DOI: 10.15439/2015F47
ACSIS, Vol. 5

Comparative Evaluation of the Stochastic Simplex Bisection Algorithm and the SciPy.Optimize Module

Christer Samuelsson
German Research Center for Artificial Intelligence
Email: christer.samuelsson@dfki.de

***Abstract*—The stochastic simplex bisection (SSB) algorithm is evaluated against the collection of optimizers in the Python SciPy.Optimize module on a prominent test set. The SSB algorithm greatly outperforms all SciPy optimizers, save one, in exactly half the cases. It does slightly worse on quadratic functions, but excels at trigonometric ones, highlighting its multimodal prowess. Unlike the SciPy optimizers, it sustains a high success rate. The SciPy optimizers would benefit from a more informed metaheuristic strategy and the SSB algorithm would profit from quicker local convergence and better multidimensional capabilities. Conversely, the local convergence of the SciPy optimizers is impressive and the multimodal capabilities of the SSB algorithm in separable dimensions are uncanny.**

I. Introduction

Stochastic optimization [1], i.e., using randomness to guide search, is currently popular: genetic algorithms [2], particle swarm optimization [3], and ant-colony optimization [4] are celebrity approaches. These schemes are not only applied to discrete problems, but also to continuous ones, especially for objective functions where the gradient and Hessian are not readily available, e.g., where the function value is only obtainable by expensive simulation. They are also used to optimize highly multimodal functions.

We here compare the performance of the stochastic simplex bisection (SSB) algorithm [5] with that of the optimizers of the Python SciPy.optimize module [6]. The former employs a common stochastic optimization scheme, but unlike other stochastic approaches, it applies the scheme to search space regions, rather than to individual points. The latter are high-quality, local optimizers that are available with most Python distributions. The SSB algorithm has not previously been tested against other optimizers. We here seek to rectify this.

The rest of the article is organized as follows. Section II presents the stochastic simplex bisection algorithm. Section III describes the optimizers of the Python SciPy.Optimize module. Section IV details the experimental setup, and reports and analyses the findings. It also contains a short digression on the employed success criterion and its success regions.

II. The Stochastic Simplex Bisection Algorithm

Consider the simple problem where we wish to minimize $f(x)$, which is strictly convex on $\mathbf{R}$. Assume further that we have found $x_1 < x_3 < x_5$, where $f(x_1) \geq f(x_3) \leq f(x_5)$, i.e., that we have found an interval that has an interior point x_3 with a smaller function value than its end points. [1]

Algorithm for Convex Functions

```
**Choose** $x_2 \in ]x_1, x_3[$ and $x_4 \in ]x_3, x_5[$, i.e., choose interior points $x_2$ and $x_4$.
**If** $f(x_2) \leq f(x_3)$, recurse on $x_1, x_2, x_3$.
**If** $f(x_4) \leq f(x_3)$, recurse on $x_3, x_4, x_5$.
**Otherwise**, recurse on $x_2, x_3, x_4$.
```

We thus recurse on the subinterval that has an interior point with a smaller function value than its end points. [2]

The SSB algorithm generalizes this to non-convex functions in n dimensions. It generalizes an interval to a simplex, rather than to a hyperbox, The latter has 2^n corners, and bisecting it requires computing the function value in 2^{n-1} new corners. The former has only $n + 1$ corners, and bisecting it only requires calculating the function value in one new point.

Core SSB Algorithm

```
**Given** a set $\{T_k\}$ of non-overlapping simplexes that partition the original simplex $T_0$, each equipped with a positive score $s_k$.
**Select** the next simplex $T_k$ to bisect at random with probability $\frac{s_k}{\sum_{k'} s_{k'}}$.
**Select** a bisection point at random roughly in the middle of the longest edge of $T_k$.
**Replace** $T_k$ with its two offspring.
```

This algorithm is complete in the sense that no portion of the search space is ever discarded, and it avoids redundancy by using non-overlapping simplexes. These are two common pitfalls of stochastic optimization. The algorithm still reaps the benefits of stochastic search in exploring more promising regions earlier, in average, while granting also less promising regions a non-zero chance of being explored.

