# StoSOO: yet an other implementation.

Philippe Preux
philippe.preux@univ-lille3.fr

December 5, 2013

This note presents my implementation of the algorithm named StoSOO as described by Valko *et al.* in [1].

In principle, this code can deal with any real function of any dimension though it is probably not appropriate to expect anything great wtih dimension more than 10.

## 1 Installation

Please note that the installation have been tested only on my laptop. I am pretty sure it will work on standard Linux computers, using `gcc` to compile it. If you encounter any trouble on a Linux system, please email me. If you are using a commercial system, I recommend you shift to a serious, open source, free system such as Linux/Ubuntu.

```
$ tar zxf stosoo.v.0.1.tgz
$ cd stosoo.v.0.1
$ make
```

produces 3 executable: `stosoo_2sine`, `stosoo_garland`, and `stosoo_para` which names are pretty straitghforward (para is a paraboloid to test ¿ 1 dimensions).

## 2 Usage

For example:

```
./stosoo_garland -N 1000 --dim 1 --ns 0.1
```

produces something like:

```
0.471092 1.10072 0.102949
# 595 nodes
```

which reads as the location of the maximum is $x^* = 0.471092$, the estimated value of $f(x^*)$ is 1.10072 and the regret is 0.102949.

`-N 1000` sets the number of evaluations of the objective function to 1000.

`--ns 0.1` indicates noise std dev is 0.1.

In more than 1 dim, you may test:

```
stosoo_para -N 1000 --dim 7 --ns .01 --split random
```

which produces something like:

```
# optimum set at (1, 0.5, 0.333333, 0.25, 0.2, 0.166667, 0.142857)
0.944444 0.611111 0.5 0.5 0.166667 0.166667 0.166667 -0.106381
# 592 nodes
```

where # lines are meant to be comments: the location of the maximum is set at random and it is printed in the first line.

The second is the location of the maximum that has been found and the value of the function at this point.

This considers a paraboloid in dimension 7 (`--dim 7`).

With more than 1 dimension comes the issue of the direction of splits. The –split option is meant to precise that: without it, split will occur successively is dimensions 0, 1, 2, ... 6, 0, ... You can specify either increasing, decreasing or random. increasing is the default; decreasing goes the other way around: 0,6,5,4,3,2,1,0,6 ..., and random selects the split direction at random.

## 3 More on usage

### 3.1 Seeding

Seeding the pseudo-random number generator: the execution of the code on a given objective function, in the same setting, gives exactly the same result each time. This is pretty nice for debugging, but a defect for real use of the code. To seed the generator, one uses the `--seed xxx` option, where `xxx` is a seed number (a positive integer then). If not provided, 1 is used as a default for `xxx`.

### 3.2 More options

A couple of parameters of the algorithm are set by default to their optimal value, optimal is the sense explained in [1]. You can always override these default values as follows:

- `-K x` sets the split factor to `s` (a positive integer). Default value is 3.
- `-k x` sets the maximum number of evaluation per node/cell to `x`.
- `-D x` sets the maximum depth of the tree to `x` ($h_{max}$ in [1]).
- `-d x` sets the value of $\delta$

## 4 Optimizing your function

In this section, you learn how to modify the provided source code to be able to optimize the objective function you wish to. Before that, please pay attention to this:

**important note** : this code has been developed keeping the efficiency as top priority. Henceforth, some parts of the code are rather sensitive and should be used as provided; clearly, the code may become severely bugged if you do not comply with the recommendations provided below.

It is very easy to implement the objective function you wish to optimize as long as:

- you wish to **maximize** it,
- the function is defined over a compact domain having the shape of a hyper-rectangle.

If your function is actually a toy function you know the location of the maximum and merely want use it to test StoSOO, you may take advantage of this information to assess the algorithm. See section "Playing with a toy function".

### 4.1 Defining the objective function

You **need** to provide the objective function as a function named `f` which prototype is:

```
double f (const DataPoint x)
```

`DataPoint` is defined in file `stosoo_optim.h`. It is merely a `double *`.

...

You *may* provide an initialization function which is called once before the first call to the objective function. Its prototype is very simple:

```
void f_init (void)
```

In this function, you may do whatever you want.

As for the name of the function implementing the objective function, the name of the initialization function can not be parameterized.

### 4.2 Defining its domain

- its dimension
- its origins
- its ranges along each coordinate axis

A function is provided to set these 3 information at once. Its prototype is:

```
void set_domain (const size_t dimension, const double *origins,
                 const double *ranges)
```

...

### 4.3 Running the example

1. Fix the `Makefile` ...
2. Compile it: a mere `make` should do it.
3. Run it...

### 4.4 Playing with a toy function

`double f_maximum (void)` returns $f(x^*)$. It is used to compute the regret. By default, it returns a NaN. As shiped, the `main` function tests whether this function returns a NaN and if not, computes and prints out the regret; if this function returns NaN, nothing is printed out.

## References

[1] Valko, Carpentier, Munos, Stochastic Simultaneous Optimistic Optimization, *Proc. ICML*, 2013