Лабораторная работа № 1

Оптимизация с помощью `scipy.optimize` Одномерные функции

Содержание

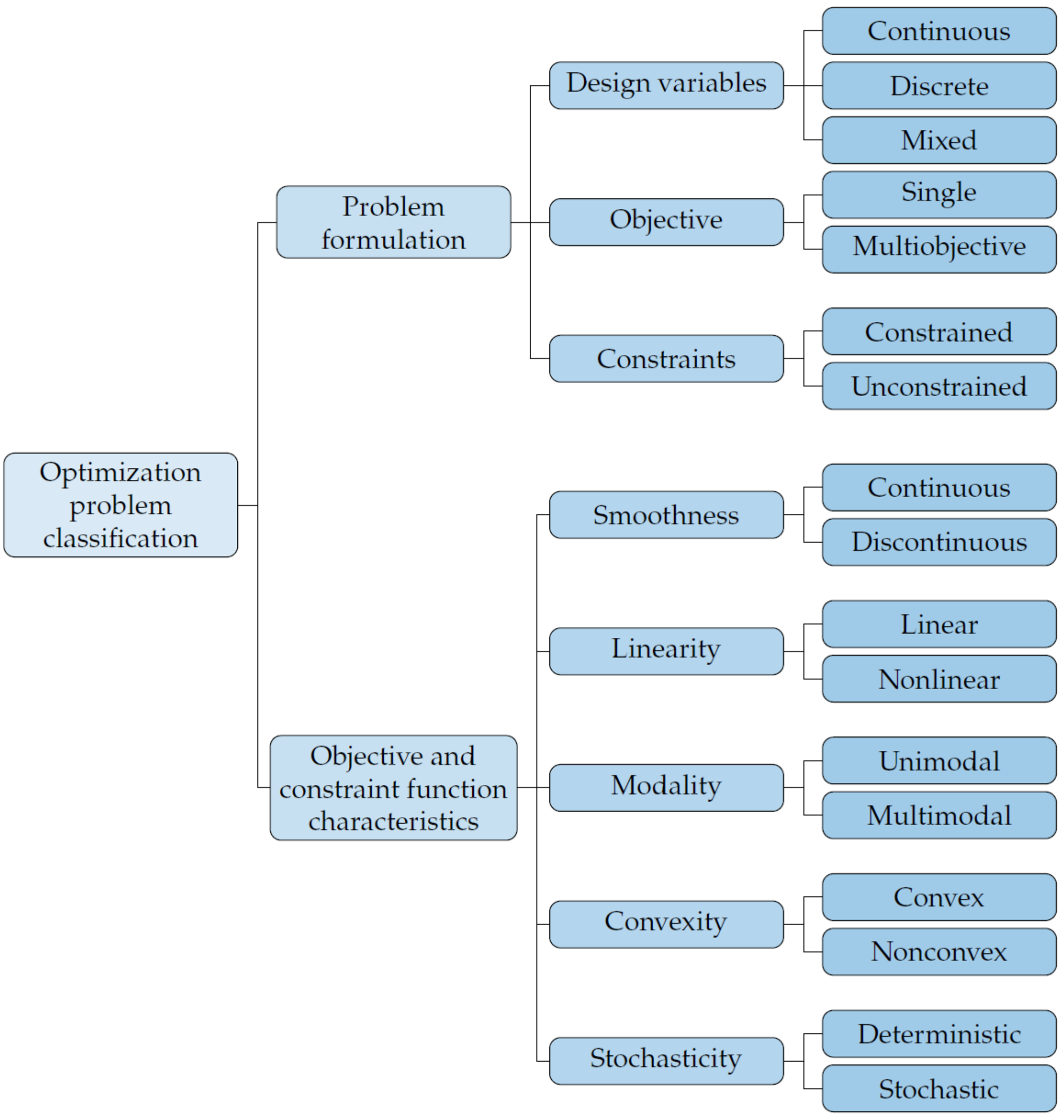
[1]:
```
# Imports
import numpy as np
import matplotlib.pyplot as plt
import scipy as sp
from scipy.optimize import minimize
```

[2]:
```
# Styles
import matplotlib
matplotlib.rcParams['font.size'] = 12
cm = plt.cm.tab10  # Colormap
import seaborn
seaborn.set_style('whitegrid')
```