It is often desirable to start from a hyperbox. For example, constraints often take the form of bounds on the individual variables of each dimension. The tested SSB algorithm restarts the core SSB algorithm repeatedly from a hyperbox created in the previous iteration. It uses an outer loop over epochs that maintains the hyperbox, and an inner loop over rounds that implements the core SSB algorithm. Note that although partitioning a hyperbox into a set of simplexes is trivial in

[1]By strict convexity, one of the two end point values must be strictly larger, i.e., either $f(x_1) > f(x_3)$ or $f(x_3) < f(x_5)$, or both.

[2]If $f(x_2) = f(x_3)$, the algorithm could be clever and instead recurse on x_2, x_{23}, x_3, with $x_{23} \in]x_2, x_3[$, where, by necessity $f(x_2) > f(x_{23}) < f(x_3)$, due to strict convexity. Similarly for $f(x_4) = f(x_3)$.

two dimensions, it is a challenging and time-consuming task in higher ones, see [7] and [8].

A. Outer Loop: Maintaining a Hyperbox

The tested SSB algorithm consists of two nested loops. The outer loop over epochs maintains a hyperbox. Each *epoch* runs the inner loop over rounds, where each *round* consists of one simplex bisection, see Section II-B. The terms *best point*, *very best point*, and *epoch phases* will be defined shortly.

The hyperbox is modified after each epoch. If the elapsed epoch had enough best points, this is a hyperbox that contains all best points and the very best point as interior points. Otherwise, the previous hyperbox is increased in size and re-centered around the current very best point, which may have changed during the epoch. In the tested SSB algorithm, the padding is 100 percent of the interval length in each dimension, when there are enough best points, and the old interval length is quadrupled, when there aren't. It turns out that in the former case, the simple scheme of updating the lower and higher bounds of the hyperbox in each dimension, for each new best point, works well in practice.

A *best point* is any point found during the second phase of an epoch that is the best this far in that epoch. The best points thus start over each epoch. The *very best point*, on the other hand, is the globally best point found in any round of any epoch.

The *first phase* of each epoch consists of the first quarter of its rounds. The rest of its rounds constitute the *second phase.* Other choices than one quarter were tested, but found less effective, albeit one third only marginally so. Typical figures are 60 epochs of 500 rounds each, but this can be varied with the search conditions, cf. Section IV. Higher dimensions require more rounds; fewer function evaluations entail fewer epochs and much fewer rounds.

The outer loop may seem somewhat ad hoc. It captures the idea, that if there has been non-trivial local improvements, search should focus on these improvements, yet also consider the globally best point found. We note that any new best point must be a bisection point, or the midpoint of a simplex, with a lower function value than its corner points. In one dimension, these cases coincide. For convex functions, such an interval must contain the minimum, which our introductory algorithm exploits. For non-convex functions, or in several dimensions, the area surrounding such a point merits further investigation.

B. Inner Loop: Bisecting Simplexes

The inner loop over rounds bisects simplexes. Each bisection replaces one simplex with two new ones. In the first phase, the simplexes are processed as a first-in-first-out (FIFO) queue to create an initial grid. In the second phase, the next simplex to bisect is selected randomly with probability

$$\frac{s_k}{\sum_{k'} s_{k'}}$$

where s_k is the score of the kth simplex. Binary trees provide an efficient way of stochastic selection that allows adding and deleting scored elements. Indexing the K simplexes in a binary tree yields $O(\log_2 K)$ time complexity for lookup, addition, and deletion, whereas naively using a list swells this to $O(K)$.

We define the simplex scores s_k as follows. Let $\{T_k = \langle \mathbf{x}_k^{(1)}, \ldots, \mathbf{x}_k^{(n+1)} \rangle\}$ be a collection of n-dimensional simplexes with (dropping the index k for clarity),

$$\bar{\mathbf{x}} = \frac{1}{n+1}\sum_{i=1}^{n+1} \mathbf{x}^{(i)} \quad ; \quad \bar{f} = \frac{f(\bar{\mathbf{x}}) + \sum_{i=1}^{n+1} f\left(\mathbf{x}^{(i)}\right)}{n+2}$$

where $\bar{\mathbf{x}}$ is the midpoint and $\bar{f}$ is the average function value over the corners and the midpoint.

