Exercise: Derivative Free Optimization

Step-size Adaptive Stochastic Search Algorithms

**Summer School "Design and Optimization
Under Uncertainty of Large Scale Numerical Models"**

Anne Auger, anne.auger@inria.fr
Asma Atamna, asma.atamna@inria.fr
Dimo Brockhoff, dimo.brockhoff@inria.fr

I Introduction

We are going to test the convergence of several algorithms on some test functions, in particular on the so-called sphere function

$$f_{\text{sphere}}(\mathbf{x}) = \sum_{i=1}^{n} \mathbf{x}_i^2$$

and the ellipsoid function

$$f_{\text{elli}}(\mathbf{x}) = \sum_{i=1}^{n} (100^{\frac{i-1}{n-1}} \mathbf{x}_i)^2 \ .$$

1. What is the condition number associated to the Hessian matrix of the functions above? Are the functions ill-conditioned?

2. Implement the above functions `fpshere` and `felli` in python. They should both take a search point $x \in \mathbb{R}^n$ as input and return $f(\mathbf{x})$. Most likely, you want to consider using `numpy` arrays as inputs. Python commands like `sum` and `arange` might be handy here.

II Adaptive step-size algorithms

The $(1+1)$-ES algorithm is one of the simplest stochastic search methods for numerical optimization. We will start by implementing a $(1+1)$-ES with constant step-size. The pseudo-code of the algorithm is given by

```
Initialize $\boldsymbol{x} \in \mathbb{R}^n$ and $\sigma > 0$
while not terminate
    $\mathbf{x}' = \mathbf{x} + \sigma\mathcal{N}(\mathbf{0}, \boldsymbol{I})$
    if $f(\mathbf{x}') \leq f(\mathbf{x})$
        $\mathbf{x} = \mathbf{x}'$
```

where $\mathcal{N}(\mathbf{0}, \boldsymbol{I})$ denotes a Gaussian vector with mean $\mathbf{0}$ and covariance matrix equal to the identity.

1. Implement the algorithm in python. To this end, you should define a function that takes as input a function to be optimized (like the two implemented previously), an initial search point $\mathbf{x} \in \mathbb{R}^n$, an initial step-size $\sigma \in \mathbb{R}_{>0}$ and a maximum number of function evaluations as stopping criterion and that returns a `numpy` array in which you have written at each iteration the best-so-far objective function value.

2. Use the algorithm to minimize the sphere function in dimension $n = 5$. We will take as initial search point $\mathbf{x}^0 = (1, \dots, 1)$ [`x=numpy.ones((1,n))`], as initial step-size $\sigma = 10^{-3}$ and as stopping criterion a maximum number of function evaluations equal to $2 \cdot 10^4$.

3. Plot the evolution of the function value of the best solution versus the number of iterations (or function evaluations). We will use a log scale for the y-axis. The easiest will be to use the `semilogy` functionality of `matplotlib`:

```
from matplotlib import pyplot as plt
plt.semilogy(X, Y)
plt.show()
```

4. Explain the three phases observed in the figure.

   To accelerate the convergence, we will implement a step-size adaptive algorithm, i.e. the step-size σ is not fixed once for all. The method to adapt the step-size, we investigate here, is called one-fifth success rule. The pseudo-code of the $(1 + 1)$-ES with one-fifth success rule is given by:

```
Initialize $\boldsymbol{x} \in \mathbb{R}^n$ and $\sigma > 0$
while not terminate
    $\boldsymbol{x'} = \boldsymbol{x} + \sigma \mathcal{N}(\mathbf{0}, \boldsymbol{I})$
    if $f(\boldsymbol{x'}) \leq f(\boldsymbol{x})$
        $\boldsymbol{x} = \boldsymbol{x'}$
        $\sigma = 1.5\,\sigma$
    else
        $\sigma = (1.5)^{-1/4}\sigma$
```

5. Implement the (1+1)-ES with one-fifth success rule and run the algorithm on the sphere function $f_{\text{sphere}}(x)$ in dimension 5 ($n = 5$) using $\mathbf{x}^0 = (1, \dots, 1)$, $\sigma_0 = 10^{-3}$, and as stopping criterion a maximum number of function evaluations equal to $6 \cdot 10^2$.

6. To visualize the optimization, plot the evolution of the square root of the best function value at each iteration versus the number of iterations. Use a logarithmic scale for the y-axis. Compare to the plot obtained in Question 3. Plot also on the same graph the evolution of the step-size.

7. Use the algorithm to minimize the function f_{elli} in dimension $n = 5$. Plot the evolution of the objective function value of the best solution versus the number of iterations. Why is the $(1+1)$-ES with one-fifth success much slower on f_{elli} than on f_{sphere}?

