Neuroevolving Efficiency in Microscopic, Nonequilibrium Systems

Joe Cavanagh

1 Introduction

Since its invention sparked the industrial revolution, thermodynamic control has been one of the essential disciplines of modern science and engineering. Generally, this discipline can be thought of as the design of a protocol that moves a system from one thermodynamic state to another in the best manner possible. What one means by "best" is somewhat dependent on the task, but one common metric is one of efficiency. The procedure that moves a system from one state to another, possibly subject to constraints, with the minimum expenditure of energy is the most efficient.

More recently than the days of Carnot, general theory for this has been developed using the correspondance of a thermodynamic system to a metric space, an analogy which has been known for some time.[1][2] Specifically, Fischer information yields a metric that defines a Riemannian manifold, along which geodesic paths between two states correspond to the most efficient procedures.[3] This was extended to nonequilibrium paths using linear response.[4] Because of their generality, these procedures have generated excitement for the possibility of use in nanotechnology. As long as this Riemannian surface can be defined, efficient protocols can be defined.

A few billion years before Carnot, nanotechnology more efficient and advanced than any steam engine was being designed and assembled. We happen to call it life. The method for its design was simple and self-serving: nanotechnology that worked efficiently could replicate itself using fewer resources and proliferate faster than its inefficient competitors. Changes were tested through repeated mutation and selection, which resulted in increasingly efficient protocols. Evolution through natural selection also has the advantage of working for complicated systems as well as simple ones— while deliberate design through analytic mathematics only works for systems where these mathematics can be feasibly computed. It doesn't matter how complicated an enzyme is— evolution only cares about well it works.

The mechanism of evolution is the principle behind genetic algorithms. Due to the abundance of computational power today compared to the past, these have become increasingly feasible for solving problems. On the topic of thermodynamic control, genetic algorithms have been used for heat engine design, resulting in reproduction of the Carnot, Otto, and Stirling cycles.[5] A similar procedure designed an efficient protocol for a theoretical model of molecules. If evolution through selection could design efficient nanoscopic protocols in the past, what's to say it can't do the same today?

This project attempts to take another step in this direction. Specifically, we will examine

the use of genetic algorithms to design optimal protocols of a microscopic thermodynamic toy system of a particle in a harmonic well undergoing inertial Langevin dynamics. The thermodynamic parameters of the system that can undergo change are the inverse temperature scaled by Boltzmann's constant, β, and the spring constant k of the harmonic well. More generally, the minimum potential location of the harmonic well, y_0 can also be changed– although in this project we will consider it a constant set to 0 for the sake of visualization. This is a system for which the most efficient protocols have been thoroughly analyzed, so we can easily check the work of the genetic algorithm using the analytical results of previous study.[6]

2 Theory

2.1 Particle Diffusing in a Harmonic Well

Most of this section comes from the paper "Geometry of Thermodynamic Control".[6] This is because we're studying precisely the same system. We're specifically interested in some protocol $\lambda(t)$ which consists of both $\beta(t)$ and $k(t)$. The excess work, or total dissipation, is given by

$$\langle \beta W \rangle = \int_0^t \dot{\beta} \frac{\langle p^2 \rangle}{2m} + \frac{\langle (y - y_0)^2 \rangle}{2} (k\dot{\beta} + \dot{k}\beta) + k\beta \dot{y_0} \langle y_0 - y \rangle - \frac{\dot{\beta}}{\beta} - \frac{\dot{k}}{2k} \tag{1}$$

Which, if we arbitrarily set $y_0 == 0$, since we won't be changing it, we get

$$\langle \beta W \rangle = \int_0^t \dot{\beta} \frac{\langle p^2 \rangle}{2m} + \frac{\langle y^2 \rangle}{2} (k\dot{\beta} + \dot{k}\beta) - \frac{\dot{\beta}}{\beta} - \frac{\dot{k}}{2k} \tag{2}$$

Here, y is the position of the particle, brackets denote an average taken over the probability distribution f, p denotes momentum, and m denotes the mass of the particle. The average

$$\frac{d}{dt} \langle y \rangle = \langle p \rangle / m \tag{3}$$

by the definition of momentum, $m\frac{dy}{dt}$. Then,

$$\frac{d}{dt} \langle p \rangle = - \langle p \rangle \frac{\zeta_c}{m} - k \langle y \rangle \tag{4}$$

This is just Newton's Second Law where the first term is the resistance due to friction and the second term is Hooke's Law. Here, ζ_c is the standard Cartesian friction coefficient. Additionally,

$$\frac{d}{dt} \langle py \rangle = \frac{\langle p^2 \rangle}{m} - k \langle y^2 \rangle - \langle py \rangle \frac{\zeta_c}{m} \tag{5}$$