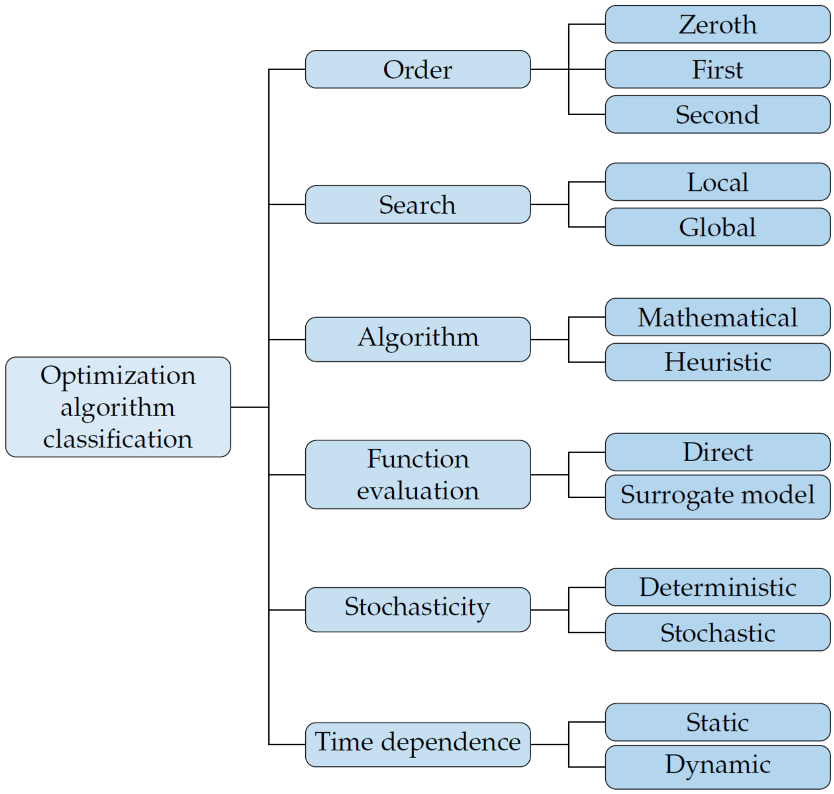
---

1. Методы оптимизации

1.1. Классификация



Классификация оптимизационных задач



Классификация оптимизационных алгоритмов

Источник: *Martins J.R.R.A. & Ning A.* Engineering Design Optimization. — 2021. — 637 с.

1.2. Библиотека `scipy.optimize`

Для решения задач условной и безусловной оптимизации пакет `scipy.optimize` предлагает набор алгоритмов, включающий в том числе следующие:

- Метод сопряжённых градиентов (CG)
- Алгоритм Бройдена — Флетчера — Гольдфарба — Шанно (BFGS)
- Последовательное квадратичное программирования (SLSQP)
- Симплекс-метод Нелдера — Мида
- Алгоритм COBYLA (Constrained Optimization By Linear Approximation)

Подробнее об оптимизации с помощью `scipy.optimize` можно прочитать тут.

---

2. Целевая функция

Подключаем библиотеки, создаём вспомогательные функции.

```
[6]: from copy import deepcopy
def counted(f):
    def wrapped(*args, **kwargs):
        wrapped.calls += 1
        wrapped.Xk.append(deepcopy(*args))
        return f(*args, **kwargs)
    wrapped.calls = 0
```

```
    wrapped.Xk = []
    return wrapped

# auxiliary function to save intermediate points
def store(xk):
    Xk.append(xk)
```

2.1. Квадратичная функция

Начнём с самого простого:

$$f(x) = x^2$$

[7]:
```
@counted
def QF(x):
    '''Quadratic function'''
    return x*x
QF.__name__ = 'QF'
```

2.2. 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: $x = 0.333$, $f(x) = 0.0$

[8]:
```
@counted
def ParSin(x):
    '''Parabola times sine'''
    return (6*x-2)**2 * np.sin(12*x-4)
ParSin.__name__ = 'ParSin'
```

[9]:
```
def set_constants(obj_fun):
    '''Set bounds and optimum point'''

    if obj_fun == QF:
        X_LIM = [-2., 2.]
        F_LIM = [0, obj_fun(X_LIM[1])]
        X_OPT = 0.
    elif obj_fun == ParSin:
        X_LIM = [0., 1.]
        F_LIM = [0, obj_fun(X_LIM[1])]
        X_OPT = 0.757

    X_LIM = np.array(X_LIM)
    F_LIM = np.array(F_LIM)
    X_OPT = np.array(X_OPT)

    return X_LIM, F_LIM, X_OPT
```

