Gradient Guided Differential Evolution-Simulated Annealing Hybrid for Differentiable Multivariate Functions

A New Hybrid Metaheuristic

Bhumrapee Soonjun

November 27, 2023

Contents

Introduction

1. We present a new hybrid metaheuristic between Differential Evolution and Simulated Annealing that uses gradient methods
2. The target functions are differentiable
3. Gradient methods are used to exploit this extra information to provide faster convergence and improve solution quality

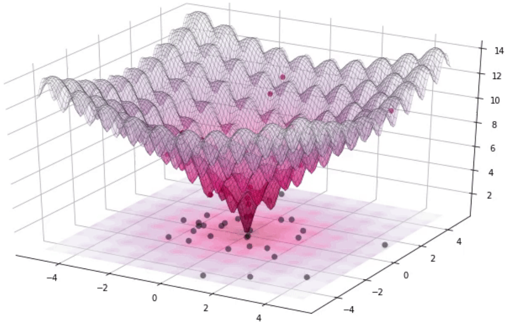


Figure: Ackley's Function

Introduction - Differential Evolution

- Differential Evolution (DE) is a global optimization method that belongs to the evolutionary class metaheuristics like Genetic Algorithm.
- Each individual in the population is a vector of real numbers rather than some high-level encoding, which makes DE very efficient because operations are allowed to work on primitive types like double and array.
- The algorithm is also very simple and contains only two main steps for offspring creation, which is
  1. Mutation
  2. Crossover
- Very easily parallelizable as well
- However, can get stuck at a local minimum and face premature convergence

Introduction - Simulated Annealing

- Simulated Annealing (SA) is considered to be a local search algorithm that is inspired by the metal annealing process
- It is like a modified hillclimbing where the algorithm is allowed to make bad jump sometime
- Good for searching for local neighborhoods because it can jump around
- Bad for global optimization because the search space can be huge

Introduction - The Best of Both Worlds

The idea is to combine the two algorithms to obtain both global optimization properties from Differential Evolution and local hillclimbing form Simulated Annealing to jump out of minima to prevent premature convergence.

- Differential Evolution as the master program, and simulated annealing acceptance criteria for local jump
- Help improve diversity as well as prevent premature convergence
- Include gradient method to provide quicker convergence initially
- The Gradient method used in the experiment is L-BFGS, but other methods can also be useful such as SGD for further minimum jumping

Storn and Price (1997): Differential Evolution

Algorithm 1: Differential Evolution

```
input  : $f$, $F$, $p_c$, $NP$, $N$
output: $B$: Best vector
$P^0$ = Initialize Population;
for $t$ from 1 up to $N$ do
    $P^{t+1} = P^t$;
    for $j$ from 1 up to NP do
        $u$ = Mutate($P^t$, $j$);
        $u$ = Crossover($u$, $P_j^t$, $p_c$);
        if $f(u) < f(P_j^t)$ then
            $P_j^{t+1} = u$
        end
    end
end
```

DE - Mutation

Algorithm 2: Mutate

```
input  : $P_t$: Population, $j$
output: $u$: Trial Vector
$j, k, l =$ randint$(0, |P|, 3)$, s.t $i \neq j \neq k \neq l$;
$u = P_j^t + F(P_k^t - P_l^t)$
```

Algorithm 3: Crossover

```
input  : $u$, $x$, $p_c$
output: $u$: Trial Vector
$inds =$ rand$(0, D) < p_c$;
$i =$ rand();
$u[inds] = x[inds]$;
$u[i] = x[i]$
```