By the product rule and the previous two relations. Finally, by the same method, we get

$$\frac{d}{dt} \langle y^2 \rangle = 2 \langle py \rangle / m \tag{6}$$

Finally, Langevin dynamics means that

$$\frac{d}{dt}\left\langle p^2\right\rangle = -2k\left\langle py\right\rangle - 2\left\langle p^2\right\rangle \frac{\zeta_c}{m} + \frac{2\zeta_c}{\beta} \tag{7}$$

The first two terms here fall out of similarly applying the product rule, and the last term is a result of Brownian motion widening the momentum distribution. We can solve these five coupled differential equations along with the initial conditions of a particle at rest at the bottom of the well:

$$\left\langle y\right\rangle (0) = 0 \tag{8}$$
$$\left\langle p\right\rangle = 0 \tag{9}$$
$$\left\langle py\right\rangle (0) = 0 \tag{10}$$
$$\left\langle y^2\right\rangle (0) = \frac{1}{k(0)\beta(0)} \tag{11}$$
$$\left\langle p^2\right\rangle (0) = \frac{m}{\beta(0)} \tag{12}$$
$$\tag{13}$$

The solutions to these coupled equations give us values that we can plug into the integral in equation (2) to find the total dissipation of some trajectory.

2.2 Reinforcement Learning as a Solution

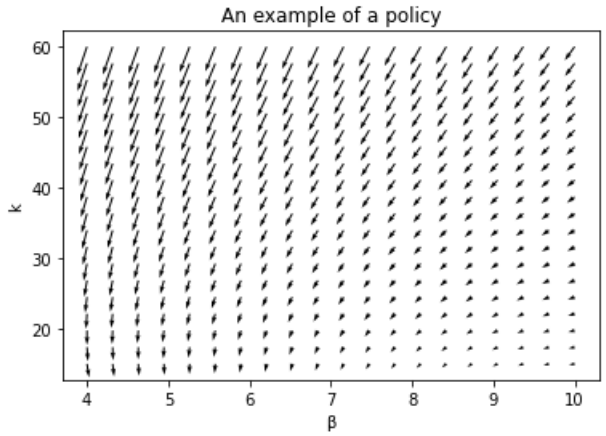

Fig. 1: An example of a policy as a vector valued function. At any given point, the network will dictate that the protocol move in the direction of the arrow with magnitude proportional to the length of the arrow.

Finding the optimal protocol for a system moving from one state to another is in many ways similar to playing a game like Chess, Go, or many of the Atari titles, all of which have been effective arenas for reinforcement learning. All of the above tasks consist of finding the best series of decisions to move through a potentially unknown environment. These decisions can be anything from moving chess pieces and pressing Atari buttons to changing the temperature and spring constant of some physical system.

Formally, reinforcement learning consists of an *agent* in an *environment*. The agent follows a *policy*, which is a function with an input that consists of the environment and an output that consists of a probability distribution over decisions. Here, our policy is simpler– it is a function from the environment (β, k) to the next step of the protocol $(\Delta\beta, \Delta k)$. We can visualize the policy, as a 2d vector-valued function over β and k, where the arrow corresponds to the

direction of the protocol at that point. Another way of thinking about this is that the arrow corresponds to the direction that the "agent" will "move" if it finds itself at a certain point. In Reinforcement learning, agents seek to maximize some reward.

2.3 Neuroevolution

This project's genetic algorithm is similar to that described in Beeler et al. [5] Each "creature" takes the form of a neural network. The first layer I has two neurons, representing the current β and k of the system. In other words, $I_1 = C_\beta \beta$ and $I_2 = C_k k$. The constants here are meant for scaling, since neural networks tend to perform best when the inputs vary similarly. The second layer is made of n hidden neurons, taking on values

$$H_j = \frac{1}{2} \tanh\left(\sum_{i=1}^{2} w_{j,i}^{(1)} I_i \right) + b_j$$

where $w^{(1)}$ is a weight matrix of size $n \times 2$ and b is a bias vector of size n. The outputs are given by

$$O_k = \sum_{j=1}^{2} w_{k,j}^{(2)} H_j$$

where $w^{(2)}$ is also a weight matrix, of size $2 \times n$. The output neuron values give the policy; O_1 tells us how much to change β by (scaled accordingly) and O_2 tells us how much to change k by (also scaled accordingly). Scaling here is also important, since the initial outputs will probably vary similarly, but changes in k and β might be needed on very different scales. This is not to say that running a genetic algorithm on a network with unscaled inputs and outputs cannot work— just that there's reason to believe it will take longer for the best parameters to evolve.

3 Numerics

The code was largely split into two parts. The first of these parts computed the dissipation given some trajectory. The second seeks to find the optimal protocol for minimizing this dissipation through a policy.