$$\begin{aligned} f^- &= \min\left(f(\bar{\mathbf{x}}), \min_i f\left(\mathbf{x}^{(i)}\right)\right) \quad ; \quad \delta = \frac{\bar{f} - f^-}{4} \\ f^\star &= f^- - \delta \quad ; \quad f^\star \leftarrow \max\left(0, f^\star - f_{\text{vb}}\right) \end{aligned}$$

$f^\star$ is a combined measure of the lowest function value f^- and an estimate δ of how much it might potentially decrease, judging by the average value $\bar{f}$. We make $f^\star$ offset-invariant by subtracting the lowest function value f_{vb}, in the very best point, then cap it to be at least zero.

$$\begin{aligned} s &= l \cdot \exp\left(-\lambda f^\star\right) \quad ; \quad l = \max_{ij} \left|\mathbf{x}^{(i)} - \mathbf{x}^{(j)}\right| \\ \lambda &= \lambda_0 \cdot \max\left(1, \frac{1}{f_{\text{w}} - f_{\text{vb}}}\right) \end{aligned}$$

The simplex score is $l \cdot \exp(-\lambda f^\star)$, where l is the length of its longest edge. λ is λ_0 times the reciprocal of the difference between the highest function value f_{w} of the corners of the bounding box and the lowest function value f_{vb}. This renders the score scale-invariant. $\frac{\lambda}{\lambda_0}$ is capped to be at least one; λ_0 defaults to one. Thus, all simplexes must be rescored whenever a new very best point is found. As this happens rather seldom, it incurs very little overhead in practice.

Each round only creates two new simplexes: the longest edge of the selected simplex is bisected. All corners remain the same, save one of the two connected by this edge. Let these corners be $\mathbf{x}^{(i)}$ and $\mathbf{x}^{(j)}$. The edge bisection point $\bar{\mathbf{x}}'$—the new corner—is also randomized to counter-act any symmetries of the objective function $f(\mathbf{x})$ in its argument $\mathbf{x}$:

$$\bar{\mathbf{x}}' = (0.5 + \theta)\mathbf{x}^{(i)} + (0.5 - \theta)\mathbf{x}^{(j)} \quad \text{for} \quad \theta \sim \mathrm{U}(-\alpha, \alpha)$$

An empirically good choice is $\alpha = 0.05$ (max 10% randomness). $\bar{\mathbf{x}}'$ replaces $\mathbf{x}^{(i)}$ in one offspring simplex and $\mathbf{x}^{(j)}$ in the other one. Thus only three new function values need be calculated for each bisection: $f(\bar{\mathbf{x}}')$ and the function values of the midpoints of the two new simplexes.

C. Related Work

The SSB algorithm uses a typical stochastic optimization scheme. It maintains a set of elements, each with a positive score; randomly selects some elements based on the scores; uses these elements to explore the search space, often creating new elements in the process; and updates the set of elements and their scores according to the findings. The scheme is however here applied to regions of the search space, not to points in it, as in, e.g., [9], [10], [11], [12], [13], and [14].

The simplex method [15], Chapter 9, doesn't actually use simplexes. To create new points, controlled random search [11] generates random simplexes. These may overlap and are not guaranteed to cover the search space. Nor are they subdivided. DIRECT [16] uses hyperboxes that partition the search space. It avoids the 2^n complexity by directional search from their midpoints, ignoring their corners. Each hyperbox potentially containing the global minimum is trisected, rather than bisected, along each dimension in turn. There is no randomized selection.

III. The Opimizers of the SciPy.Optimize Module

The optimizers from the Python SciPy.Optimize module that we tested were code named "Nelder-Mead," "Powell," "CG," "BFGS," "L-BFGS-B," "TNC," and "SLSQP." We did not provide the gradient nor the Hessian of the objective functions. Whenever an optimizer used these, it had to estimate them itself numerically.

The SciPy module also contains the "Anneal," "COBYLA," "dogleg," and "trust-ncg" optimizers. Anneal proved too slow and performed too poorly to be included. COBYLA was not robust enough for large-scale testing. The dogleg and trust-ncg optimizers require explicit gradients, which were not provided.

Nelder-Mead uses the simplex method, see [17], [18]. Powell is a modification of Powell's algorithm, a conjugate direction method, that performs sequential one-dimensional minimizations, see [19], [20]. CG uses a nonlinear conjugate gradient method by Polak and Ribiere, a variant of the Fletcher-Reeves algorithm described in [21], pp. 120–122.

