A unified framework for the application and evaluation of different methods for neural parameter optimization

Máté Mohácsi
Institute of Experimental Medicine
Faculty of Information Technology and Bionics, Pázmány Péter Catholic University
Budapest, Hungary
mohacsi.mate@koki.mta.hu

Márk Patrik Török
Institute of Experimental Medicine
Faculty of Information Technology and Bionics, Pázmány Péter Catholic University
Budapest, Hungary
torok.mark.patrik@gmail.com

Sára Sáray
Institute of Experimental Medicine
Faculty of Information Technology and Bionics, Pázmány Péter Catholic University
Budapest, Hungary
saray.sara@koki.mta.hu

Szabolcs Káli
Institute of Experimental Medicine
Budapest, Hungary
kali@koki.hu

Abstract— Automated parameter search has become a standard method in the modeling of neural systems. These studies could potentially take advantage of recent developments in nonlinear optimization, and the availability of software packages containing high-quality implementations of algorithms that proved useful in other domains. However, a systematic comparison of the available algorithms for problems that are typical in neuroscience has not been performed.

We developed a software tool for fitting the parameters of neural models, which provides intuitive, uniform access to a variety of state-of-the-art optimization algorithms implemented by four different Python packages. We also established a set of benchmark problems of different complexity that involve a variety of widely used neuronal models. We then used our optimization tool to systematically evaluate the performance of the algorithms on our set of benchmark problems.

We found that several evolutionary and related algorithms consistently provided good solutions for all of our benchmarks. However, the relative performance of the different methods, both in terms of the quality of the final result and in terms of convergence speed, depended substantially on the nature of the problem. We hope that our software tool and benchmarking results will facilitate the choice and application of the best parameter-fitting methods in neuroscientific research.

Keywords—neuronal modeling, Python, software, simulation, model fitting, parameter optimization, algorithm

I. Introduction

The detailed modeling of neurons is becoming an increasingly widespread method in neurobiological research. Currently available experimental data make it possible to create complex multicompartmental conductance-based neuron models. In principle, such models can approximate the behavior of real neurons very well. However, these models have many parameters and some of these parameters cannot be directly determined in experiments. Therefore, a common approach is to tune parameter values to bring the model's behavior as close as possible to the experimental data. Several software solutions implementing various nonlinear optimization methods have been developed recently, which allow the determination of the unknown parameters of the neural models in a systematic way. However, only a few of these packages and algorithms have been evaluated on the problems that are typical in neuroscience [1]-[4]. Our aim in this study was (1) to create a software tool that provides uniform access to a large variety of different optimization algorithms; (2) to develop a set of benchmark problems for neural parameter tuning; and (3) to systematically evaluate and compare the various algorithms and implementations using our software and benchmarking suite.

II. Implementation

Our starting point was our previously developed software (Optimizer; https://github.com/KaliLab/optimizer), which was already shown to be a useful tool for neuronal optimization [4]. In Optimizer, model evaluations can be performed either by the NEURON simulator [5] (handled internally) or any external (black-box) simulator. All functionalities can be accessed from the graphical user interface; there is also a command line interface for batch processing. The modular, object-oriented structure of the program makes it possible to add new error functions and optimization algorithms.

We now created an updated and enhanced version of this software. The new version was developed in Python 3 to support recent open-source Python modules, such as search algorithms, graphical and parallelization interfaces. The repertoire of algorithms was extended by several new methods that proved effective in other studies. For many of these search algorithms, parallel optimization is also supported and easily configurable.

A wide variety of features (including those in the eFEL package) [6] can be used to evaluate the error of the optimization; multiple, weighted features are also supported.

Our optimization tool supports optimization algorithms implemented by four separate Python packages: Inspyred [7], Pygmo [8], BluePyOpt [9], and Scipy. [10] The following algorithms are currently included:

Classical Evolutionary Algorithm (CEO) – also known as a genetic algorithm, which is inspired by natural evolution. The process of the algorithm consists of selection, crossover, and mutation. The best individuals are selected for reproduction in order to produce offspring for the next generation. [11]

Particle Swarm Optimization (PSO) – This algorithm represents candidate solutions as particles moving around in the search space, influenced by the current best known positions for the individual particles and that of the entire group. [12]

Simulated Annealing (SA) – This method is based on exploring the search space in random steps, where the probability of accepting a worse solution in each step decreases as the search progresses. [13]

Differential Evolution (DE) - An evolutionary algorithm with a recombination approach, that involves the creation of offspring based on the weighted difference between two randomly selected individuals. [14]

Non-dominated Sorting Genetic Algorithm (NSGA) - An evolutionary multi-objective algorithm, where the population is sorted in each step based on the ordering of the Pareto dominance of the individuals. [15]

Pareto Archived Evolution Strategy (PAES) - An evolutionary multi-objective algorithm, using local search from a population and using a reference archive of previously found solutions to approximate the dominance ranking of current vectors. [16]

