Exercises 07: Optimization

Kevin Heubacher (15329079), Akshay Chikhalkar (15489036),
John Paul Semakula (15487087)
Group: LATE

29.11.2022

1 Exercise 1

The scikit-optimize python extension implements bayesian optimization using gaussian processes with *gp_minimize*(). Some of the used parameters can be seen in table 2.

Parameter	**Description**	**Value**
func	Function to minimize	polynomial
dimensions	The search space	[(x_min, x_max)]
n_calls	Number of calls to func	4
acq_func	Aquisition function	"EI"
random_state	Set for reproducible results	1234
noise	Adds noise to the objective function (minimum: 1e-10)	1e-10

Table 1: Parameters used in gp_minimize() function

The number of loops done is determined by the *n_calls* parameter. To compare this implementation with the implementation of exercise 6 both optimization algorithms are run 4 times with $x\_min = 1$ and $x\_max = 20$. The comparison can be seen in figure 1. Noticeable is the switch to a discrete X dimension. The MSE value is hold until the next integer is reached. This is done automatically by the *gp_minimize* function when the dimensions parameters are integers. The two implementations also chose completely different Observations witch each iteration. The best degree found was 11 with $MSE = 0.0062$ for *gp_minimize*() and 10 with $MSE = 0.0128$ in exercise 6. However, given the randomness of the *gp_minimize*() function, the result will likely change with a different *random_state* value.

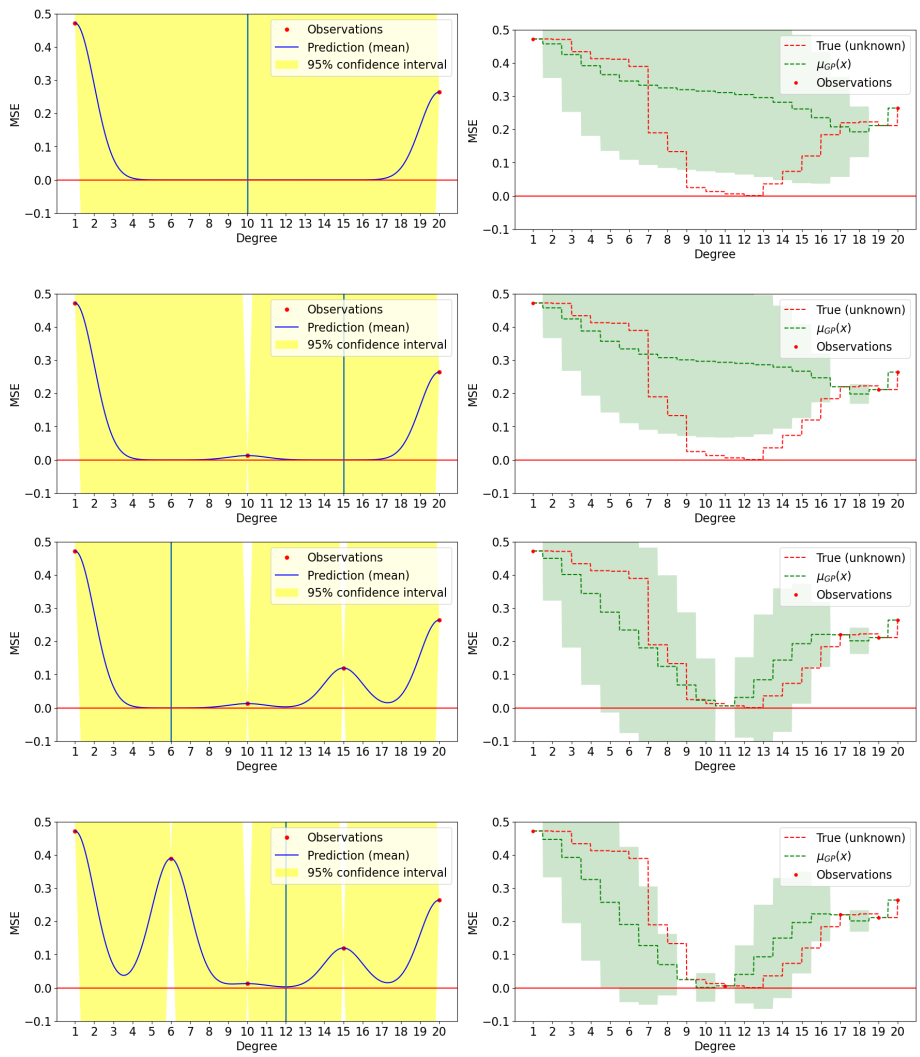

Figure 1: Left: Implementation Ex. 6, Right: Implementation Ex. 7

2 Exercise 2

The aim of this exercise is to benchmark Knapsack Problem with different items with their values and objects for multiple iterations.