BFGS uses the quasi-Newton method of Broyden, Fletcher, Goldfarb, and Shanno, see [21], p 136, and L-BFGS-B uses the L-BFGS-B algorithm for bound constrained minimization, see [22], [23]. TNC uses a combined Newton and conjugate gradient method, aka a truncated Newton method, see [21], p 168, and [24]. SLSQP uses sequential least squares programming, see [25].

IV. Experiments

A very reasonable baseline, seeing that the SSB algorithm essentially adds structure to randomized search, is to evaluate it against unstructured randomized search conducted by each ScyPy optimizer. In each trial, the ScyPy optimizer was repeatedly restarted in a new random point, until a limit on the total number of function evaluations had been exceeded.

A. Experimental Setup

We tested all *two-dimensional* objective functions of Figure 2 (last page), most of which are from [26]. Function 0 comes from [5], Function 20 from [27], and Functions 21, 22, 24, and 25 are of our own design. All functions have unique global minima, except Function 0, due to symmetry in x and $x + y$, and Functions 12, 14, and 17, which have four global minima, due to symmetry in $\pm x, \pm y$. Functions 16 and 17 were corrected using [28]. We did not provide the gradient nor the Hessian of these objective functions.

We investigated the frugal function evaluation scenario, where computing function values comes at a premium, and restricted the number of function evaluations to four thousand. The domain was $[-80, 120] \times [-80, 120]$, which is typically larger than that of the test set: often $[-10, 10] \times [-10, 10]$ or even $[-5, 5] \times [-5, 5]$. It was made asymmetric in x and y, since many test functions have their global minimum in $\mathbf{x} = \mathbf{0}$. As in [5] and [28], success was defined as finding any argument with a function value within 10^{-6} of the known global minimal value. See Section IV-C for a discussion on this success criterion.

No attempt was made to optimize the optimizers. The SciPy optimizers were used with their default parameter settings, and the search parameters of the SSB algorithm were taken over from [5], Section 5.3: λ_0 was set ten; epoch phase one consisted of the first five of its 50 rounds in total; once the system had made 4000 function evaluations, the current epoch and the algorithm were terminated.

In each trial, the SciPy optimizer was repeatedly run starting from a randomized point and the best candidate point of the trial was updated if the new one was better. The trial continued until 4000 function evaluations had been exceeded. The best result in the trial determined success or failure.

B. Experimental Results

Table I shows their respective success rates in 1000 trials. To analyze these results, we need to consider the nature of each test objective function. These can be classified into:

- unimodal quadratic forms, Fcns 2, 6, 8;
- oligo-modal [3] polynomials, Fcns 3, 4, 5, 10, 18;
- multimodal damped trigonometrics, Fcns 0, 1, 12–17, 25;
- mixed trigonometrics and quadratics, Fcns 9, 11, 20, 24;
- other (unimodal) functions, Fcns 7, 21, 22.

In exactly half of the cases (Functions 0, 1, 9, 11–17, and 22–25), the SSB algorithm stands head and shoulders above the competition, except for Powell's algorithm in two of these cases, namely Functions 1 and 9. *This includes all seriously multimodal functions.* Clearly, when faced with numerous local optima, the extra structure afforded by the SSB algorithm outweighs the benefit of highly accurate local search.

Conversely, it includes no unimodal or oligo-modal function, except Function 22. This stands to reason, as any adept local optimizer should succeed for these functions, even when starting from a randomized point some distance away. See Section IV-C for a discussion of Functions 2, 21, and 22.

Function 7 is very hard, and defeats all optimizers. It has a concave valley, kinks, and a very anisotropic variable coupling. The gradient is ill-defined and unbounded in the valley bottom, and especially ill-behaved in the optimum. *Memento mori.*

Viewing the results from another angle, we note that the SSB algorithm performs under 50% in only five cases of 22. In two of these, it still outperforms all other algorithms, and in a third case, all algorithms come up empty-handed. In one of the two remaining cases, Function 3, the SSB algorithm

[3]. . . apologies for mixing Greek and Latin roots. . .