Basinhopping (BH) – This is a generalization of the Simulated Annealing algorithm, which uses local optimization in each step. [17]

Nelder-Mead - A simplex-based direct search method to find a local minimum of the cost function. [18]

L-BFGS-B - Finds the local minimum, by using objective function values and its gradient, using a limited amount of computer memory. [19]

Self-Adaptive Differential Evolution (SADE) – A version of Differential Evolution, which adjusts the mutation rate and the crossover rate adaptively. [20]

Covariance Matrix Adaptation Evolutionary Strategy (CMAES) – An evolutionary algorithm which samples candidate solutions from multivariate normal distributions with adapting mean and covariance matrix. [21]

Exponential Natural Evolution Strategies (XNES) – An evolutionary algorithm that uses the natural gradient to update the search distribution. [22]

Simple Genetic Algorithm (SGA) - Classical evolutionary algorithm consisting of selection, crossover, and mutation.

Bee Colony - Artificial Bee Colony Optimization (ABC) is a swarm-based algorithm like Particle Swarm Optimization, which simulates the behaviors of honeybee. [23]

Differential Evolution 1220 - Differential Evolution that adds self-adaptation for the mutation variant.

Indicator Based Evolutionary Algorithm (IBEA) - Multi-objective Evolutionary Algorithm that computes the fitness value based on predefined binary indicators. [24]

Random Search (RAND) – This is our baseline method, which samples solutions from the search space based on the uniform probability distribution. Optimizer uses our own implementation of this method. [25]

Algorithms integrated into our software from the Inspyred package are Classical Evolutionary Algorithm (CEO), ParticleSwarm Optimization (PSO), Simulated Annealing (SA), Differential Evolution (DE), Non-dominated Sorting Genetic Algorithm (NSGA), and Pareto Archived Evolution Strategy (PAES).

Pygmo package algorithms are Differential Evolution (DE), Self-Adaptive Differential Evolution (SADE), Covariance Matrix Adaptation Evolutionary Strategy (CMAES), Particle Swarm Optimization (PSO), Exponential Evolution Strategies (XNES), Simple Genetic Algorithm (SGA), Bee Colony, and Differential Evolution 1220.

Scipy package algorithms are Basinhopping (BH), Nelder-Mead, and L-BFGS-B.

BluePyOpt package algorithms are Indicator Based Evolutionary Algorithm (IBEA), and Non-dominated Sorting Genetic Algorithm (NSGA-II).

In the current study, we evaluated a large subset of these algorithms, including several of the most widely used single-objective and multi-objective methods.

III. Use cases

The neuronal optimization problems that we included among our benchmarks differ in complexity, model type, simulation protocol, fitness functions, and the number of unknown parameters. Some of our benchmarks (Hodgkin-Huxley and Voltage Clamp, see below) use surrogate data as the target. In this case, target data are generated by a neuronal model with known parameters; some of these parameters are then considered to be unknown, and the task is to reconstruct the correct values. Therefore, in these test cases, a perfect solution with zero error is known to exist, and the corresponding parameters can be compared to those found by the search algorithms. However, in most of our benchmarks, the target data were recorded in physiological experiments, or were generated by more complex models than the one we were fitting. In these cases, the best-fitting parameters and the minimal possible error score are unknown.

A. Hodgkin-Huxley

This use case is based on a single-compartment model, which contains conductances from the original Hodgkin-Huxley model [26], and is implemented in NEURON. To generate the target voltage trace, a step current is injected into the neuron (amplitude = 200 pA, delay = 200 ms, duration = 500 ms, and the voltage trace duration is 1000 ms). The test case involves recovering the correct conductance densities (Na+, K+, leak – 3 parameters). A combination of four features (spike count, spike amplitude, spike width, mean squared error of voltage excluding spikes) was used to compare each simulated trace to the original (target) trace (Fig. 1).

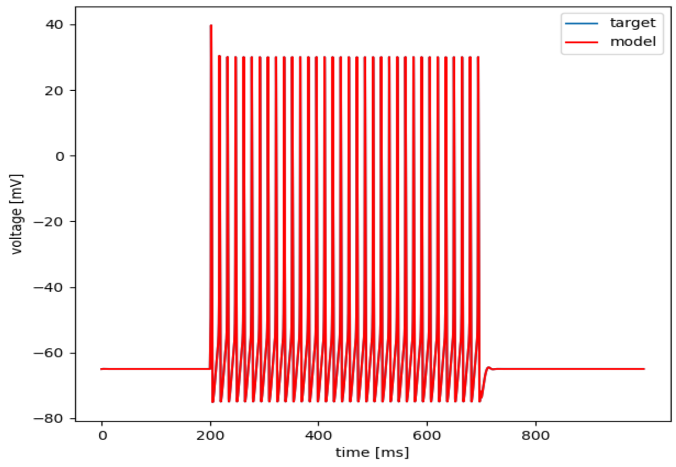

Fig. 1. Target and best fitting traces for a Hodgkin-Huxley model.