The knapsack problem is a problem in combinatorial optimization: Given a set of items, each with a weight and a value, determine the number of each item included in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible. This exercise is using Pyomo, a Python-based open-source software package that supports a diverse set of optimization capabilities for formulating, solving, and analyzing optimization models. A core capability of Pyomo is modelling structured optimization applications.

The most common problem being solved is the $0-1$ knapsack problem, which restricts the number x_i of copies of each kind of item to zero or one.
Given a set of k items numbered from 1 up to k, each with a weight w_i and a value v_i, along with a maximum weight capacity $limit$,

$$\text{maximize} \sum_{i=1}^{k} v_i x_i \quad \text{subject to} \sum_{i=1}^{k} w_i x_i \leq limit \quad \text{where} \quad x_i \in \{0,1\}$$

$v-$Item value, $w-$ Item weight, $k-$No. of items, $limit-$Weight limit
x_i- the number of instances of item i to include in the knapsack.

Steps:

1. Items generator: This function generates a given number of items with a random name of string less than 10 as well as a random value between 1 to 20 and weight between 1 to 10.
2. Recording execution time for n (i.e. $n = 5$) number of iterations.
3. Calculating the mean of a recorded set of time (i.e. Set of n).
4. Calculating variance of a recorded set of time (i.e. Set of n).
5. Visual representation of time vs iteration.

The following table represents the benchmarking results:

Number of items = 10
Number of iterations $n = 5$

Iteration (n)	**Execution Time (sec)**	**Value**	**Weight**
1	0.054	37	10
2	0.038	37	10
3	0.043	37	10
4	0.030	37	10
5	0.030	37	10

Table 2: Benchmarking results for n iterations

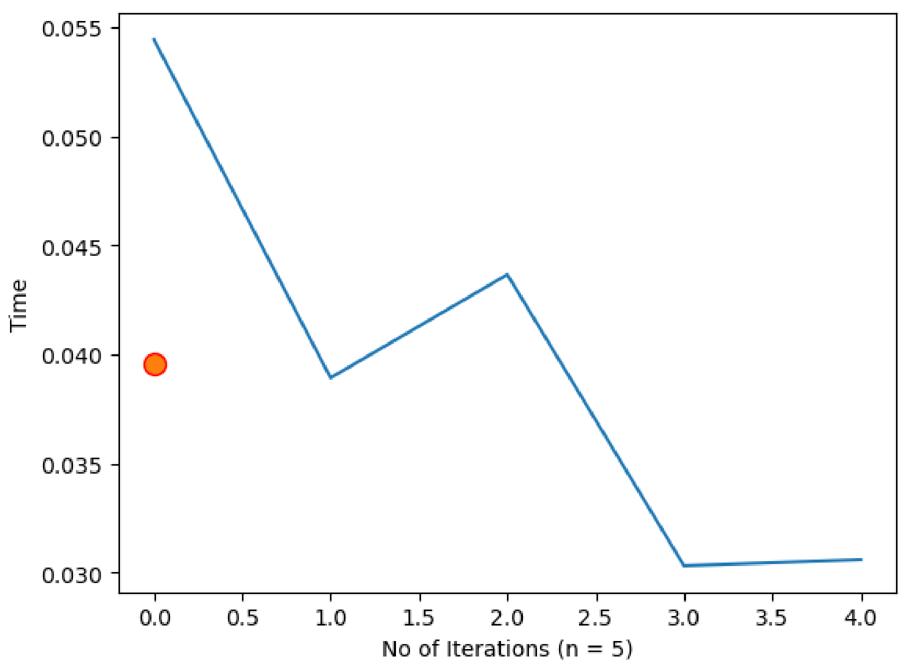

Figure 2: Execution time plot for n iterations

The mean execution time for above n iterations is **0.04 sec** (0.039 sec).

The variance of execution time for above n iterations is **8.07 Sec**.