3.1 Calculation of Entropy

We can solve the numeric equations using the explicit Runge-Kutta method of order 5(4) as implemented by Scipy.[7]. We assume a time of $t = 100$ seconds for the protocol to take place, with each step in the protocol taking 1 second. We also set $m = 1$ and $\zeta_c = 1$ for convenience. To keep things consistent, we also calculate the dissipation along a trajectory that starts at $(\beta, k) = (4, 60)$ and ends at $(\beta, k) = (10, 15)$.

3.2 Genetic Algorithm Implementation

We start by initializing 100 random neural networks. Specifically, we pick independent and identically distributed random numbers from a Gaussian distribution with mean 0 and variance 1 for the weights and biases for 100 separate neural networks of the type described in section 2.3. We then let each follow its protocol, starting at $(\beta, k) = (4, 60)$. We score the protocols based on their performance.[1] If the protocol ends sufficiently close to the intended stopping point, $(\beta, k) = (10, 15)$, we connect the protocol to this endpoint and score based on the entropy for the total protocol– including the ad-hoc connection to the intended endpoint. If not, we score the protocol based on its distance from the endpoint. This ensures that the best-scoring protocol is the optimal trajectory, and that trajectories that reach for the endpoint are selected for, while balancing computational cost.

After all 100 neural network policies are scored based on the protocols they generate. The 25 neural networks whose protocols score the highest are kept, and the rest are deleted. Each of the top 25 neural networks is copied three times, and each copy has all of its weights and biases modified by the addition of $0.05X_i$, where each X_i is picked iid from a Gaussian distribution with mean 0 and variance 1. This is analogous to each of the protocols surviving and reproducing in the natural world due to their "fitness" measured by generating a good protocol.

We can gain an intuition from the optimal number of neuron nodes by running tests with several of them. This is precisely what's done in Figure 2. Here, the genetic algorithm was run using populations of neural networks with 1, 4, 16, 64, 256, 1024, 4096, or 16384 hidden neurons. These were run for 32 generations each, and the resulting trajectories at selected generations and minimum dissipations are shown in Figure 2.[2] A trend can be seen here: if there are too few hidden neurons, then there are too few parameters in the network to fit to the optimum curve. If there are too many hidden neurons, then there are so many parameters that evolving all of the to be optimal takes too long. In theory, the best performance of a 16,384 hidden-neuron network is at least as good as all the others, but it is overshadowed by the 1024 and 256 hidden-layer-neuron networks. Because of its performance here, *all other figures will use a network with 1024 hidden neurons.*

In order to convey an intuition for the nature of these genetic algorithms, Figures 3 and 4 show the most and least successful protocols in each generation. Likewise, to convey a sense of the policies generated by each network, the most successful policies for protocol generation in selected generations are visualized in Figure 5. The best policy evolved (also given in the final plot in Fig. 5) was plotted against the analytically-derived result by Zulkowski et.al. in curve 6.[**?**, main]ll of these figures were produced with the publicly-available code on github.

[1] Choosing the scoring here is somewhat of an art. Calculating the excess work is both computationally costly and absurd for a trajectory that goes in the opposite direction from the intended endpoint. To incentivize the protocols to end in the intended place by giving them a score based on how far the endpoint is from the intended endpoint (here we use the L_2 norm). However, we ultimately are interested in the path with the least excess work. We end up using a hybrid approach here.

[2] The minimum dissipation is listed as "xs work" on the top of each plot. This represents the most efficient path of all of the protocols, when connected to the final point.