B. Voltage Clamp

Another benchmark involves recovering synaptic parameters (weight, rise and decay time constants, delay – 4 parameters) from simulated voltage clamp recordings during synaptic stimulation in a single-compartment model, using the mean squared error fitness function (Fig. 2).

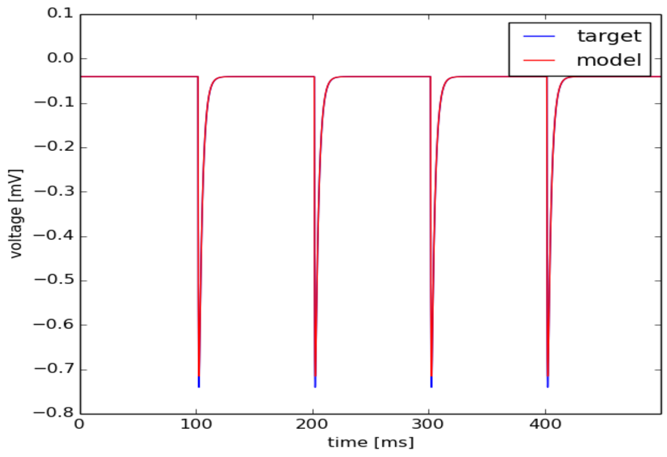

Fig. 2. Target and best fitting traces for a simulated voltage-clamp experiment.

C. Passive, anatomically detailed neuron

This benchmark uses a morphologically detailed passive model of a hippocampal CA1 pyramidal cell [27]. A short (3 ms, 500 pA) and a long (600 ms, 10 pA) current pulse (separated by 300 ms) were injected into the soma, and the membrane potential was also recorded there. The task involves fitting 3 passive parameters (capacitance, leak conductance, axial resistance) to experimental data recorded using the same complex current clamp stimulus (Fig. 3). Traces are compared via the mean squared error fitness function.

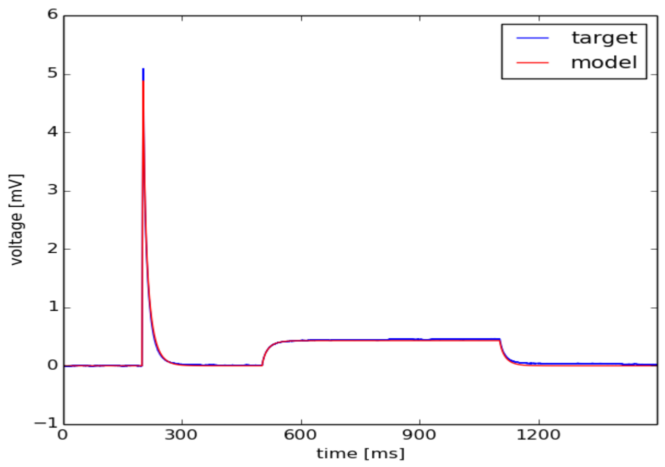

Fig. 3. Target and best fitting traces for a passive, anatomically detailed neuron.

D. Simplified model

This use case attempts to fit a six-compartmental simplification of a biophysically accurate and morphologically detailed hippocampal CA1 pyramidal cell model to voltage responses of the original full model. We injected a 200 pA step current stimulus into the soma. Stimulus starts at 200 ms and lasts for 600 ms, with 1000 ms recording duration as shown in Fig. 4. The fit was evaluated via a combination of features including mean squared error (excluding spikes), spike count, latency to first spike, action potential amplitude, action potential width, and after-hyperpolarization depth. The model contained 9 tunable parameters.

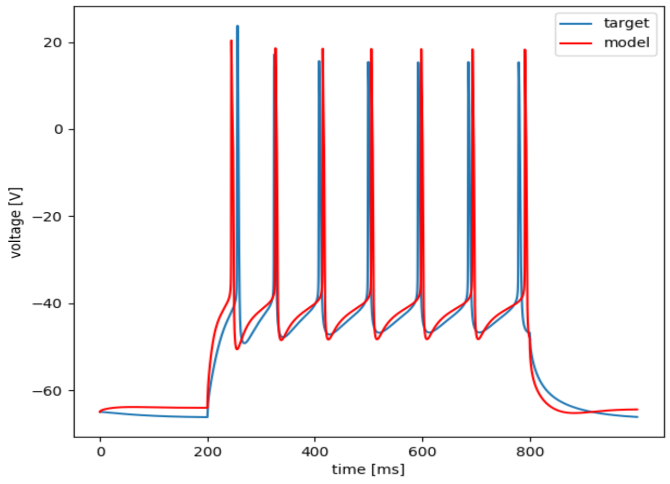

Fig. 4. Target and best fitting traces for a simplified (6-compartment) model.