3. Постановка задачи

3.1. Данные без шума

Выбор задачи и установка констант

[10]:
```
obj_funs = [QF, ParSin] # choose a function

obj_fun = obj_funs[1]
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 графиков выбранной целевой функции

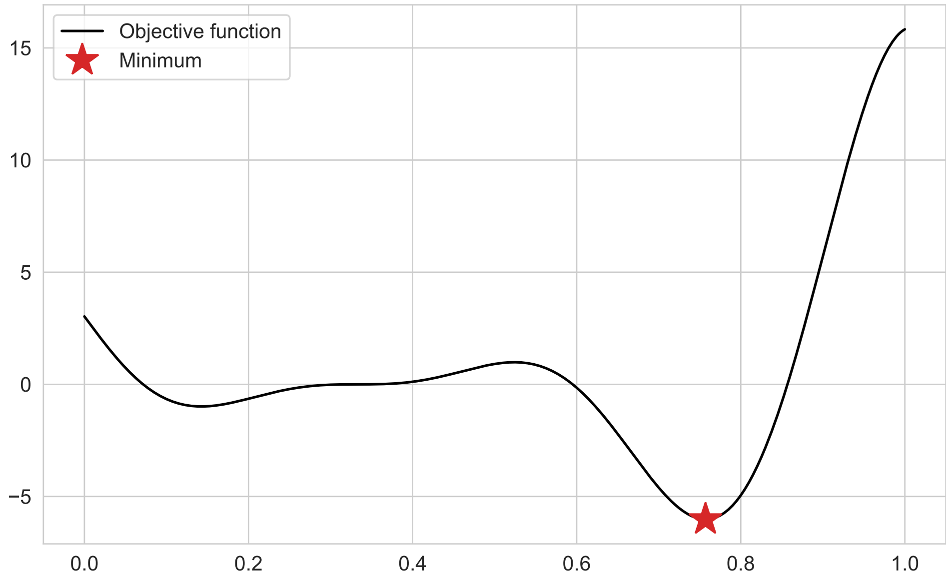
[11]:
```
def graph_fun(fun, trajectory=[], figname='', noisy=False):
    '''Plot function'''
    plt.figure(figsize=(8, 5))
    X_test = np.linspace(*X_LIM, 401)

    # 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')
```

[12]:
```
graph_fun(obj_fun)
```



3.2. Данные с шумом

Теперь добавим к целевой функции шум:

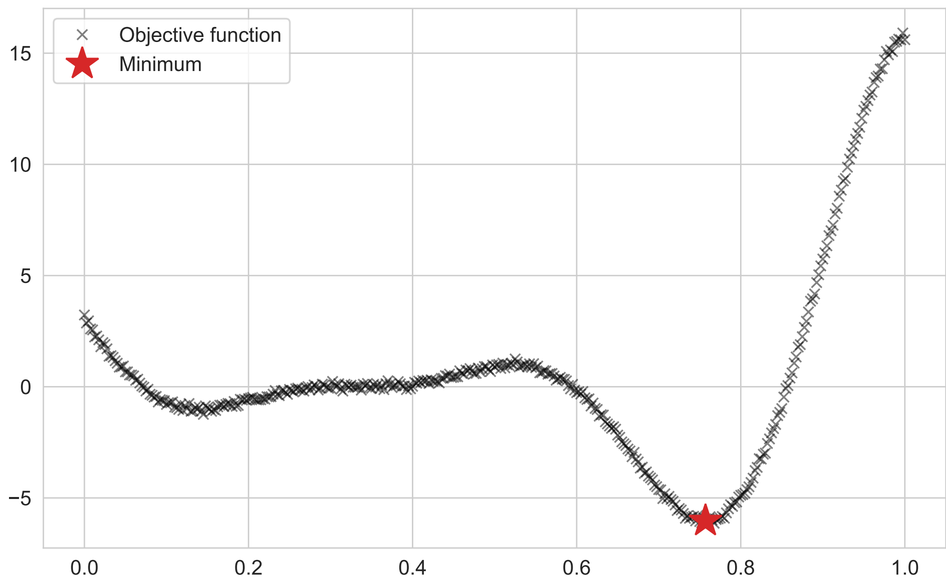
$$f_{noisy} = f(x) + \sigma_n \xi.$$

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

[13]:
```
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
```

[14]:
```
sigma_n = 1e-1
obj_fun_noisy = add_noise(obj_fun, sigma_n)
```

[15]:
```
graph_fun(obj_fun_noisy, noisy=True)
```



4. Задание

Необходимо провести сравнительное тестирование двух предложенных алгоритмов (можно больше) и оформить отчёт в виде файла `.ipynb`.

Отчёт должен содержать:

1. Теоретическая часть: краткое описание алгоритмов, как их можно классифицировать.
2. Сравнение эффективности работы алгоритмов. Эффективность работы оценивается по количеству вызовов целевой функции при заданной точности. Для сравнения необходимо использовать две целевые функции: парабола и ParSin. Приветствуется визуализация работы алгоритма (пример будет ниже).
3. Анализ результатов. Заключение.
4. (Опционально) Исследовать влияние амплитуды шума на точность работы алгоритмов и количество вызовов целевой функции.

Замечания:

1. Список параметров алгоритма можно получить так:
`sp.optimize.show_options(solver='minimize', method='Nelder-Mead')`
2. Определённая выше функция `graph_fun()` может рисовать траекторию поиска и сохранять рисунок (см. параметры функции)