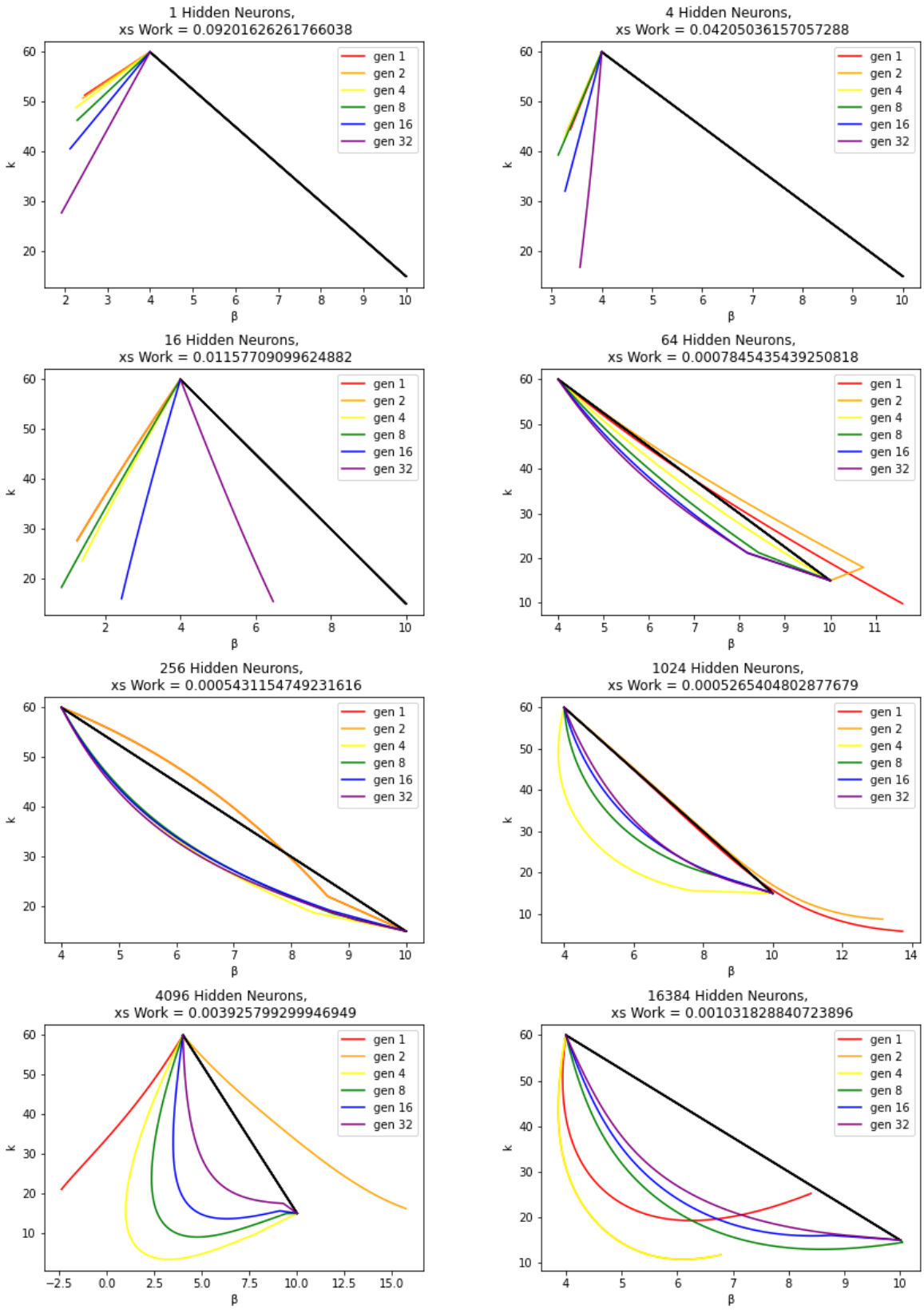
1 Hidden Neurons,
xs Work = 0.09201626261766038
4 Hidden Neurons,
xs Work = 0.04205036157057288
16 Hidden Neurons,
xs Work = 0.01157709099624882
64 Hidden Neurons,
xs Work = 0.0007845435439250818
256 Hidden Neurons,
xs Work = 0.0005431154749231616
1024 Hidden Neurons,
xs Work = 0.0005265404802877679
4096 Hidden Neurons,
xs Work = 0.003925799299946949
16384 Hidden Neurons,
xs Work = 0.001031828840723896
gen 1
gen 2
gen 4
gen 8
gen 16
gen 32
k
β