E. *Extended integrate-and-fire model*

In this benchmark, an abstract (adaptive exponential integrate-and-fire) spiking model was fitted to four different voltage traces of a real neuron (hippocampal CA3 pyramidal cell), with 0.30, 0.35, 0.40, and 0.45 nA current stimulation amplitudes (Fig. 5). Sampling frequency was 5 kHz with 1100 ms recording duration. The unknown optimizable parameters are capacitance, leak conductance, leak reversal potential, threshold voltage, reset voltage, refractory period, steepness of exponential part of the current-voltage relation, subthreshold adaptation conductance, spike adaptation current, adaptation time constant (10 parameters). Fitness functions were mean squared error, spike count, and latency to first spike.

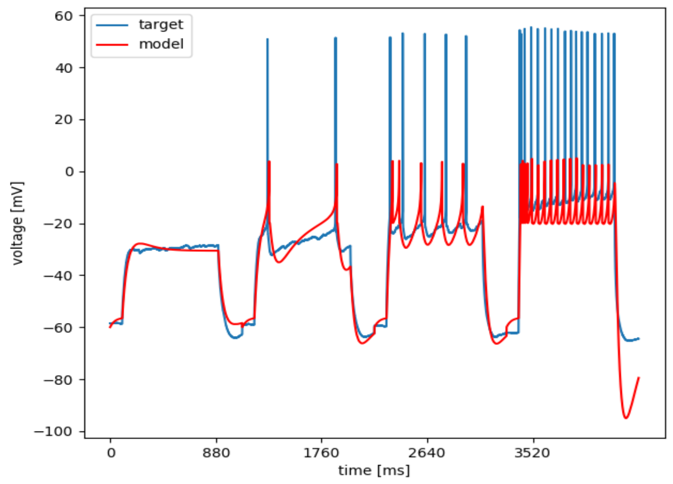

Fig. 5. Target and best fitting traces for an extended integrate-and-fire model. The four traces are displayed in a concatenated form. Note that the apparent difference in spike amplitude is irrelevant for the IF model.

F. CA1 pyramidal cell

This was the most complex test case based on an anatomically and biophysically detailed hippocampal CA1 pyramidal cell model built in our research group. Multiple attributes determined by experiments, including the biophysics, ion channel distribution, and electrophysiological characteristics were used in designing the model. The task was to optimize the 16 channel density and kinetic parameters of the model. The stimulus amplitudes were -0.25, 0.05, 0.1, 0.15, 0.2 0.25 nA, respectively. Somatic subthreshold and spiking features extracted by eFEL from the model's voltage response were compared to same features extracted from experimental measurements from several cells of the cell type.

IV. Evaluations

We tested the algorithms on the different model optimization tasks described above, which differ in many characteristics. Every problem was evaluated with 10,000 model runs (100 generations with a population size of 100 for evolutionary and related algorithms) to get a unified evaluation. We tested, compared and visualized the performance of the algorithms using several different methods.

A. *Error space*

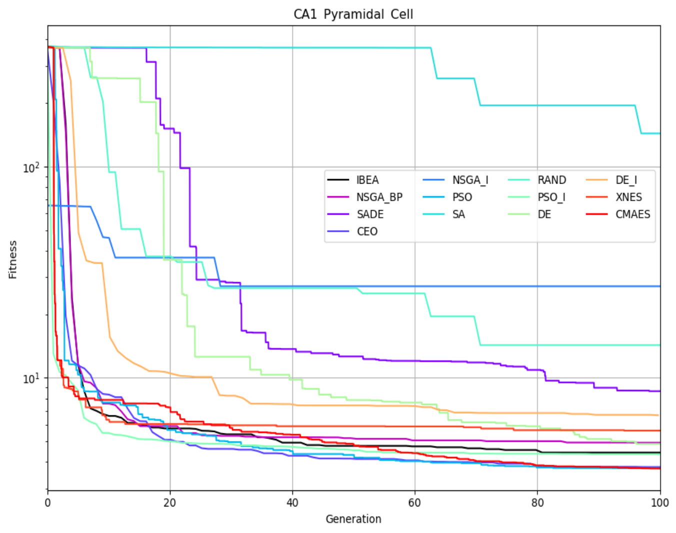

Fig. 6. Generation plot of all the algorithms used to optimize the CA1 pyramidal cell use-case. The figure shows the median over 10 consecutive runs of the cumulative minimum error.