	Optimizer							
Fcn	BFGS	CG	L-BFGS-B	Simplex	Powell	SLSQP	TNC	SSB
0	0	0	0	0	0	0	0	**27**
1	45	57	76	17	835	52	52	**896**
2	**1000**	**1000**	**1000**	**1000**	**1000**	**1000**	**1000**	989
3	999	**1000**	**1000**	**1000**	**1000**	996	921	188
4	521	725	741	**998**	109	975	716	661
5	949	493	**1000**	**1000**	951	**1000**	**1000**	765
6	**1000**	**1000**	**1000**	**1000**	**1000**	**1000**	**1000**	862
7	0	0	0	0	0	0	0	0
8	**1000**	**1000**	**1000**	**1000**	966	**1000**	**1000**	872
9	50	89	140	545	**1000**	149	34	*897*
10	448	**1000**	960	**1000**	984	920	**1000**	844
11	297	239	289	136	222	269	202	**638**
12	397	259	440	380	251	391	310	**890**
14	91	112	251	52	136	201	57	**808**
16	42	34	75	9	109	159	163	**857**
17	83	68	147	17	10	243	34	**815**
18	996	**1000**	**1000**	**1000**	**1000**	**1000**	998	906
20	125	156	160	975	**1000**	460	48	256
21	**1000**	750	**1000**	0	**1000**	**1000**	585	829
22	0	0	0	0	304	0	0	**626**
24	2	0	13	14	111	0	2	**139**
25	121	19	265	73	205	221	635	**803**

TABLE I
SUCCESS RATE IN 1000 TRIALS OF THE SCIPY OPTIMIZERS AND THE SSB ALGORITHM.

is hampered by quadratic terms and a dominating variable coupling, and it lags behind the competition.

The other remaining case is Function 20. It has dominating quadratic terms and strong trigonometric confusion terms. The SSB algorithm outperforms half of the SciPy optimizers, but not the SLSQP, Simplex, and Powell optimizers, the latter two which excel. By contrast, all SciPy optimizers perform under 50% in more than half of the cases, except Powell's algorithm, which performs under 50% in exactly half of the 22 cases.

C. A Digression on Success Regions

The success criterion $|f(\mathbf{s}) - f(\mathbf{x}^*)| < \epsilon$, where $\mathbf{x}^*$ is a known global optimum, is certainly reasonable for practitioners, who care mainly about finding a good enough solution $\mathbf{s}$. But when evaluating optimization algorithms, one must be careful not to draw incorrect conclusions from this criterion.

For example, when comparing performances on Functions 2, 21, and 22 in Table I, one might be tempted to conclude that the discontinuity in the gradient in the global optimum caused by the absolute value does some damage to gradient-based methods, and wreaks havoc with the simplex method, while the additional concavity of the square root foils all algorithms, save Powell's method and the SSB algorithm.

This analysis fails to take into account that the success region of Function 2 is a circle with diameter $2\sqrt{\epsilon}$, while that of Function 21 is a square with diagonal 2ϵ, and that of Function 22 is a concave diamond shape with diagonal $2\epsilon^2$. The alchemy-like Figure 1 illustrates this for the unrealistically large $\epsilon = 0.5$. The ratios of their surface areas are $\pi\epsilon : 2\epsilon^2 : \frac{2}{3}\epsilon^4$. This makes the latter two increasingly harder to hit, which is a potentially greater difficulty for the optimizers, given strict limitations on the number of function evaluations.

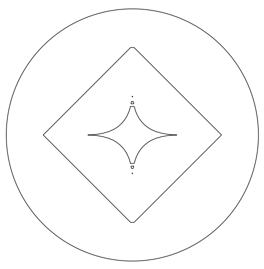
Fig. 1. $|f(x,y) - f(0,0)| = 0.5$ for Fcns 2, 21, and 22.

V. SUMMARY AND CONCLUSIONS

We evaluated the stochastic simplex bisection (SSB) algorithm against the optimizers of the Python SciPy.Optimize module on a prominent test set. The former employs a common stochastic optimization scheme, but unlike other stochastic approaches, it applies the scheme to search space regions, rather than to individual points. The latter are readily available, high-quality, local optimizers. This is the first evaluation of the SSB algorithm against other optimizers.