Fig. 2

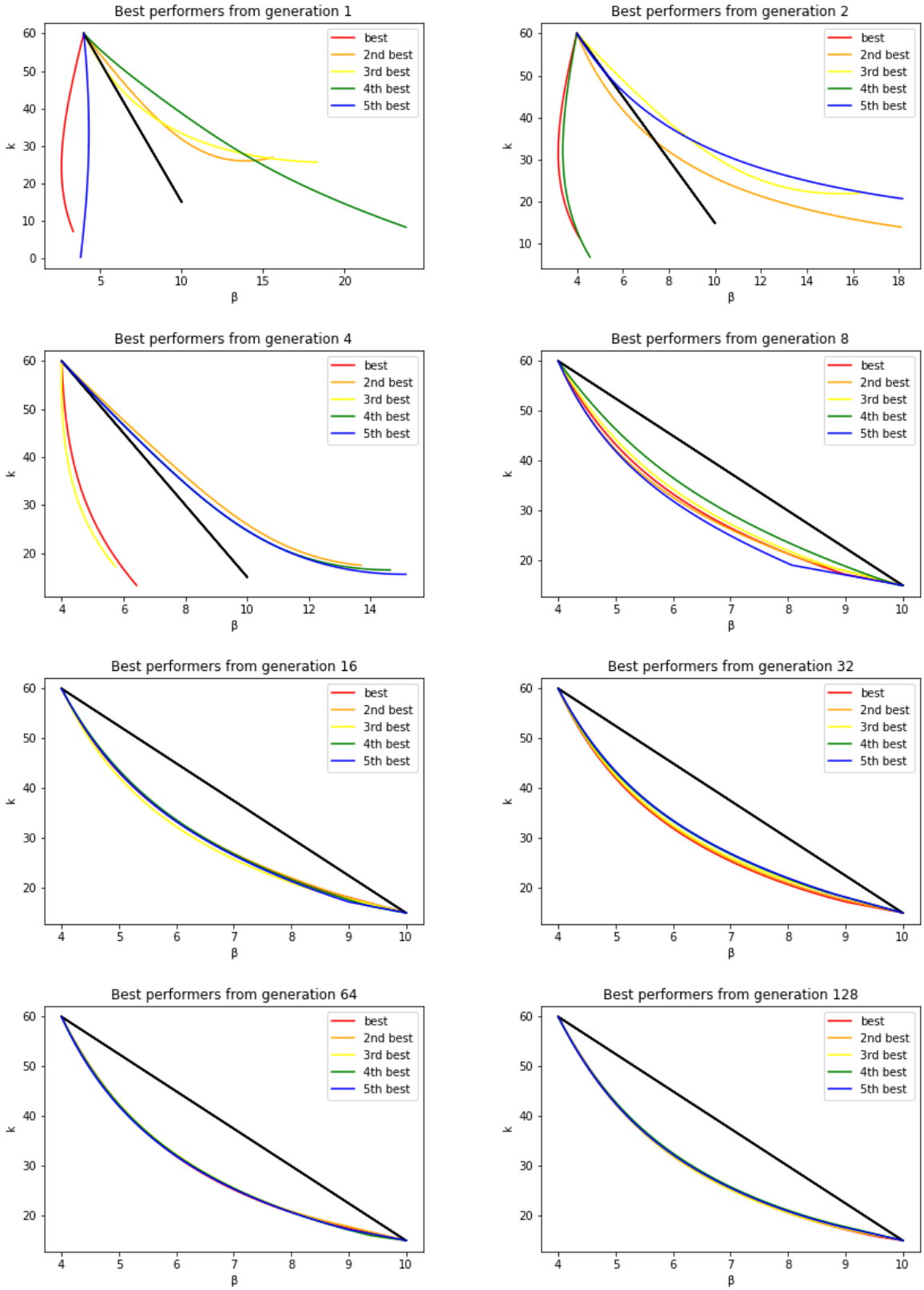

Fig. 3: The best performers from selected generations. Note how the variability of these paths decreases as the generations go by. This is an example of the genetic algorithms balancing early "exploration" with later "exploitation". The black line is the straight line from the beginning state to the final state.

