OPT2021: 13th Annual Workshop on Optimization for Machine Learning

Optimization with Adaptive Step Size Selection from a Dynamical Systems Perspective

Neha S. Wadia NEHA.WADIA@BERKELEY.EDU
Michael I. Jordan JORDAN@CS.BERKELEY.EDU
University of California, Berkeley

Michael Muehlebach MICHAELM@TUEBINGEN.MPG.DE
Max Planck Institute for Intelligent Systems, Tübingen

Abstract

We investigate how adaptive step size methods from numerical analysis can be used to speed up optimization routines. In contrast to line search strategies, the proposed methods recycle available gradient evaluations. Thus, neither evaluations of the objective function nor additional gradient evaluations are required, and the computational complexity per iteration remains at $\mathcal{O}(d)$, where d is the problem dimension. On strongly convex functions, our results show a consistent improvement in the number of iterations by up to a factor of 2 with the gradient method and of 1.4 with the heavy ball method across a range of condition numbers.

1. Introduction

Choosing step sizes is an important and unavoidable task in optimization. The choice of step size can determine whether or not an algorithm converges as well as its iteration complexity. By exploiting analogies between first-order optimization algorithms and dynamical systems, we analyze the use of adaptive step size methods from numerical analysis in optimization. These methods can be applied to any algorithm with a continuous-time representation, and we find that they can often significantly reduce the number of iterations needed for convergence while leaving the computational complexity per iteration at $\mathcal{O}(d)$ where d refers to the problem dimension.

In mathematical optimization, there is a very long tradition of using steps of variable length. The methods typically fall into two categories: line search and trust region strategies. The former aims to choose a step size that sufficiently reduces the objective function at each iteration. The difficulty lies in balancing the amount of decrease with the resulting computational effort [13]. Well-known strategies with convergence guarantees are the Wolfe [17] or the Armijo-Goldstein conditions [1]. In contrast to line search, where the search direction is fixed and only the step size is varied, trust region strategies optimize over both the step size and the step direction. This is typically done by constructing a local approximation of the objective function about the current iterate. Furthermore, the step size is often restricted, ensuring that the approximation remains valid. Excellent texts on this subject include Fletcher [5] and Nocedal and Wright [13], for example.

In contrast to these techniques, our approach has its origins in the numerical analysis of differential equations. We draw on recent results that view optimization algorithms as continuous-time dynamical systems [see, e.g., 4, 8, 10, 15, 16] and apply adaptive step size techniques as a means to efficiently discretize the continuous-time equations. This provides an efficient alternative to line search and trust region methods.

The computational complexity of optimization algorithms is typically measured by how the number of iterations grows as a function of accuracy (in terms of function value or distance to the optimizer) over a given class of functions and over a set of initial conditions. While this notion of complexity often neglects constants, it has been successful at characterizing fundamental performance limits [see, e.g., 3, 11, 12] and deriving optimal algorithms. However, when facing a practical problem instance, the constants do matter and determine the actual number of iterations needed. On the class of strongly convex functions, gradient descent and the heavy ball method with specific constant step size schemes achieve their respective complexity lower bounds. We find that the adaptive step size method we develop in this work also achieves these complexity lower bounds, and improves on the constant compared to the constant step size methods.

2. Adaptive Step Size Routine

We introduce an adaptive step size routine using the specific example of gradient descent. Extensions to momentum-based optimizers are illustrated with the heavy ball algorithm. Lastly, we provide a meta-algorithm to generalize the adaptive step size method to any algorithm with a continuous-time representation as an ordinary differential equation.

We consider an objective function $f : \mathbb{R}^d \to \mathbb{R}$ with a Lipschitz-continuous gradient. For simplicity, we further assume that f has a single, isolated minimum x^* at the origin with $f(0) = 0$, and that the Lipschitz constant of the gradient is unity. We write the smallest and largest eigenvalues of the Hessian of f at x^* as μ and L, respectively, and introduce the condition number as $\kappa = L/\mu$.