The experiments were conducted in two dimensions on a prominent test set. The domain was mostly larger, by an order of magnitude in each direction, than the domains indicated by the test set, and the number of function evaluations was limited to a few thousand. The SSB algorithm used the latter as a termination criterion and returned its very best point. Each ScyPy optimizer was repeatedly restarted in a random point, until it had exceeded the function evaluation limit, and the overall best result was returned. In both cases, if the returned solution was within 10^{-6} of the known global minimal value, it was deemed a successful trial. Each optimizer was tested 1000 times on each test function.

The SSB algorithm greatly outperformed all SciPy optimizers, save one, in exactly half the cases. It did slightly worse on polynomial functions, but excelled at trigonometric ones, highlighting its multimodal prowess. And unlike the SciPy optimizers, it sustained a high success rate.

We conclude that the SciPy optimizers would benefit from a more informed metaheuristic strategy and that the SSB algorithm would profit from quicker local convergence and better multidimensional capabilities. Conversely, the local convergence of the SciPy optimizers is impressive and the multimodal capabilities of the SSB algorithm in separable dimensions are uncanny.

ACKNOWLEDGMENT

This work was funded by the BAULT network and the METALOGUE project. BAULT, building and using language technology, is a multidisciplinary research community at the University of Helsinki. METALOGUE is a Seventh Framework Programme collaborative project funded by the European Commission, grant agreement number 611073 (http://www.metalogue.eu).

REFERENCES

[1] M. Wahde, *Biologically Inspired Optimization Algorithms*. WIT Press, 2008.

[2] D. E. Goldberg and J. H. Holland, "Genetic algorithms and machine learning," *Machine Learning*, vol. 3, no. 2-3, pp. 95–99, 1988. doi: 10.1023/A:1022602019183

[3] J. Kennedy and R. Eberhart, "Particle swarm optimization," 1995. doi: 10.1109/ICNN.1995.488968

[4] M. Dorigo, V. Maniezzo, and A. Colorni, "The ant system: Optimization by a colony of cooperating agents," *IEEE Trans. on Systems, Man, and Cybernetics*, vol. 26, no. 1, pp. 29–41, 1996. doi: 10.1109/3477.484436

[5] C. Samuelsson, "The stochastic simplex bisection algorithm," in *Procs. 15th Int.'l Conf. Computational Science*, ser. ICCS/15. Elsevier, 2015. doi: 10.1016/j.procs.2015.05.215 pp. 855–864. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877050915010236

[6] scipy, "scipy.optimize," http://http://docs.scipy.org/doc/scipy/reference/optimize.html#module-scipy.optimize, 2015, accessed: 2015-04-16.

[7] M. Haiman, "A simple and relatively efficient triangulation of the n-cube," *Discrete & Computational Geometry*, vol. 6, no. 1, pp. 287–289, 1991. doi: 10.1007/BF02574690. [Online]. Available: http://dx.doi.org/10.1007/BF02574690

[8] R. B. Hughes and M. R. Anderson, "Simplexity of the cube," *Discrete Mathematics*, vol. 158, no. 13, pp. 99 – 150, 1996. doi: http://dx.doi.org/10.1016/0012-365X(95)00075-8. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0012365X95000758

[9] Q. Duan, S. Sorooshian, and V. Gupta, "Effective and efficient global optimization for conceptual rainfall-runoff models," *Water resources research*, vol. 28, no. 4, pp. 1015–1031, 1992. doi: 10.1029/91WR02985

[10] N. Hansen and S. Kern, "Evaluating the CMA evolution strategy on multimodal test functions," in *Parallel Problem Solving from Nature VIII*, X. Yao *et al.*, Eds. Springer, 2004. doi: 10.1007/978-3-540-30217-9_29 pp. 282–291.

[11] P. Kaelo and M. M. Ali, "Some variants of the controlled random search algorithm for global optimization," *J. Optim. Theory Appl*, vol. 130, no. 2, pp. 253–264, 2006. doi: 10.1007/s10957-006-9101-0

[12] K. V. Price, R. M. Storn, and J. A. Lampinen, *Differential Evolution - A Practical Approach to Global Optimization*, ser. Natural Computing. Springer-Verlag, 2006, iSBN 540209506.