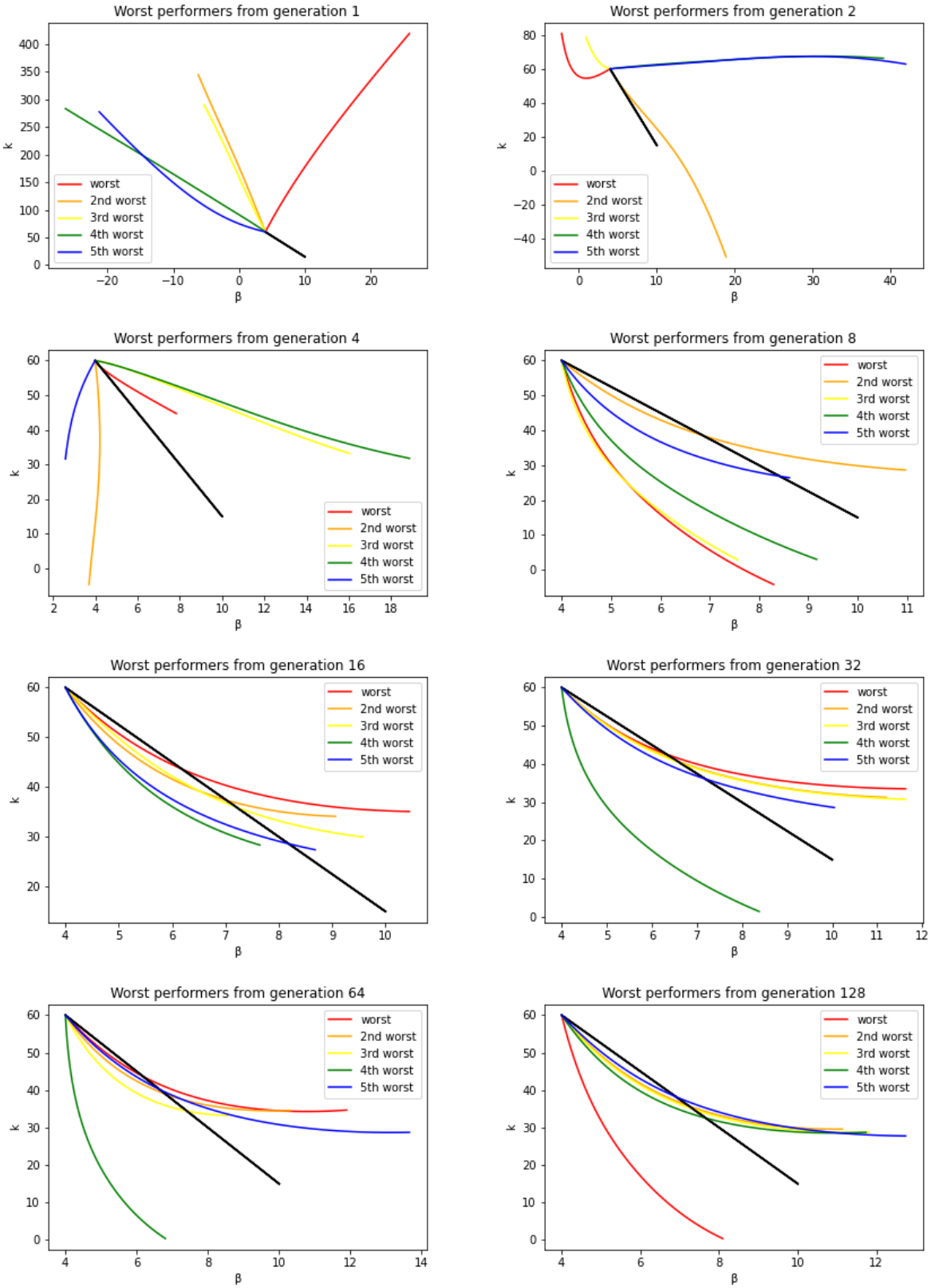

Fig. 4: The worst performers from selected generations. Note how different the early ones are from the later ones. This is because the genes from successful early trajectories pervade the population of later trajectories.