DE - Mutation Intuition

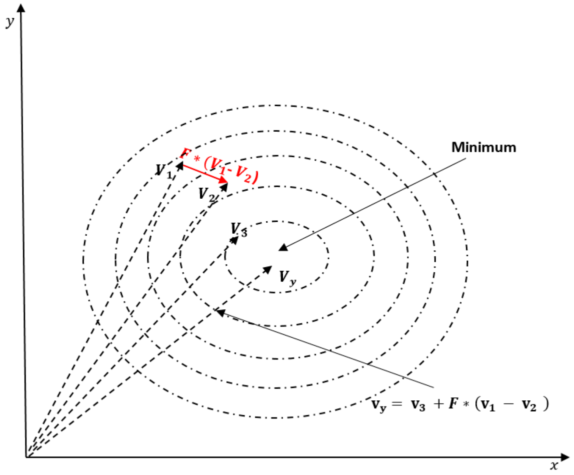

y
x
Minimum
V_1
V_2
V_3
V_y
$F * (V_1 - V_2)$
$\mathbf{v_y} = \mathbf{v_3} + F * (\mathbf{v_1} - \mathbf{v_2})$

Figure: Mutation Process

DE- Other Mutation Strategies

To generalize the mutation strategy, the author adopted the following notation,

$$DE/x/y/z$$

where x is the base vector choosing criteria, y is the number of vectors to use for differencing, and z is the crossover criteria. With this, the above mutation scheme can be written as

$$DE/rand/1/bin$$

However, the author also suggested other strategies such as

$$DE/best/2/bin$$

which translates to

$$u = P_i + F(P_j - P_k + P_l - P_m)$$

where $i = \arg\min(f(P_0), f(P_1), ..., f(P_{NP}))$

Quasi Newton's Method

Newton's method is an iterative algorithm to find the root of a function.

$$x_{t+1} = x_t - \frac{f(x_t)}{f'(x_t)}$$

For optimization purposes, one can also derive the following update rule for minimizing a function by doing Taylor expansion of $f(x_t + h)$ as follows,

$$f(x_t + h) \approx f(x_t) + f'(x_t)h + \frac{1}{2}f''(x_t)h^2$$

This is just a quadratic polynomial, minimizing w.r.t h yields

$$h = \frac{f'(x_t)}{f''(x_t)}$$

Plugging back in yields

$$x_{t+1} = x_t - \frac{f'(x_t)}{f''(x_t)}$$

Quasi Newton's Method

For higher dimension, this generalizes to

$$\begin{aligned}
f(\mathbf{x}_t+\mathbf{h}) &\approx f(\mathbf{x_t}) + \nabla f(\mathbf{x}_t)\mathbf{h}^T + \frac{1}{2}\mathbf{h}\nabla^2 f(\mathbf{x}_t)\mathbf{h}^T \\
\mathbf{0} &= \frac{d}{d\mathbf{h}}\left(f(\mathbf{x}_t) + \nabla f(\mathbf{x}_t)\mathbf{h}^T + \frac{1}{2}\mathbf{h}\nabla^2 f(\mathbf{x}_t)\mathbf{h}^T\right) \\
&= \nabla f(\mathbf{x}_t) + \frac{1}{2}\mathbf{h}\left(\nabla^2 f(\mathbf{x}_t) + (\nabla^2 f(\mathbf{x}_t))^T\right) \\
\mathbf{0} &= \nabla f(\mathbf{x_t}) + \mathbf{h}\nabla^2 f(\mathbf{x}_t) \\
\mathbf{h} &= -(\nabla^2 f(\mathbf{x}_t))^{-1}\nabla f(\mathbf{x}_t)
\end{aligned}$$

So,

$$\begin{aligned}
\mathbf{x}_{t+1} &= \mathbf{x}_t - (\nabla^2 f(\mathbf{x_t}))^{-1}\nabla f(\mathbf{x}_t) \\
&= \mathbf{x}_t - \eta H_f^{-1}\nabla f(\mathbf{x}_t)
\end{aligned}$$

where $0 < \eta < 1$ is the step size.

Quasi Newton's Method

Calculating the Hessian takes $O(d^2)$ where d is the dimension of f. For expensive functions, this is very costly. Furthermore, matrix inversion takes $O(d^3)$ time.

Because of this, people developed the Quasi-Newton method which approximates the Hessian using the Jacobian. The following are some well-known Quasi-Newton methods.