8. Repeat the same experiment of item 5 and 6 again on the function

$$f_{\text{Rosenbrock}}(x) = \sum_{i=1}^{n-1} (100(x_i^2 - x_{i+1})^2 + (x_i - 1)^2)$$

   where you start from $x^0 = (0.5, \dots, 0.5)$.

9. We now consider the functions, $g(f_{\text{sphere}})$ and $g(f_{\text{elli}})$ where $g : \mathbb{R} \to \mathbb{R}, y \mapsto y^{1/4}$. First of all, write the function `g` that can do the transformation on its own and that can be combined with the above `fsphere` and `felli`.

10. Modify your implementation of the (1+1)-ES in Questions 5–7 so as to save at each iteration the distance between `x` and the optimum. Plot the evolution of the distance to the optimum versus the number of function evaluations on the functions f_{sphere} and $g(f_{\text{sphere}})$ as well as on the functions f_{elli} and $g(f_{\text{elli}})$. What do you observe? Explain.

III Running and Understanding CMA-ES

Download and install the python module of the CMA-ES algorithm from the webpage of Nikolaus Hansen (the main author of the algorithm):

`http://www.lri.fr/~hansen/cmaes_inmatlab.html`

The easiest is to type `python -m pip -U install cma` within your terminal in case you have already `pip` installed. Note that if you work on the provided desktops, the command `python` is linked with a Python 3 installation and that you must use a Python 3 jupyter notebook in this case.

1. Run the CMA-ES algorithm in dimension 10 to minimize the following functions
   - $f_{\text{elli}}(\mathbf{x}) = \sum_{i=1}^{n}((10^3)^{\frac{i-1}{n-1}}\mathbf{x}_i)^2$
   - $f_{\text{tablet}}(\mathbf{x}) = 10^6\mathbf{x}_1^2 + \sum_{i=2}^{n}\mathbf{x}_i^2$

   The main entry point for us to CMA-ES here will be `cma.fmin` which takes an objective function to be optimized, an initial starting solution, and the initial step size as mandatory arguments (type `cma.fmin?` to get more detailed help). The above functions are as well already available with the `cma` module via `cma.fcts.elli` and `cma.fcts.tablet` respectively. Typing

   ```
   cma.plot()
   cma.show()
   ```

   will display all kinds of (graphical) information from the algorithm run and in particular the evolution of the mean vector, the step-size and the covariance matrix adapted in the CMA-ES algorithm.
2. Explain the different plots that appear on the screen.
3. Identify and explain the two main (convergence) phases observed.
4. What is the relationship between the eigenvalues of the covariance matrix in the end and the eigenvalues of the Hessian matrix of the functions?
5. Connect the asymptotic convergence rate on the convex quadratic function that corresponds to the slope of the last part of the convergence graph with the convergence rate on the sphere function. Explain.
6. The function $f_{\text{ellirot}}(\mathbf{x})$ is defined by $f_{\text{ellirot}}(\mathbf{x}) = f_{\text{elli}}(P\mathbf{x})$ where P is a rotation matrix (sampled uniformly among the rotation matrices). Run the CMA-ES algorithm on f_{ellirot} and on f_{elli}. Remember, that several functions are already provided with the `cma` module, in particular also `cma.fcts.ellirot`. Understand and explain the differences observed in the graphical output.
7. Compare now the convergence rate of CMA-ES and of the $(1+1)$-ES with one-fifth success rule on $f_{\text{elli}-2}(\mathbf{x}) = \sum_{i=1}^{n}((10^2)^{\frac{i-1}{n-1}}\mathbf{x}_i)^2$ for $n = 10$. For this you can for instance report the number of function evaluations that both algorithms need to reach 10^{-6} for 6 different runs. Explain the differences observed.
8. We consider now the function

   $$f_{\text{rastrigin}}(\mathbf{x}) = 10n + \sum_{i=1}^{n}(\mathbf{x}_i^2 - 10\cos(2\pi\mathbf{x}_i))$$

   Show that the function is multimodal. We remind that a function to be minimized is multimodal if it has more than one local optimum.
9. The default population size (the parameter λ) in CMA-ES equals $4 + \lfloor(3\log(n))\rfloor$. Run 5 times the CMA-ES algorithm with its default population size to minimize the Rastrigin function in dimension 10 starting with a point sampled uniformly at random within $[0,1]^n$ and with initial step-size equal to 10. Measure the success probability of the algorithm reaching a function value of 10^{-6} or smaller. Realize the same experiment by multiplying the default population size by $2, 4, 8, 16, 32$. What do you observe? Explain.