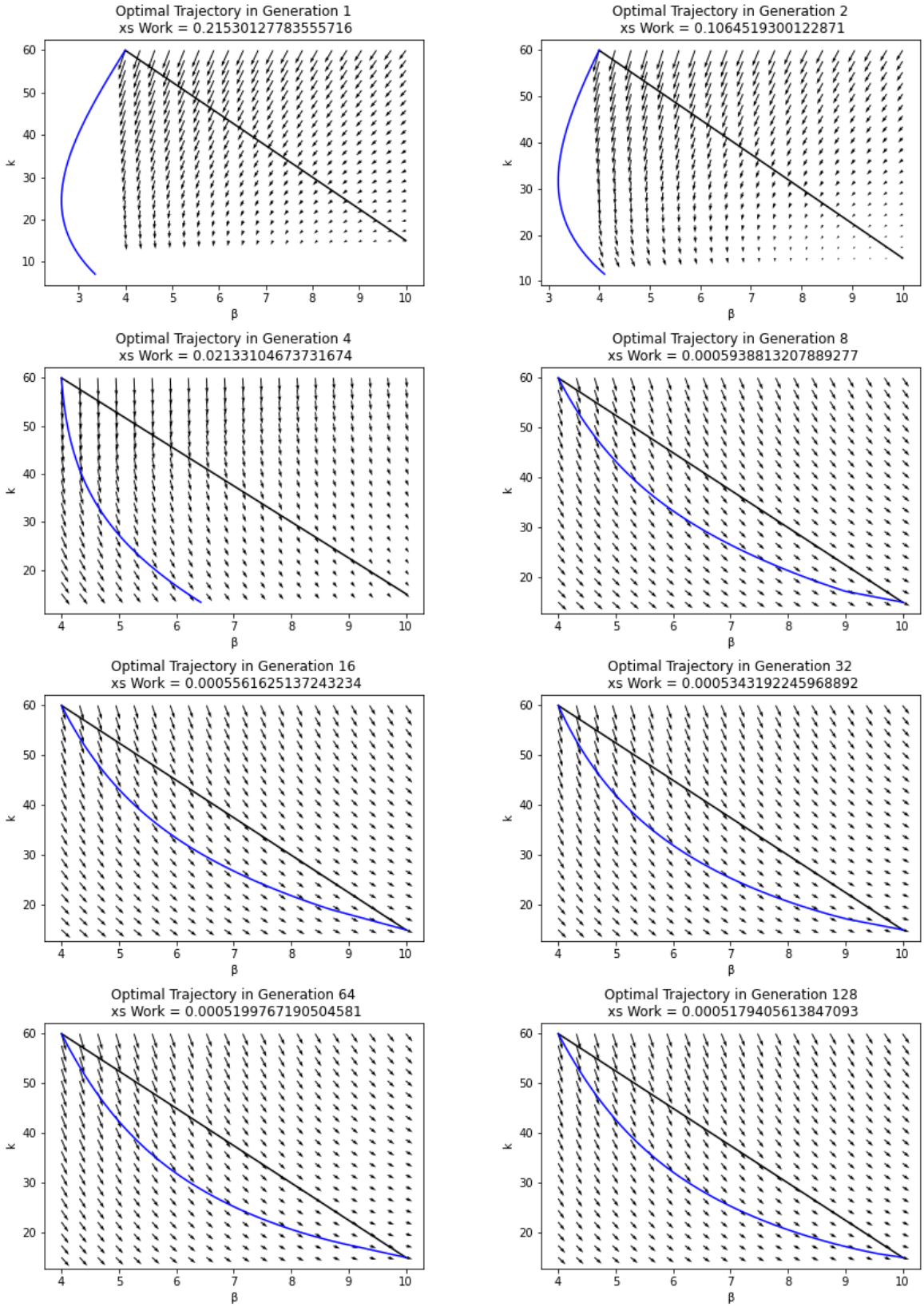

Fig. 5: The best-performing policies from each generation, visualized as a vector field. Note how the change becomes increasingly subtle, even after exponentially more generations separate each plot

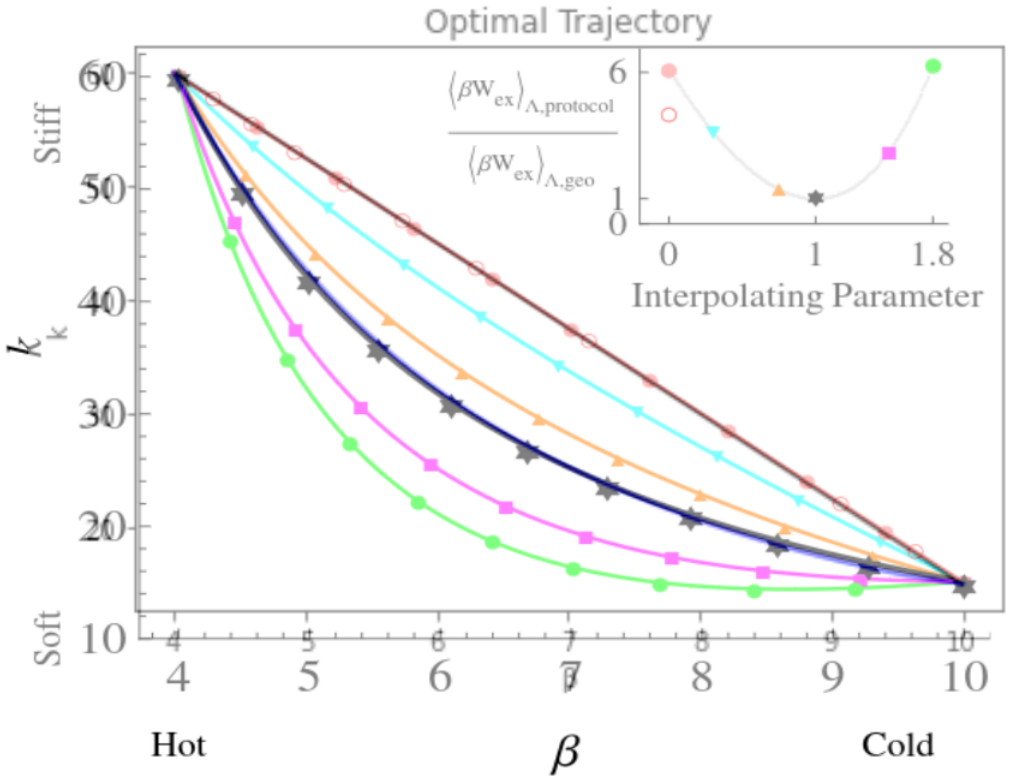

Fig. 6: The best-scoring protocol plotted against the analytically-derived optimal protocol in Zulkowski et.al.[**?**, main]he genetic-algorithm derived optimal protocol is in blue, and the analytically proven optimal protocol is in black. The curves are nearly identical. The two straight lines are also plotted together.

