Objective
 ↪function')
    else:
        plt.plot(X_test, fun(X_test), 'k-', label='Objective function')

    # points
    plt.plot(X_OPT, fun(X_OPT), '*', ms=20, c=cm(3), label='Minimum')
    if (len(trajectory) != 0):
        X = trajectory
        plt.plot(X[0],  fun(X[0]),   'o', c=cm(0), ms=8)
        plt.plot(X,     fun(X),     '-o', c=cm(0), ms=3.5)
        plt.plot(X[-1], fun(X[-1]),  '+', c=cm(0), mew=2., ms=15)

    plt.legend()
    plt.tight_layout()
    if (figname):
        plt.savefig(figname, dpi=200, bbox_inches='tight')
```

1.1. Данные без шума

Выбор задачи и установка констант

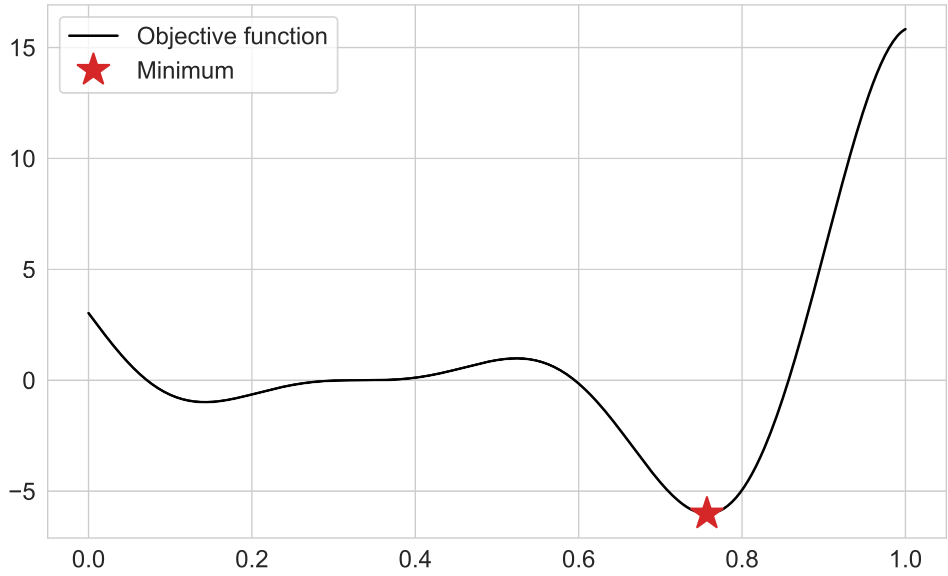
[7]:
```
obj_fun = ParSin
X_LIM, F_LIM, X_OPT = set_constants(obj_fun)

print(f'obj_fun = {obj_fun.__name__}')
print(f'X_OPT = {X_OPT}, obj_fun(X_OPT) = {obj_fun(X_OPT):.3f}')
```