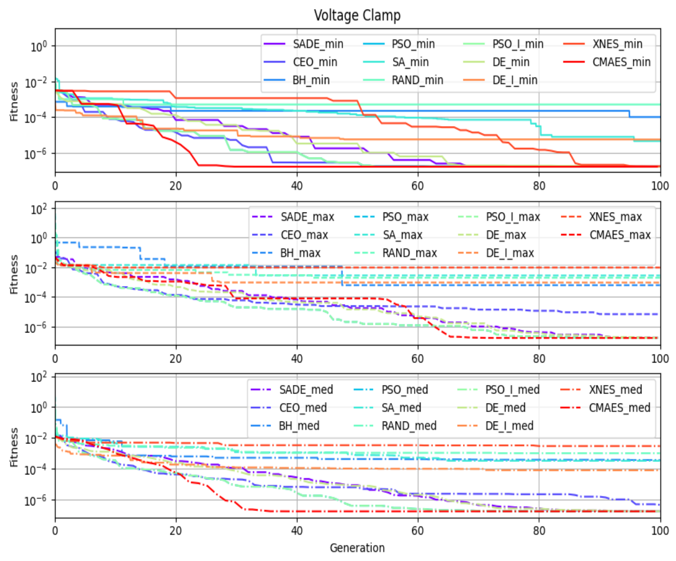

Fig. 7. Generation plot of all the single-objective algorithms used to optimize the Voltage Clamp use-case. The figure contains the minimum (solid), median (dash dotted) and the maximum (dashed) error scores of 10 independent optimization runs. The cumulative minimum error for each run up to the given number of generations was used for all statistics.

The generation plot shows the fitness (error) values through consecutive generations of the optimization. We calculate the cumulative minimum error in 10 independent optimizations with different random seeds, and then their median to get the most robust statistic and the typical values (Fig. 6). This tells us which algorithms typically find the best solutions after a given number of model evaluations. The minimum and the maximum are also calculated to observe how good the algorithm is in the best case, and whether it gets stuck in some cases (Fig. 7).

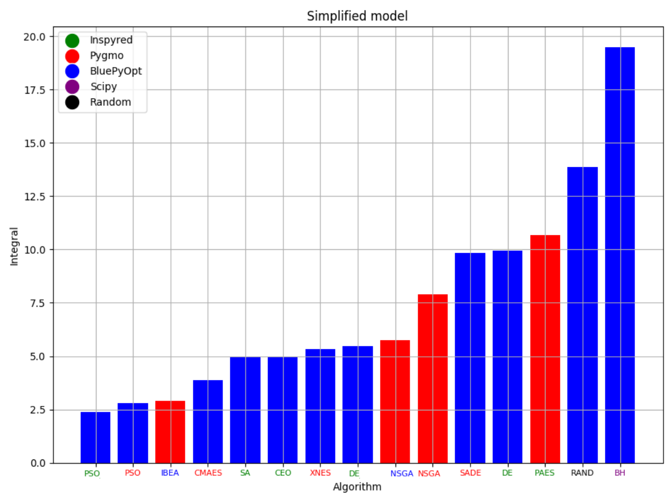

Fig. 8. Comparison of the different algorithms for the simplified model use case based on the area under the median curve of the generation plot. The algorithms are represented on the x-axis. The single-objective algorithms are represented by blue bars and the multi-objective ones by red bars. The plot is sorted from the best-performing algorithm to the worst.

In the case of more complex, detailed models, each model evaluation (simulation) can be time-consuming, and thus we are also interested in which algorithms can find a reasonably good solution in a relatively short time. This means that algorithms with better initial performance are better for these problems. To calculate the convergence speed of an algorithm, we used the integral of the curve representing the median of the 10 evaluations, represented in Fig. 8. The smaller the area under the generation curve, the faster the algorithm found a relatively good solution.

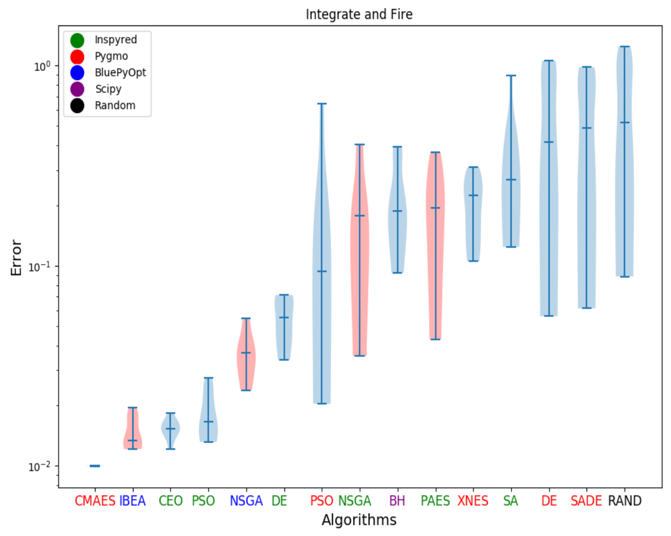

Fig. 9. Violin plot representing the distribution of the best error scores over 10 independent runs of all the algorithms for the Extended integrate-and-fire model use case. The algorithms are represented on the x-axis, and error rates on the y-axes. The legend shows the color code applied to the package names, showing the implementing packages. The single-objective algorithms are colored by blue and the multi-objective ones by red. Results are sorted by the median score, from the best to the worst.