1. BFGS: Broyden–Fletcher–Goldfarb–Shanno
2. L-BFGS: Limited Broyden–Fletcher–Goldfarb–Shanno
3. Broyden
4. DFP: Davidon–Fletcher–Powell

This takes $O(d)$ function evaluations, and $O(d^2)$ time which is much better. We will use these to improve the solution.

Gradient Descent

As the name suggests, the gradient descent algorithm descent down the gradient of a function with the following update rule,

$$\mathbf{x_{t+1}} = x_t - \eta \nabla f(\mathbf{x_t})$$

Very simple and efficient. Has $O(d)$ number of function evaluations and also running time. Has a close variant also known as Coordinate Descent where we iteratively make one in one direction at a time. The update rule per big iteration is

$$x_{t+1} = x_t + \eta \nabla_i f(\mathbf{x_t})$$

where i is the permutation index. Similar computational complexity.

Convergence Rates

Newton's Method has a quadratic convergence rate or $O(h^2)$. However, this is very expensive. BFGS on the other hand has superlinear convergence if the function is strongly convex and is Lipschitz continuous. Let k, μ, L, be iteration counter, strong convexity parameter, and Lipshitz constant respectively, BFGS has the following convergence rate Rodomanov and Nesterov (2022),

$$\left(\frac{nL}{\mu k}\right)^{\frac{k}{2}}$$

Gradient descent has the following convergence if f is convex with Lipschitz gradient

$$O\left(\frac{1}{h}\right)$$

Although our function might not necessarily be convex or have a Lipschitz gradient, we can still apply these methods. Moreover, we can hope that in some neighborhoods the function behaves reasonably like a convex function.

Rafajłowicz (2015): A Hybrid Differential Evolution-Gradient Optimization Method

The author experimented with the idea of using Newton's Method to help improve the convergence speed of DE. In addition, they also presented a way to dynamically modify F - the differential factor - as a function of iteration. The results showed that using Newton's method can improve the speed with nearly the same computational burden; however, the test function used in that experiment has only two variables, and on modern computers this is negligible.

Because we want to choose a good method with fast convergence and minimum computation complexity, we will go with the BFGS method because evaluating a function is much more expensive than computing a $n \times n$ matrix. Moreover, we are getting faster convergence than gradient descent or coordinate descent as well.

Roadmap

Here's the high-level overview of the whole system before we dive into each technical part.

1. DE will be used as the master program which means mutation strategies and the main loop is borrowed from DE
2. DE's population update will be modified to be immediate
3. The population will be divided into two parts where one is strictly elitist and the other is semi
4. The latter part of the population acceptance criteria will be Boltzmann's criteria
5. Gradient method is used as another mutation operator to generate an offspring

Notations

We will work with the following notations.

1. $f(x)$: Function to be minimized
2. N: Number of generations
3. P: Population
4. P_i: Individual ith
5. u_i: Trial vector ith
6. $|P|/NP$: Population size
7. F: Differential factor
8. p_c: Crossover probability
9. $GM(x)$: Gradient method
10. p_g: Gradient probability
11. T: Temperature
12. α: Cooling rate
13. β: α update rate
14. E_r: Elitism ratio

Population Data Structure

To cater to elitism while also maintaining a certain level of freedom of search space exploration, we will maintain a single population that is divided into two parts depending on the elitism ratio E_r.

The first part of the population, $P[:\lceil E_r \cdot |P| \rceil]$, will be selected in an elitist fashion. This means if u is $f(u_i) > f(P_i)$ then we immediately reject the trial solution. This is normal DE.

The second part acceptance criteria is Boltzmann's, let $\Delta = f(u_i) - f(P_i)$

$$p = \begin{cases} 1, & \text{if } \Delta < 0, \\ \exp(-\Delta/T), & \text{otherwise} \end{cases}$$

In other words, this is just SA acceptance criteria. This is the part that allows our algorithm to jump off local minima.