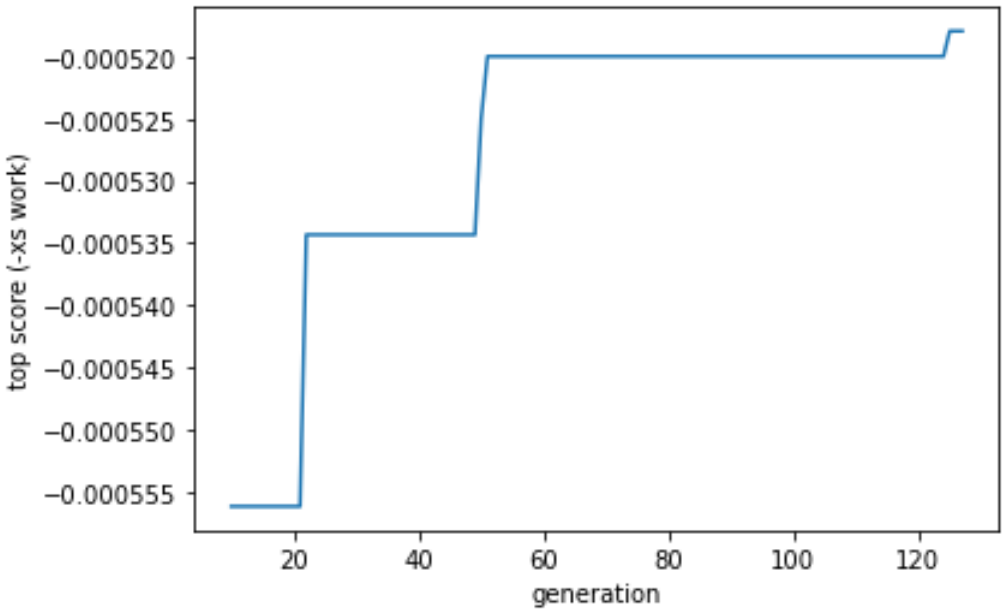

Fig. 7: How the score improved with time. Note that the innovation happened in a relatively small number of generations.

4 Discussion

This study presents strong evidence that finding maximally efficient protocols using genetic algorithms is a feasible endeavor, by virtue of our genetic algorithm finding the optimal path previously found using the Fisher information metric.[6] However, the optimal usage of genetic algorithms for this process is relatively unknown.

One major pitfall was encountered in this project. If the scores were setup incorrectly, such as by penalizing distance from the target state too harshly, then a local optimum was quickly reached— the descendants of one neural network that generated a protocol leading almost directly to the final state came to dominate the gene pool. The necessary variation to find the optimal path was simply not present, since the networks were not allowed to roam the space of all possible solutions freely. This may become an issue with the current scoring if the optimal path deviates significantly from a straight line.

All code used is available at `https://github.com/jmcavanagh/neuroevolution_ho`

4.1 Future Directions

This study was concerned with optimizing a trajectory, but the solutions took the form of a policy of movements to be made at any given point along a trajectory, as a way to get to some destination. Such a policy could be extended into one that gives the optimal change at any given point *in general* to arrive at a destination. This may be able to be accomplished by varying the starting location during each generation and between generations, and scoring the network policies based on overall performance.

Additional investigation with different selection processes could also yield valuable re-

sults. Commonly, genetic algorithms include some kind of crossing over, where "genes" from separate "organisms" can combine, analogous to sexual reproduction. Additionally, the selection criteria could be modified– perhaps a different distribution of successes could improve the simulation.

References

[1] F. Weinhold; Metric geometry of equilibrium thermodynamics. J. Chem. Phys. 15 September 1975; 63 (6): 2479–2483. https://doi.org/10.1063/1.431689

[2] G. Ruppeiner; Thermodynamics: A Riemannian geometric model. Phys. Rev. A 20, 1608 – Published 1 October 1979

[3] Gavin E. Crooks; Measuring Thermodynamic Length Phys. Rev. Lett. 99, 100602 – Published 7 September 2007

[4] David A. Sivak and Gavin E. Crooks; Thermodynamic Metrics and Optimal Paths Phys. Rev. Lett. 108, 190602 – Published 8 May 2012

[5] Chris Beeler, Uladzimir Yahorau, Rory Coles, Kyle Mills, Stephen Whitelam, and Isaac Tamblyn; Optimizing thermodynamic trajectories using evolutionary and gradient-based reinforcement learning Phys. Rev. E 104, 064128 – Published 20 December 2021

[6] Zulkowski, Patrick, Sivak, David, Crooks, Gavin, Deweese, Michael. (2012). Geometry of thermodynamic control. Physical review. E, Statistical, nonlinear, and soft matter physics. 86. 041148. 10.1103/PhysRevE.86.041148

[7] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, CJ Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E.A. Quintero, Charles R Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. (2020) SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. Nature Methods, 17(3), 261-272.