```
obj_fun = ParSin
X_OPT = 0.757, obj_fun(X_OPT) = -6.021
```

Отрисовка графика выбранной целевой функции

[8]:
```
graph_fun(obj_fun)
```



1.2. Данные с шумом

Теперь добавим к целевой функции шум:

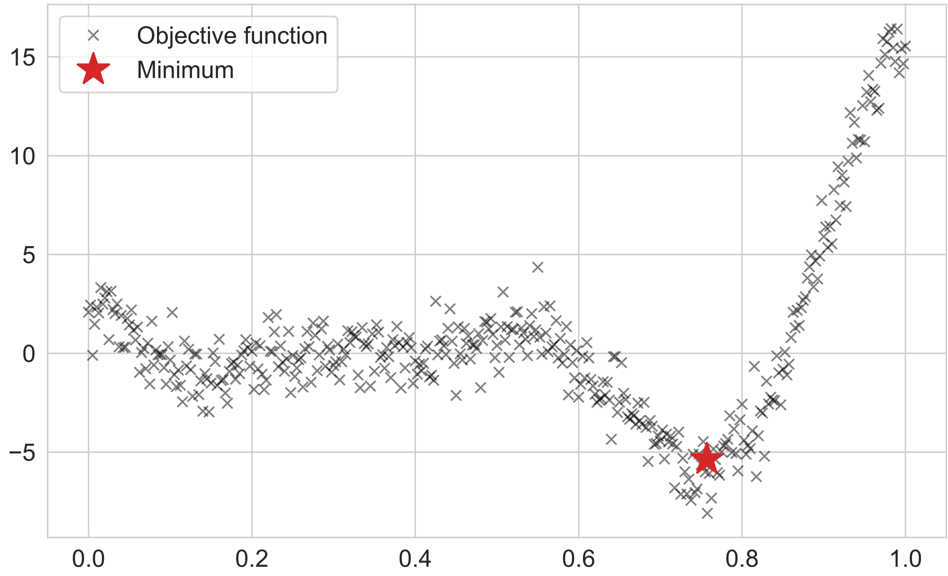
$$f_{noisy} = f(x) + \sigma_n \xi.$$

Здесь ξ — нормальная случайная величина, переменная σ_n задаёт амплитуду шума.

[9]:
```
def add_noise(fun, sigma_n):
    def ret_fun(x):
        xi = np.random.randn(*x.shape)
        return fun(x) + sigma_n * xi
    return ret_fun
```

[10]:
```
sigma_n = 1.0
obj_fun_noisy = add_noise(obj_fun, sigma_n)
```

[11]:
```
graph_fun(obj_fun_noisy, noisy=True)
```



2. Двумерный случай

Функция Розенброка

$$f(x_1, x_2) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$$

Функция имеет единственный минимум, находящийся внутри узкой параболической долины в точке $x = (1, 1)$ и равный 0.

[12]:
```
def rosen(x):
    '''Rosenbrock function'''
    # 2D: f = 100*(x2 - x1**2)**2 + (1 - x1)**2
    return sum(100.0*(x[1:]-x[:-1]**2.0)**2.0 + (1-x[:-1])**2.0)
rosen.__name__ = 'rosen'
```

[13]:
```
# functions for visualization
def fun_2d(X1, X2, fun):
    array_2d = np.zeros((len(X1), len(X2)))
    for i, x2 in enumerate(X2):
        for j, x1 in enumerate(X1):
            array_2d[i, j] = fun(np.array([x1, x2]))
    return array_2d
```

[14]:
```
def fun_contours(fun, points=[], constr=None, trajectory=[], figname=''):
    '''Draw function 2D contours'''
    seaborn.set_style('white')

    plt.figure(figsize=(7, 7))
    X1 = X2 = np.linspace(*X_LIM, 401)

    # function contours
    z_lines = np.linspace(0, F_LIM[1]**0.5, 20)**2
    if (F_LIM[0] < 0):
        z_lines_1 = np.linspace(F_LIM[0], 0, 20)
        z_lines = np.concatenate((z_lines_1[:-1], z_lines))

    contours = plt.contour(X1, X2, fun_2d(X1,X2,fun), z_lines,
                           linewidths=1., colors='k', alpha=0.9)
    plt.clabel(contours, fontsize=8, fmt='%.0f')

    # points
    for point in points:
        plt.plot(*point, 'x', c=cm(3), mew=2., ms=15)

    # trajectory
    if (len(trajectory) != 0):
        plt.plot(*trajectory[:,0],   'o', c=cm(0), ms=8)
        plt.plot(*trajectory,       '-o', c=cm(0), ms=3.5)
        plt.plot(*trajectory[:,-1],  '+', c=cm(0), mew=2., ms=15)

    # constraint
```

```
    if constr:
        plt.contour(X1,X2,fun_2d(X1,X2,constr),0,linewidths=1.,colors=cm(1))

    plt.xlabel(r"$x_1$")
    plt.ylabel(r"$x_2$", rotation='horizontal', horizontalalignment='center')
    plt.xlim(*X_LIM)
    plt.ylim(*X_LIM)
    plt.tight_layout()
    plt.show()
    if (figname):
        plt.savefig(figname, dpi=200, bbox_inches='tight')
```

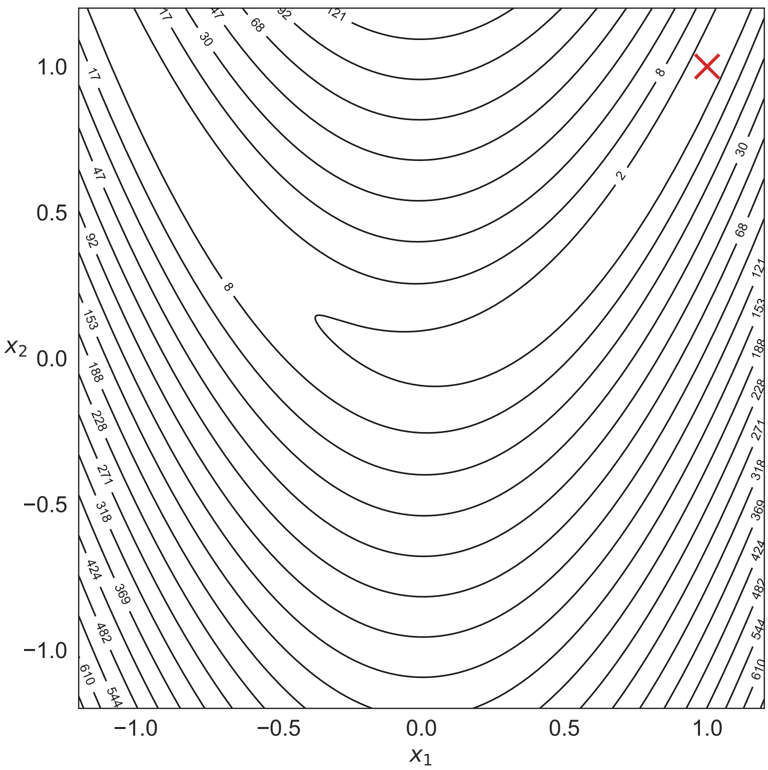
2.1. Данные без шума

[15]:
```
obj_fun = rosen
X_LIM, F_LIM, X_OPT = set_constants(obj_fun)
F_OPT = obj_fun(X_OPT)