Offspring Creation

There are two methods for creating an offspring.

1. Gradient Method: Apply L-BFGS, a variant of BFGS that is memory efficient, to produce a new vector
2. Mutation and Crossover: $DE/rand/1/bin$, we use this instead of $DE/best/1/bin$ to prevent premature convergence. Although, it possible is to use a combination of both to provide faster convergence and also maintain diversity.

The algorithm

Algorithm 4: Gradient DE-SA Hybrid

```
input  : $f$, $N$, $F$, $p_c$, $p_g$, $GM$, $E_r$, $T$, $\alpha$, $\beta$
output: $P_n$: Last Generation Population
$P$ = Initialize Population;
for $i$ from 1 up to $N$ do
    if $i$ mod $\beta$ == 0 then
        $T = T \cdot \alpha$
    end
    for $j$ from 1 up to $|P|$ do
        if Unif$(0, 1) < p_c$ then
            $u_i = GM(P_i)$;
        else
            $u_i$ = Mutate($u_i$, $F$, $j$);
            $u_i$ = Crossover($u_i$, $p_c$);
        end
    end
    $\Delta = f(u_i) - f(P_i)$;
    if $\Delta < 0$ then
        $P_i = u_i$;
    else if Unif(0, 1) $< \exp(-\Delta/T)$ and $i < E_r \cdot |P|$ then
        $P_i = u_i$;
    end
end
```

Experiment

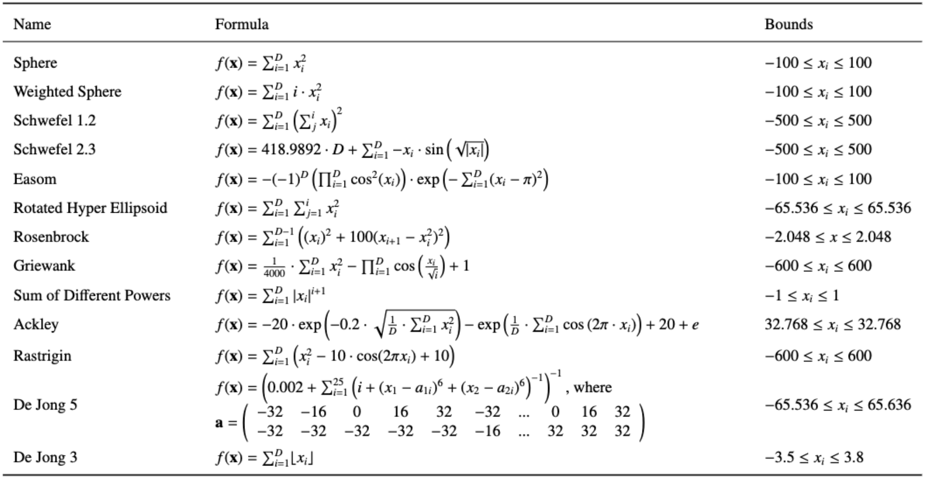
DE-SA is tested against other heuristics including Gradient Evolution, Differential Evolution, Dual Annealing, Artificial Bee Colony, and Particle Swarm Optimization.

1. Gradient Evolution and Differential Evolution is implemented in python
2. Dual Annealing is from SciPy's optimization package
3. Artificial Bee Colony is from Python's Bee Colony package
4. Particle Swarm Optimization is from Pymoo's package

The test machine has the following specifications.