The violin plot (Fig. 9) represents the distribution of the best results achieved in 10 independent runs (marking the minima, maxima, and median values).

B. Parameter Space

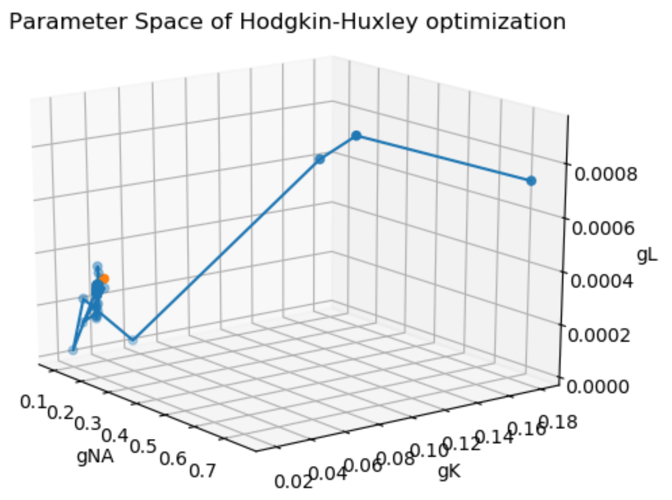

Fig. 10. Convergence of Indicator Based Evolutionary Algorithm in parameter space for the classical Hodgkin-Huxley model use case. The three axes are the conductances of Na+, K+ and leak channels.

We also investigated the convergence of optimization algorithms in parameter space. The Hodgkin-Huxley model has three parameters to optimize (conductance densities of Na, K, and leak channels), which we can visualize in a 3D point cloud plot. To observe how the algorithms progress through generations in parameter space, we find the parameter combination with the lowest error score in every generation (blue dots) and represent it in 3D space, along with the original parameter values (orange dot). The plot in Fig. 10 shows results from an optimization by the Indicator Based Evolutionary Algorithm (IBEA).

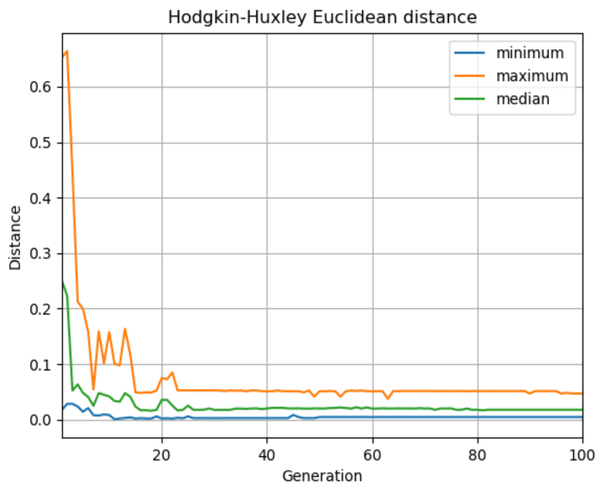

Fig. 11. Minimum, maximum and median Euclidean distance in parameter space of 10 consecutive runs of Indicator Based Evolutionary Algorithm optimized on a classical Hodgkin-Huxley model.

However, the 3D point cloud visualizes only one optimization of an algorithm, and visualization becomes more challenging in higher dimensional spaces. Therefore, we also calculated the Euclidean distance from the target parameter set, and plot the minimum, median and maximum distance value of 10 independent runs for each generation (Fig. 11). Parameters are normalized (to their possible ranges) to be considered equally in the calculations. This measure can be represented for parameter spaces of arbitrarily high dimension.

V. Ranking Algorithms

Finally, to summarize the performance of the algorithms in general, we wanted to determine how well they typically perform on these neuronal problems. Therefore, we summed their rankings on all the tests, and sorted the results in ascending order. We considered two different versions of this ranking. In one version, the final error scores were based on the median of the best error scores achieved in 10 independent optimizations (as shown in the violin plot in Fig. 9). In the other version, we ranked the algorithms according to their convergence speed, calculated from the integral of the median fitness score across generations (see Fig. 8). Fig. 12 displays ranks summed over all of our benchmarks (including both single- and multi-objective problems) for the single-objective algorithms, while Fig. 13 shows these aggregate scores for only the multi-objective problems, but including all the algorithms tested.

