Report Lab 6

Nguyen Tien Duc - ITITIU18029

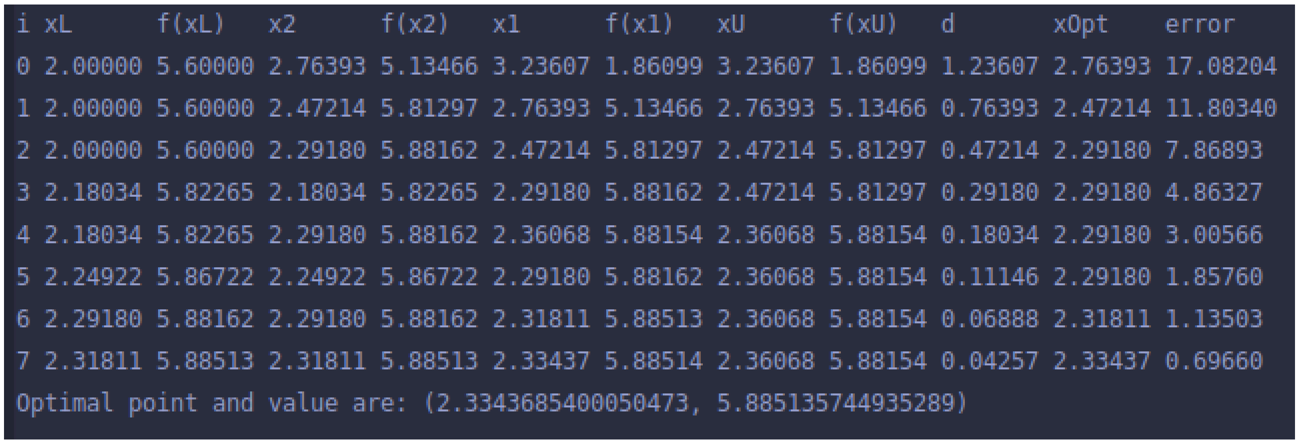
December 17, 2019

1/ Golden-Section Search

Code

```
import math as m


def function(x):
  return 4 * x - 1.8 * x * x + 1.2 * x ** 3 - 0.3 * x ** 4


def golden_section_search(f, xL, xU, e_s, max_iter):
  print(
    "%-2s%-8s%-8s%-8s%-8s%-8s%-8s%-8s%-8s%-8s%-8s%-9s" % ("i", "xL", "f(xL)", "x2", "
                                                          f(x2)", "x1", "f(x1)", "xU", "f(xU)", "d"
                                                          , "xOpt", "error"))
  R = (m.sqrt(5) - 1) / 2
  xOpt = 0
  for iter in range(0, max_iter):
    d = R * (xU - xL)
    x1 = xL + d
    x2 = xU - d

    if f(x2) > f(x1):
      xOpt = x2
      xU = x1
    if f(x2) < f(x1):
      xOpt = x1
      xL = x2

    err = (1 - R) * abs((xU - xL) / xOpt) * 100

    print("%-2d%-8.5f%-8.5f%-8.5f%-8.5f%-8.5f%-8.5f%-8.5f%-8.5f%-8.5f%-8.5f%-9.5f" %
                                                          (
      iter, xL, f(xL), x2, f(x2), x1, f(x1), xU, f(xU), d, xOpt, err))

    if err < 1:
      return xOpt, f(xOpt)


print("Optimal point and value are: " + str(golden_section_search(function, 2, 4, 1,
                                                          1000)))
```

Running



```
i xL      f(xL)   x2      f(x2)   x1      f(x1)   xU      f(xU)   d       xOpt    error
0 2.00000 5.60000 2.76393 5.13466 3.23607 1.86099 3.23607 1.86099 1.23607 2.76393 17.08204
1 2.00000 5.60000 2.47214 5.81297 2.76393 5.13466 2.76393 5.13466 0.76393 2.47214 11.80340
2 2.00000 5.60000 2.29180 5.88162 2.47214 5.81297 2.47214 5.81297 0.47214 2.29180 7.86893
3 2.18034 5.82265 2.18034 5.82265 2.29180 5.88162 2.47214 5.81297 0.29180 2.29180 4.86327
4 2.18034 5.82265 2.29180 5.88162 2.36068 5.88154 2.36068 5.88154 0.18034 2.29180 3.00566
5 2.24922 5.86722 2.24922 5.86722 2.29180 5.88162 2.36068 5.88154 0.11146 2.29180 1.85760
6 2.29180 5.88162 2.29180 5.88162 2.31811 5.88513 2.36068 5.88154 0.06888 2.31811 1.13503
7 2.31811 5.88513 2.31811 5.88513 2.33437 5.88514 2.36068 5.88154 0.04257 2.33437 0.69660
Optimal point and value are: (2.3343685400050473, 5.885135744935289)
```

Analysis and Comparison

Firstly, this test and comparison between grid search, random search and bayesian Optimization is nowhere perfect. Obvious flops are:

- Only one function is used for all tests.
- Test object(the function) is too simple to demonstrate the methods.
- Bayesian implementation is borrowed from external source. Thus, it is not thoroughly understood.

Random Search vs Grid Search

The result from 3 different numbers of maximum steps clearly show that random search is more efficient as an algorithm.
In all 3 cases, random search always takes longer to process all the steps but, however, always takes less steps to reach the optimal point than grid search.
⇒ The long processing time of random search is substituted by a much faster approach to the solution.

	Random Search	Grid Search
Max Time	Longer, depends on Random algorithm	Faster as linspace algorithm is has linear time
Solve Time	Faster, independent of max steps	Longer, depends on max steps to form the grids
Accuracy	Unpredictable, but performs better in 3 cases	Predictable, but performs worse in 3 cases

AutoML(Bayesian) vs Non-AutoML(Random/Grid)

From the Bayesian data, it takes 15 iterations to reach the solution, not an impressive performance.
However, this result depends on the implementation of Bayesian Optimization from bayes-opt package.
In theory, Bayesian, as an auto-machine-learning method, proved to be really efficient as new choices are sampled from the data based on the history probability, thus able to avoid bad patterns and reach the optimization solution faster.

	AutoML(Bayesian)	Non-AutoML(Random/Grid)
Max Time	Longer, depends on bayes-opt algorithm	Faster as algorithm is customized for this test
Solve Time	Good but not impressive, same as Random	Grid search is still the longest
Accuracy	Bad guesses are corrected immediately	Either unpredictable or ill-performed

Conclusion

- Between the naive, non-machine-learning, methods, random search is the obvious best. Grid search is too tedious, as every models is tested for optimization, this will be a burden if the object function is too costly to compute.
- However, even with the better random search method, as the function becomes harder to compute and the variables bounds get complicated (e.g. becomes piece-wise), it will be really costly to regenerate random samples, and, costly to evaluate random, meaningless samples.
- As a result, between the best candidate of naive methods and auto-machine-learning methods, i.e. random search vs bayesian optimization, AutoML still proved to be a bester choice because only methods from AutoML that can learn from the past statistics is capable of optimizing real world problems, which is usually complicated, messy, non-linear, multidimensional, ...
- Finally, we can rank the 3 methods above as: Bayesian Optimization > Random search > Grid search