1. Macbook M2 Pro Max
2. 12 CPU cores
3. 30 GPU cores
4. 32 GB Ram

Test Functions



Name	Formula	Bounds
Sphere	$f(\mathbf{x}) = \sum_{i=1}^{D} x_i^2$	$-100 \leq x_i \leq 100$
Weighted Sphere	$f(\mathbf{x}) = \sum_{i=1}^{D} i \cdot x_i^2$	$-100 \leq x_i \leq 100$
Schwefel 1.2	$f(\mathbf{x}) = \sum_{i=1}^{D} \left(\sum_{j}^{i} x_i\right)^2$	$-500 \leq x_i \leq 500$
Schwefel 2.3	$f(\mathbf{x}) = 418.9892 \cdot D + \sum_{i=1}^{D} -x_i \cdot \sin\left(\sqrt{\lvert x_i \rvert}\right)$	$-500 \leq x_i \leq 500$
Easom	$f(\mathbf{x}) = -(-1)^D \left(\prod_{i=1}^{D} \cos^2(x_i)\right) \cdot \exp\left(-\sum_{i=1}^{D}(x_i - \pi)^2\right)$	$-100 \leq x_i \leq 100$
Rotated Hyper Ellipsoid	$f(\mathbf{x}) = \sum_{i=1}^{D} \sum_{j=1}^{i} x_i^2$	$-65.536 \leq x_i \leq 65.536$
Rosenbrock	$f(\mathbf{x}) = \sum_{i=1}^{D-1} \left((x_i)^2 + 100(x_{i+1} - x_i^2)^2\right)$	$-2.048 \leq x \leq 2.048$
Griewank	$f(\mathbf{x}) = \frac{1}{4000} \cdot \sum_{i=1}^{D} x_i^2 - \prod_{i=1}^{D} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	$-600 \leq x_i \leq 600$
Sum of Different Powers	$f(\mathbf{x}) = \sum_{i=1}^{D} \lvert x_i \rvert^{i+1}$	$-1 \leq x_i \leq 1$
Ackley	$f(\mathbf{x}) = -20 \cdot \exp\left(-0.2 \cdot \sqrt{\frac{1}{D} \cdot \sum_{i=1}^{D} x_i^2}\right) - \exp\left(\frac{1}{D} \cdot \sum_{i=1}^{D} \cos(2\pi \cdot x_i)\right) + 20 + e$	$32.768 \leq x_i \leq 32.768$
Rastrigin	$f(\mathbf{x}) = \sum_{i=1}^{D} \left(x_i^2 - 10 \cdot \cos(2\pi x_i) + 10\right)$	$-600 \leq x_i \leq 600$
De Jong 5	$f(\mathbf{x}) = \left(0.002 + \sum_{i=1}^{25} \left(i + (x_1 - a_{1i})^6 + (x_2 - a_{2i})^6\right)^{-1}\right)^{-1}$, where $\mathbf{a} = \begin{pmatrix} -32 & -16 & 0 & 16 & 32 & -32 & \ldots & 0 & 16 & 32 \\ -32 & -32 & -32 & -32 & -32 & -16 & \ldots & 32 & 32 & 32 \end{pmatrix}$	$-65.536 \leq x_i \leq 65.636$
De Jong 3	$f(\mathbf{x}) = \sum_{i=1}^{D} \lfloor x_i \rfloor$	$-3.5 \leq x_i \leq 3.8$

Figure: Test Functions

Test Functions Characteristics

- Convex
  1. Sphere
  2. Weighted Sphere
  3. Schwefel 1.2
  4. Rotated Hyper Ellipsoid
  5. Sum of Different Powers
- Unimodal Steep Hole
  1. Easom
  2. De Jong 5
- Unimodal Flat Steps
  1. De Jong 3
- Unimodal Non Convex Narrow
  1. Rosenbrock
- Multimodal Functions include
  1. Griewank
  2. Ackley
  3. Rastrigin
  4. Schwefel 2.3

Parameter Tuning

Table: Parameters Tuning for DE-SA

	p_g	F	p_c	T	α	β	E_r
Sphere	0.01	0.3	0.05	0	0	0	1
Weighted Sphere	0.00	0.04	0.1	0	0	0	1
Schwefel 1.2	0.01	0.5	0.1	10	0.9	5	0.6
Schwefel 2.3	0.05	1	0.1	100	0.9	10	0.5
Easom	0.0	0.5	0.1	0	0	0	1
Rotated Hyper Ellipsoid	0.01	0.5	0.1	0	0	0	1
Rosenbrock	0.1	0.5	0.01	100	0.95	5	0.5