print(f'obj_fun = {obj_fun.__name__}')
print(f'X_OPT = {X_OPT}, F_OPT = {F_OPT:.3f}')
```

```
obj_fun = rosen
X_OPT = [1. 1.], F_OPT = 0.000
```

[16]:
```
fun_contours(obj_fun, points=[X_OPT])
```

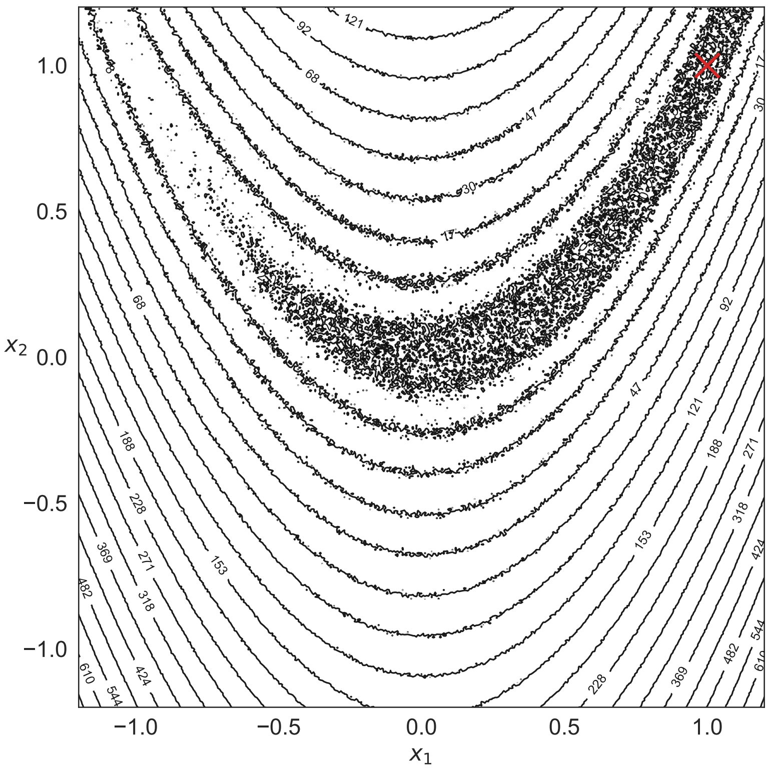


2.2. Данные с шумом

[17]:
```
def add_noise(fun, sigma_n):
    def ret_fun(x):
        xi = np.random.randn()
        return fun(x) + sigma_n * xi
    return ret_fun
```

[18]:
```
sigma_n = 1.0
obj_fun_noisy = add_noise(obj_fun, sigma_n)
```

[19]:
```
fun_contours(obj_fun_noisy, points=[X_OPT])
```



3. Задание

Провести сравнение двух методов построения суррогатных моделей: полиномиальная регрессия (PR) и регрессия на основе гауссовских процессов (GPR). Рассмотреть два случая: одномерный (функция ParSin) и двумерный (функция Розенброка).

Задачи:

1. Сделать план экспериментов, рекомендуется использовать метод LHS из библиотеки `pyDOE2` или `DOEpy`.
2. Построить суррогатные модели методами: PR и GPR. Количество признаков для полиномиальной регрессии выбрать самостоятельно.
3. Построить зависимость ошибки по норме L_2 от количества точек в обучающей выборке N_{train} (1D: $N_{train,1D} = [4, 8, 16]$, 2D: $N_{train,2D} = [8, 16, 32]$). Ошибку считать по методу кросс-валидации.
4. Повторить процедуру для данных с шумом $\sigma_n = 1.0$.