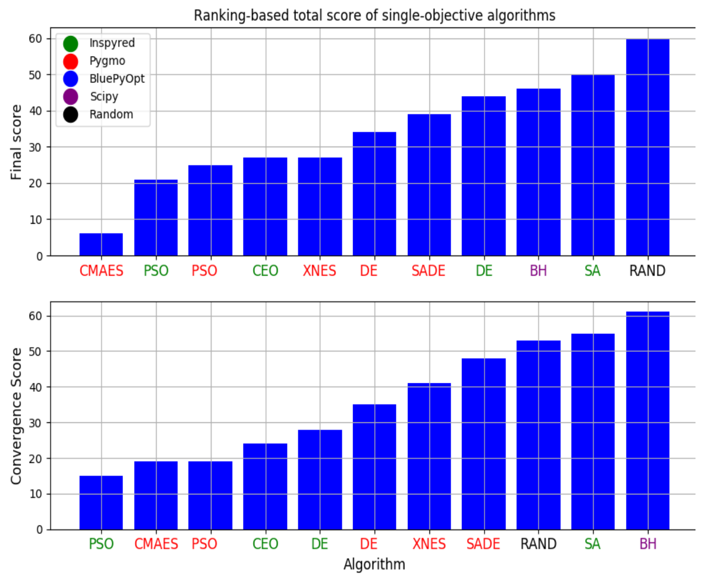

Fig. 12. Bar plots of the single-objective algorithms ordered by the sum of the ranks they achieved on violin plots and integral plots of all the use-cases.

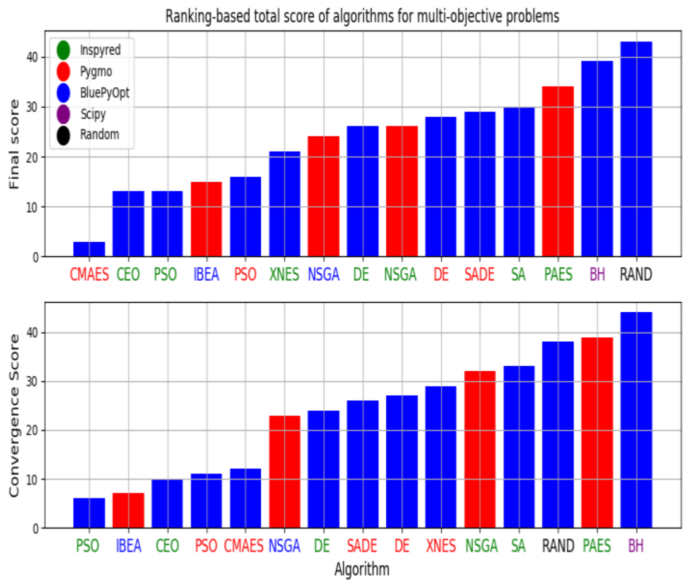

Fig. 13. Bar plots of all the algorithms tested, ordered by the sum of the ranks they achieved on violin plots and integral plots of all the multi-objective use-cases. The algorithms are represented on the x-axis. The single-objective algorithms are shown in blue and the multi-objective ones in red.

VI. Conclusions

Our software is an easy to use, extensible, general-purpose tool for fitting the parameters of neuronal models, which allowed us to systematically test the efficiency of a variety of algorithms on a set of different test cases. We found that although the relative performance of the algorithms depended on the nature of the problem, several algorithms (including CMAES, CEO, PSO, and IBEA) delivered consistently good results across our entire test suite, even for higher-dimensional, multi-objective problems. Therefore, we would recommend trying these algorithms first for novel optimization problems. We observed only minor differences across multiple implementations of the same algorithm. We hope to extend our test suite with new problems and algorithms, so that we can track new developments, and offer reliable solutions for an increasing variety of neural optimization problems.

Acknowledgement

We received funding from the European Union's Horizon 2020 Framework Programme for Research and Innovation under the Specific Grant Agreements No. 720270 and No. 785907 (Human Brain Project SGA1 and SGA2), from the Széchenyi 2020 Program of the Human Resource Development Operational Program, and from the Program of Integrated Territorial Investments in Central-Hungary (EFOP-3.6.2-16-2017-00013 and 3.6.3-VEKOP- 16-2017- 00002).

REFERENCES

[1] Vanier, M.C., Bower, J.M. A Comparative Survey of Automated Parameter-Search Methods for Compartmental Neural Models. J Comput Neurosci 7, 149–171 (1999). https://doi.org/10.1023/A:1008972005316

[2] Druckmann, S., Berger, T.K., Hill, S. et al. Evaluating automated parameter constraining procedures of neuron models by experimental and surrogate data. Biol Cybern 99, 371 (2008). https://doi.org/10.1007/s00422-008-0269-2

[3] Van Geit, W., Achard, P., and De Schutter, E. (2007). Neurofitter: a parameter tuning package for a wide range of electrophysiological neuron models. Front. Neuroinform. 1:1. doi: 10.3389/neuro.11.001.2007

[4] Friedrich P, Vella M, Gulyás AI, Freund TF and Káli S (2014) A flexible, interactive software tool for fitting the parameters of neuronal models. Front. Neuroinform. 8:63. doi: 10.3389/fninf.2014.00063