Parameter Tuning

Table: Parameters Tuning for DE-SA

	p_g	F	p_c	T	α	β	E_r
Griewank	0.01	0.5	0.01	100	0.95	10	0.5
Power Sum	0.001	0.5	0.1	0	0	0	1
Ackley	0.01	0.5	0.1	1000	0.95	10	0.8
Rastrigin	0.01	0.5	0	100	0.95	10	0.8
De Jong 5	0	0.5	0	100	0.95	10	0.5
De Jong 3	0	0.5	0.1	0	0	0	1

Gradient Method Application

1. Instead of applying it directly to the old vector, we can apply it after crossover.
2. Keep track of individual that has been recently updated and defer the gradient application out if it's recently updated
3. Moreover, instead of setting a maximum number of iterations for the gradient method, we can let it run to termination by reaching a minimum or precision loss instead
4. We can encapsulate all of these as a table and instead of setting a cooldown period by hand, we can make it the expectation of p_g as well.

Mutation Strategies

1. To yield faster convergence and close the gap in the population, we can use $DE/best/1/bin$ instead
2. However, as we want to maintain diversity we want to use a combination of both for example
   - 50%: $DE/rand/1/bin$
   - 50%: $DE/best/1/bin$

Cooling Schedule

Another improvement that can be made is to borrow the cooling schedule of the Generalised Simulated Annealing where T is allowed to be decreased like an inverse of time. For Generalised Simulated Annealing, this works because of the visiting distribution that makes frequent local jumps but occasionally long jumps like the Cauchy-Lorentz Distribution. The generalized acceptance criteria reads

$$P(E) = \left\{ \begin{array}{ll} 1, & \text{if } \Delta < 0 \\ \left[(1 + (q_A - 1)(\Delta)/T)^{1/(q_A - 1)}\right]^{-1}, & \text{otherwise} \end{array} \right\}$$

and T is updated as

$$T = T_0 \cdot \frac{\exp((q_V - 1) \cdot \ln 2) - 1}{\exp((q_v - 1) \cdot (t + 2)) - 1}$$

For our case, since DE is already making long jumps and able to search locally, we can allow T to decrease as the above to get faster convergence. Furthermore, with this, we can also make F a function of T such that F decreases over time so that it searches more locally.

Thanks to Nattakorn Massaya-anon (NCSU) and Peeranat Tanasatitchai (CU) for the literature that I got paywalled.

Frame Title

1. Virtanen et al. (2020)
2. Oliveira (2021)
3. Atangana (2018)
4. Li and Fukushima (2001)
5. Dai (2002)
6. Storn and Price (1997)
7. Eltaeib and Mahmood (2018)
8. Rafajłowicz (2015)
9. Kuo and Zulvia (2015)
10. Molga and Smutnicki (2005)
11. Ponsich and Coello Coello (2013)
12. Shi and Eberhart (1999)
13. Blank and Deb (2020)
14. Rafati and Marcia (2019)
15. Rodomanov and Nesterov (2022)
16. Odeniyi et al. (2015)
17. Surjanovic and Bingham
18. Tibshirani (2013)
19. Tsallis and Stariolo (1996)
20. Virk (2023)

Atangana, A., 2018. Chapter 2 - principle of groundwater flow, in: Atangana, A. (Ed.), Fractional Operators with Constant and Variable Order with Application to Geo-Hydrology. Academic Press, pp. 15–47. URL: https://www.sciencedirect.com/science/article/pii/B9780128096703000023, doi:https://doi.org/10.1016/B978-0-12-809670-3.00002-3.

Blank, J., Deb, K., 2020. pymoo: Multi-objective optimization in python. IEEE Access 8, 89497–89509.

Dai, Y.H., 2002. Convergence properties of the bfgs algoritm. SIAM Journal on Optimization 13, 693–701. doi:10.1137/S1052623401383455.