[13] A. I. Vaz and L. N. Vicente, "A particle swarm pattern search method for bound constrained global optimization," *J. of Global Optimization*, vol. 39, no. 2, pp. 197–219, Oct. 2007. doi: 10.1007/s10898-007-9133-5. [Online]. Available: http://dx.doi.org/10.1007/s10898-007-9133-5

[14] X.-S. Yang, "Firefly algorithms for multimodal optimization," in *Procs. 5th Int.'l Conf. Stochastic Algorithms: Foundations and Applications*, ser. SAGA'09. Springer-Verlag, 2009. doi: 10.1007/978-3-642-04944-6_14. ISBN 3-642-04943-5, 978-3-642-04943-9 pp. 169–178. [Online]. Available: http://dl.acm.org/citation.cfm?id=1814087.1814105

[15] N. Andréassson, A. Egrafov, and M. Patriksson, *An Introduction to Continuous Optimization*. Studentlitteratur, 2005.

[16] D. R. Jones, C. D. Perttunen, and B. E. Stuckman, "Lipschitzian optimization without the Lipschitz constant," *J. Optim. Theory Appl.*, vol. 79, no. 1, pp. 157–181, Oct. 1993. doi: 10.1007/BF00941892. [Online]. Available: http://dx.doi.org/10.1007/BF00941892

[17] J. A. Nelder and R. Mead, "A simplex method for function minimization," *The Computer Journal*, vol. 7, pp. 308–313, 1965. doi: 10.1093/comjnl/7.4.308

[18] M. H. Wright, "Direct Search Methods: Once Scorned, Now Respectable," in *Numerical Analysis 1995 (Proceedings of the 1995 Dundee Biennial Conference in Numerical Analysis)*, ser. Pitman Research Notes in Mathematics, D. F. Griffiths and G. A. Watson, Eds., vol. 344. Boca Raton, Florida: CRC Press, 1996. doi: 10.1.1.47.6891 pp. 191–208.

[19] M. J. D. Powell, "An efficient method for finding the minimum of a function of several variables without calculating derivatives," *The Computer Journal*, vol. 7, no. 2, pp. 155–162, 1964. doi: 10.1093/comjnl/7.2.155. [Online]. Available: http://comjnl.oxfordjournals.org/content/7/2/155.abstract

[20] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes 3rd Edition: The Art of Scientific Computing*, 3rd ed. New York, NY, USA: Cambridge University Press, 2007. ISBN 0521880688, 9780521880688

[21] J. Nocedal and S. J. Wright, *Numerical Optimization*, 2nd ed. New York: Springer, 2006.

[22] R. H. Byrd, P. Lu, J. Nocedal, and C. Zhu, "A limited memory algorithm for bound constrained optimization," *SIAM Journal on Scientific Computing*, vol. 16, no. 5, pp. 1190–1208, 1995. doi: 10.1137/0916069

[23] C. Zhu, R. H. Byrd, P. Lu, and J. Nocedal, "Algorithm 778: L-bfgs-b: Fortran subroutines for large-scale bound-constrained optimization," *ACM Trans. Math. Softw.*, vol. 23, no. 4, pp. 550–560, Dec. 1997. doi: 10.1145/279232.279236. [Online]. Available: http://doi.acm.org/10.1145/279232.279236

[24] S. Nash, "Newton-type minimization via the Lanczos method," *SIAM Journal on Numerical Analysis*, vol. 21, no. 4, pp. 770–788, 1984. doi: DOI:10.1137/0721052

[25] D. Kraft, *A software package for sequential quadratic programming*, ser. Deutsche Forschungs- und Versuchsanstalt für Luft- und Raumfahrt Köln: Forschungsbericht. Wiss. Berichtswesen d. DFVLR, 1988. [Online]. Available: http://books.google.fr/books?id=4rKaGwAACAAJ

[26] Wikipedia, "Test functions for optimization," http://en.wikipedia.org/wiki/Test_functions_for_optimization.html, 2014, accessed: 2014-12-14.

[27] K. Mullen, D. Ardia, D. Gil, D. Windover, and J. Cline, "DEoptim: An R package for global optimization by differential evolution," *J. of Stat. Software*, vol. 40, no. 6, pp. 1–26, 2011. [Online]. Available: http://www.jstatsoft.org/v40/i06/

[28] A. Gavana, "Global optimization benchmarks and AMPGO," http://infinity77.net/global_optimization, 2015, accessed: 2015-01-09.