[5] N. T. Carnevale and M. L. Hines, The NEURON Book, 1st ed. New York, NY, USA: Cambridge University Press, 2009

[6] W. van Geit, R. Moor, R. Ranjan, C. Roessert, and L. Riquelme, “Electrophys Feature Extraction Library.” 2020. [Online]. Available: https://github.com/BlueBrain/eFEL

[7] Francesco Biscani et al. 10.5281/zenodo.3364433

[8] Van Geit W, Gevaert M, Chindemi G, Rössert C, Courcol J, Muller EB, Schürmann F, Segev I and Markram H (2016). BluePyOpt: Leveraging open source software and cloud infrastructure to optimise model parameters in neuroscience. Front. Neuroinform. 10:17. doi: 10.3389/fninf.2016.00017

[9] Garrett, A. „Inspired Intelligence Initiative, inspyred: Bio-inspired Algorithms in Python,” 2012. [Online]. Available: http://pythonhosted.org/inspyred/.

[10] Pauli Virtanen et al. (2019) SciPy 1.0–Fundamental Algorithms for Scientific Computing in Python. preprint arXiv:1907.10121

[11] D. Vasiljevic, Classical and Evolutionary Algorithms in the Optimization of Optical Systems (Springer Science & Business Media, 2012).

[12] J. Kennedy and R. Eberhart, "Particle swarm optimization," *Proceedings of ICNN'95 - International Conference on Neural Networks*, Perth, WA, Australia, 1995, pp. 1942-1948 vol.4.doi: 10.1109/ICNN.1995.488968

[13] Kirkpatrick, S., Gelatt, C. D. Jr., and Vecchi, M. P. (1983). Optimization by simulated annealing. Science 220, 671–680. doi: 10.1126/science.220.4598.671

[14] Price, K. (1996), Differential Evolution: A Fast and Simple Numerical Optimizer, NAFIPS’96, pp. 524–527.

[15] K. Deb, A. Pratap, S. Agarwal and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," in IEEE Transactions on Evolutionary Computation, vol. 6, no. 2, pp. 182-197, April 2002. doi: 10.1109/4235.996017

[16] Knowles, Joshua & Corne, David. (1999). The Pareto Archived Evolution Strategy: A New Baseline Algorithm for Pareto Multiobjective Optimisation. Proceedings of the 1999 Congress on Evolutionary Computation, CEC 1999. 1. 10.1109/CEC.1999.781913.

[17] Wales, David & Doye, Jonathan. (1998). Global Optimization by Basin-Hopping and the Lowest Energy Structures of Lennard-Jones Clusters Containing up to 110 Atoms. The Journal of Physical Chemistry A. 101. 10.1021/jp970984n.

[18] Nelder, J. A., and Mead, R. (1965). A simplex method for function minimization. Comput. J. 7, 308–313. doi: 10.1093/comjnl/7.4.308

[19] Byrd, R. H.; Lu, P.; Nocedal, J.; Zhu, C. (1995). "A Limited Memory Algorithm for Bound Constrained Optimization". SIAM J. Sci. Comput. 16 (5): 1190–1208. doi:10.1137/0916069.

[20] A. K. Qin and P. N. Suganthan, "Self-adaptive differential evolution algorithm for numerical optimization," 2005 IEEE Congress on Evolutionary Computation, Edinburgh, Scotland, 2005, pp. 1785-1791 Vol. 2.
doi: 10.1109/CEC.2005.1554904

[21] Hansen and Ostermeier, 2001. Completely Derandomized Self-Adaptation in Evolution Strategies. Evolutionary Computation

[22] T Glasmachers, T Schaul, S Yi, D Wierstra (2010) Exponential Natural Evolution Strategies

[23] Karaboğa, Derviş (2005). "An Idea Based on Honey Bee Swarm For Numerical Optimization"

[24] Zitzler, Eckart & Künzli, Simon. (2004). Indicator-Based Selection in Multiobjective Search. Conference on Parallel Problem Solving from Nature (PPSN VIII). 832-842. 10.1007/978-3-540-30217-9_84.

[25] Rastrigin, L.A. (1963). "The convergence of the random search method in the extremal control of a many parameter system". Automation and Remote Control. 24 (10): 1337–1342.

[26] Hodgkin AL, Huxley AF (August 1952). "A quantitative description of membrane current and its application to conduction and excitation in nerve". The Journal of Physiology. 117 (4): 500–44. doi:10.1113/jphysiol.1952.sp004764. PMC 1392413. PMID 129912 37.

[27] Káli, S., and Freund, T. F.Distinct properties of two major excitatory inputs to hippocampal pyramidal cells: a computational study. Eur. J. Neurosci. 22, 2027–2048. (2005) doi: 10.1111/j.1460-9568.2005.04406.x