Eltaeib, T., Mahmood, A., 2018. Differential evolution: A survey and analysis. Applied Sciences 8, 1945. URL: http://dx.doi.org/10.3390/app8101945, doi:10.3390/app8101945.

Kuo, R., Zulvia, F.E., 2015. The gradient evolution algorithm: A new metaheuristic. Information Sciences 316, 246–265. URL: https://www.sciencedirect.com/science/article/pii/S0020025515002996, doi:https://doi.org/10.1016/j.ins.2015.04.031. nature-Inspired Algorithms for Large Scale Global Optimization.

Li, D.H., Fukushima, M., 2001. On the global convergence of the bfgs method for nonconvex unconstrained optimization problems. SIAM Journal on

Optimization 11, 1054–1064. URL: https://doi.org/10.1137/S1052623499354242, doi:10.1137/S1052623499354242, arXiv:https://doi.org/10.1137/S1052623499354242.

Molga, M., Smutnicki, C., 2005. Test functions for optimization needs. Test functions for optimization needs 101, 48.

Odeniyi, O., Omidiora, O., Olabiyisi, O.S., Aluko, A., 2015. Development of a modified simulated annealing to school timetabling problem. International Journal of Applied Information Systems 8, 16–24. URL: https://api.semanticscholar.org/CorpusID:16419715.

Oliveira, S.C.P., 2021. beecolpy. - -, –. doi:–.

Ponsich, A., Coello Coello, C.A., 2013. A hybrid differential evolution—tabu search algorithm for the solution of job-shop scheduling problems. Applied Soft Computing 13, 462–474. URL: https://www.sciencedirect.com/science/article/pii/S1568494612004188, doi:https://doi.org/10.1016/j.asoc.2012.07.034.

Rafajłowicz, W., 2015. A hybrid differential evolution-gradient optimization method, in: Rutkowski, L., Korytkowski, M., Scherer, R., Tadeusiewicz, R., Zadeh, L.A., Zurada, J.M. (Eds.), Artificial Intelligence and Soft Computing, Springer International Publishing, Cham. pp. 379–388.

Rafati, J., Marcia, R.F., 2019. Deep reinforcement learning via l-bfgs optimization. arXiv:1811.02693.

Rodomanov, A., Nesterov, Y., 2022. Rates of superlinear convergence for classical quasi-newton methods. Mathematical Programming 194, 159–190. URL: https://doi.org/10.1007/s10107-021-01622-5, doi:10.1007/s10107-021-01622-5.

Shi, Y., Eberhart, R., 1999. Empirical study of particle swarm optimization, in: Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406), pp. 1945–1950 Vol. 3. doi:10.1109/CEC.1999.785511.

Storn, R., Price, K., 1997. Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces. Journal of Global Optimization 11, 341–359. doi:10.1023/A:1008202821328.

Surjanovic, S., Bingham, D., . Virtual library of simulation experiments: Test functions and datasets. Retrieved November 24, 2023, from http://www.sfu.ca/~ssurjano.

Tibshirani, R., 2013. Gradient descent: Convergence analysis.

Tsallis, C., Stariolo, D.A., 1996. Generalized simulated annealing. Physica A: Statistical Mechanics and its Applications 233, 395–406. URL: https://www.sciencedirect.com/science/article/pii/S0378437196002713, doi:https://doi.org/10.1016/S0378-4371(96)00271-3.

Virk, Z., 2023. Gradient descent and quasi-newton methods.

Virtanen, P., Gommers, R., Oliphant, T.E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S.J., Brett, M., Wilson, J., Millman, K.J., Mayorov, N., Nelson, A.R.J., Jones, E., Kern, R., Larson, E., Carey, C.J., Polat, İ., Feng, Y., Moore, E.W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E.A., Harris, C.R., Archibald, A.M., Ribeiro, A.H., Pedregosa, F., van Mulbregt, P., SciPy 1.0 Contributors, 2020. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. Nature Methods 17, 261–272. doi:10.1038/s41592-019-0686-2.