Objective Functions

Fcns 21, 22, 24, 25 are novel; Fcn 0 from [5]; Fcn 20 from [27]; remainder from [26]. Fcns 16 and 17 corrected using [28].

$$f(\mathbf{x}) = f(x_1,\dots,x_n) = \prod_{k=1}^{n} f_0\left(\sum_{i=1}^{k} x_i\right) \quad \text{with} \tag{0}$$

$$f_0(x) = \sin(\sqrt{1+x^2}) \cdot \cos(2x(x+1)) \cdot \frac{\ln(1+x^2)}{\sqrt{1+x^2}}$$

$$f(\mathbf{x}) = 20\left(1-\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right)\right) + e - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right) \tag{1}$$

$$f(\mathbf{x}) = f(x_1,\dots,x_n) = \sum_{i=1}^{n} x_i^2 \tag{2}$$

$$f(\mathbf{x}) = f(x_1,\dots,x_n) = \sum_{i=1}^{n-1}\left(100\cdot(x_{i+1}-x_i^2)^2 + (x_i-1)^2\right) \tag{3}$$

$$f(x,y) = (1.5-x+xy)^2 + (2.25-x+xy^2)^2 + (2.625-x+xy^3)^2 \tag{4}$$

$$\begin{aligned} f(x,y) = {} & \left(1+(x+y+1)^2(19-14x+3x^2-14y+6xy+3y^2)\right)\cdot \\ & \cdot\left(30+(2x-3y)^2(18-32x+12x^2+48y-36xy+27y^2)\right) \end{aligned} \tag{5}$$

$$f(x,y) = (x+2y-7)^2 + (2x+y-5)^2 \tag{6}$$

$$f(x,y) = 100\sqrt{|y-0.01x^2|} + 0.01|x+10| \tag{7}$$

$$f(x,y) = 0.26(x^2+y^2) - 0.48xy \tag{8}$$

$$f(x,y) = \sin^2(3\pi x) + (x-1)^2\left(1+\sin^2(3\pi y)\right) + (y-1)^2\left(1+\sin^2(2\pi x)\right) \tag{9}$$

$$f(x,y) = 2x^2 - 1.05x^4 + \frac{x^6}{6} + xy + y^2 \tag{10}$$

$$f(x,y) = -\cos(x)\cos(y)\exp\left(-(x-\pi)^2-(y-\pi)^2\right) \tag{11}$$

$$f(x,y) = -0.0001\left(|\sin(x)\sin(y)|\exp\left(\left|100-\frac{\sqrt{x^2+y^2}}{\pi}\right|\right)+1\right)^{0.1} \tag{12}$$

$$f(x,y) = -|\sin(x)\sin(y)|\exp\left(\left|1-\frac{\sqrt{x^2+y^2}}{\pi}\right|\right) \tag{14}$$

$$f(x,y) = 0.5 + \frac{\sin^2(x^2-y^2)-0.5}{1+0.001(x^2+y^2)^2} \tag{16}$$

$$f(x,y) = 0.5 + \frac{\cos^2(\sin(x^2-y^2))-0.5}{1+0.001(x^2+y^2)^2} \tag{17}$$

$$f(\mathbf{x}) = f(x_1,\dots,x_n) = \frac{1}{2}\sum_{i=1}^{n}\left(x_i^4-16x_i^2+5x_i\right) \tag{18}$$

$$f(\mathbf{x}) = f(x_1,\dots,x_n) = \sum_{i=1}^{n}\left(x_i^2+10\cdot(1-\cos(2\pi x_i))\right) \tag{20}$$

$$f(\mathbf{x}) = f(x_1,\dots,x_n) = \sum_{i=1}^{n}|x_i| \tag{21}$$

$$f(\mathbf{x}) = f(x_1,\dots,x_n) = \sum_{i=1}^{n}\sqrt{|x_i|} \tag{22}$$

$$f(\mathbf{x}) = f(x_1,\dots,x_n) = \sum_{i=1}^{n}|x_i| + \left(10+x_i^2\right)\cdot(1-\cos(2\pi x_i)) \tag{24}$$

$$f(\mathbf{x}) = f(x_1,\dots,x_n) = \sum_{i=1}^{n} 1-\operatorname{sink}(x_i) \quad \text{with } \operatorname{sink}(x) = \frac{\sin(x)}{x} \tag{25}$$

Fig. 2.

Fig. 2 List of objective functions