StoSOO: yet an other implementation.

Philippe Preux
philippe.preux@univ-lille3.fr

June 5, 2014

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

This note has been updated to document version 0.3 of StoSOO.

Pre-requisites: from a practical point of view, to be able to take advantage of this note you should know how to write a (simple) program in C, use the `gcc` compiler to obtain an executable file, how to run that program, and (very) basic abilities in the usage of `make`.

1 Getting the code

The code of this implementation of StoSOO is available at this url:

`http://www.grappa.univ-lille3.fr/~ppreux/software/StoSOO/`

2 Installation

Please note that the installation has 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.0.3.tgz
$ cd stosoo.v.0.3
$ make
```

You will get a few warning messages when doing the `make`, nothing important about it.

This produces 3 executables: `stosoo_2sine`, `stosoo_garland`, and `stosoo_para` which names are pretty straightforward (para is a paraboloid to test > 1 dimension).

3 Usage

For example:

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

produces something like:

```
0.611111: 0.897011 0.100761
0.611111: 0.821933 0.175839
0.574074: 0.920353 0.0774195
0.574074: 0.920353 0.0774195
0.574074: 0.964724 0.0330487
0.574074: 0.964724 0.0330487
0.471193: 1.0821 0.0843291
0.471193: 1.0821 0.0843291
0.471193: 1.0821 0.0843291
0.471193: 1.0821 0.0843291
0.471193: 1.0821 0.0843291
0.471193: 1.0821 0.0843291
0.574913: 1.06444 0.0666646
0.574998: 1.03546 0.0376829
```

Each line corresponds to a certain amount of function evaluations having been done. Each line contains:

- the location of the current optimum (x_t^* =`0.611111` in the first line),
- the value of the objective function at that point ($f(x_t^*)$ = `0.897011` in the first line),
- the absolute difference between this value and the optimal one (if this optimal value is known, which is true only for toy functions).

Each line corresponds respectively to 0.01, 0.02, 0.03, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0 times the maximum number of function evaluations. Hence, the last line contains the optimum found using the full budget of function evaluations.

The arguments on the command line have the following meaning:

- `-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 dimension, you may test:

```
$ ./stosoo_para -N 1000 --dim 7 --ns .01
```

which produces something like:

```
# optimum set at (1, 0.5, 0.333333, 0.25, 0.2, 0.166667, 0.142857)
0.5 0.5 0.5 0.166667 0.5 0.5 0.166667: -0.484454 NA
0.5 0.5 0.5 0.166667 0.5 0.5 0.166667: -0.479902 NA
```

```
0.5 0.5 0.5 0.166667 0.166667 0.5 0.166667: -0.404462 NA
0.5 0.5 0.5 0.166667 0.166667 0.5 0.166667: -0.404462 NA
0.833333 0.5 0.5 0.5 0.166667 0.5 0.5: -0.357781 NA
0.833333 0.5 0.5 0.5 0.166667 0.5 0.462963: -0.33281 NA
0.833333 0.5 0.5 0.166667 0.166667 0.5 0.277778: -0.193062 NA
0.833333 0.5 0.5 0.166667 0.166667 0.5 0.277778: -0.193062 NA
0.833333 0.5 0.5 0.166667 0.166667 0.5 0.277778: -0.193062 NA
0.833333 0.5 0.5 0.166667 0.166667 0.5 0.277778: -0.193062 NA
0.833333 0.5 0.5 0.166667 0.166667 0.5 0.277778: -0.193062 NA
0.833333 0.5 0.5 0.166667 0.166667 0.5 0.277778: -0.193062 NA
0.833333 0.5 0.5 0.0555556 0.166667 0.166667 0.277778: -0.112897 NA
0.944444 0.5 0.5 0.166667 0.166667 0.166667 0.0555556: -0.0462446 NA
```

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

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

Each line should be interpreted in the same mannor as in the previous example. However, the dimension of the search space being 7, each optimum is made of 7 components.

In this example, we did not define the maximum of the function. So, the difference between the optimum found by StoSOO and the optimum can not be computed, hence the `NA`. (Obviously, this optimal value can be computed, this is just for the sake of the example.)

4 More on usage

4.1 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`.

4.2 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 δ

- `--soo` sets the SOO mode, that is assuming the obective function is deterministic (each point is evaluated only once)
- `--hmax h` where `h` is the maximum depth of the split tree.

4.3 Minimizing a function

Minimizing a function is just maximizing the opposite of the function. So, there are two things to do:

1. in the objective function evaluation, return the opposite of the value of the function;
2. use the `--minimize` argument on the command-line (this has absolutly no influence on the optimization process, only on the output of StoSOO).

We provide a source file named `objective_function_template.c` that serves as a template to define the objective function. Please, open it up with your favorite text editor and read it; it is pretty short and simple to understand.

5 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 to use it to test StoSOO, you may take advantage of this information to assess the algorithm. See section "Playing with a toy function".

5.1 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 *`.

`last_fct_evaluation ()` returns the value returned by the last function evaluation. The objective function is not evaluated again, since we consider that functions are stochastic, and since each call to the function evaluation counts.

You may check examples provided in the tarball contained in files `2sine.c`, `garland.c`, and `paraboloid.c`. These sources are very simple.

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 (const size_t N, const size_t d)
```

where `N` is the number of function evaluations, and `d` is the dimension of the problem.

With regards to previous versions of StoSOO, please note that this function has changed: it used to take no argument.

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.

5.2 Writing a program to optimize a function

To optimize a function with StoSOO, you have to:

- define its domain
- initialize StoSOO
- call StoSOO for optimization
- do whatever you wish with the result of the optimization

We provide a template source file named `stosoo.c`. The easiest way to begin with is to edit this file. There will be very little to do to make it optimize the function you wish to optimize.

Go directly to the `main` function.

First we define the domain of optimization. This is detailed in the next section. Afterwards, the function named `yaStoSOO_optim_init ()` is called to optimize the optimizer; then, the function named `yaStoSOO_optim ()` is called to perform the optimization. You do not have anything to modify: just leave these two calls as they appear. Next is the use of the optimum that have been found.

Naming convention : you'll notice that all objects (function, or variable) related to StoSOO have their name beginning with the character string `yaStoSOO_`. We decided that the objective function being not part of StoSOO itself, their names do not begin by this string of characters.

5.2.1 Defining its domain

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

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

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

5.3 Running the example

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

5.4 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.

6 Citing this software

If you use this software in work that leads to a publication, we would appreciate it if you would kindly cite us in your manuscript as:

Philippe Preux, *StoSOO: yet an other implementation*, `http://www.grappa.univ-lille3.fr/~ppreux/software/StoSOO/`

References

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