2.1. Gradient Descent

Gradient descent has a well-defined continuous-time representation in the form of gradient flow,

$$\dot{x}(t) = -f'(x(t)). \tag{1}$$

Dots denote time derivatives and primes derivatives with respect to x. Gradient descent is the Euler discretization of (1). Using x_n and h_n to write the n^{th} discrete-time update and step size, respectively, this is

$$x_{n+1} = x_n - h_n f'(x_n). \tag{2}$$

There are many other discretization schemes for (1) that differ, for example, in the accuracy with which they track the underlying continuous dynamics. More accurate discretization schemes afford the ability to pick larger step sizes [6]. The central idea of our work is that we can use this fact to guide adaptation of the step size during optimization. Two ingredients are required to implement this idea: a way to measure the accuracy of the Euler step (2), and a mechanism for adapting the step size accordingly. These two ingredients are described next.

Measuring accuracy. The error of a discretization scheme is the difference between the discrete and continuous-time trajectories. Starting from $x(t) = x_n$ and applying Taylor's theorem, we have

$$x(t + h_n) - x_{n+1} = \frac{1}{2} f''(x(\xi)) f'(x(\xi)) h_n^2 = C(x_n, h_n)\, h_n^2, \tag{3}$$

for some $\xi \in (t, t + h_n)$, provided that $f'(x)$ is continuously differentiable. The function $C(x_n, h_n)$ is roughly constant for small h_n. The local error of gradient descent is therefore quadratic in h_n.

We compare this to Heun's method (also called the midpoint rule), which is a more accurate discretization of (2). Using the notation x_n^{H} for the n^{th} Heun update, this is

$$x_{n+1}^{\mathrm{H}} = x_n - \frac{1}{2}h_n \left(f'(x_n) + f'(x_{n+1})\right), \tag{4}$$

where x_{n+1} is given by (2). Again, by applying Taylor's theorem we find

$$x(t+h_n) - x_{n+1}^{\mathrm{H}} = C^{\mathrm{H}}(x_n, h_n)h_n^3, \tag{5}$$

where the function $C^{\mathrm{H}}(x_n, h_n)$ is roughly constant for small h_n. The local error of Heun's method is cubic in h_n, and so it is more accurate than (2).

For small h_n, we can therefore estimate the accuracy of gradient descent at each iteration by comparing it to the Heun update. Using $||\cdot||$ to denote the Euclidean norm, we have

$$||x(t+h_n) - x_{n+1}|| = ||x_{n+1}^{\mathrm{H}} - x_{n+1}|| + \mathcal{O}(h_n^3) \equiv \delta_n(x_n, h_n) + \mathcal{O}(h_n^3). \tag{6}$$

Although the error analysis just presented is strictly valid only for small h_n, we observe in experiments that the resulting adaptive step size routine is still effective at step sizes of order one.

A note on complexity. The Heun discretization has the advantage of being computationally efficient. Each evaluation of f' is a single oracle query for a gradient. The gradient evaluation of $f'(x_{n+1})$ in (4) can be reused in (2) for the next iteration, and so δ_n can be computed without additional evaluation of the gradient. The step size control mechanism we propose in the following therefore comes with almost no additional computational cost; in fact, it requires precisely *one* additional gradient computation in total, over all the iterations.

Adapting step size: proportional control. We use a proportional controller ('P control') to set step sizes based on the value of δ_n. P control is the simplest version of a family of control mechanisms called PID (proportional integral derivative) control. These find use as automated mechanisms for controlling a user-specified quantity in a dynamical system. The quantity we wish to control is δ_n. While the idea of using PID control to set step sizes for a discretization scheme for differential equations is not new [6, 7], our innovation here is to apply this idea to optimization.

The P control mechanism introduces two hyperparameters: r, the desired error between the Heun and gradient descent updates on any given iteration, and θ, which can be interpreted as a gain. The control mechanism gives the following prescription for modulating the step size:

$$h_{n+1} = \left(\frac{r}{\delta_n}\right)^{\theta/2} h_n. \tag{7}$$

The form of (7) is motivated by the fact that, after taking logarithms, neglecting $\mathcal{O}(h_n^3)$ terms, and using (6) and (3), we have

$$\log h_{n+1} = \log h_n + \frac{\theta}{2}\left(\log r - \log \delta_n\right) = (1-\theta)\log h_n + \frac{\theta}{2}\left(\log r - \log C\right).$$

This means that $\log h_n$ evolves roughly as a linear system, and is expected to converge to its steady state, given by $\delta_n = r$, with a rate of $1-\theta$. The feedback law (7) is called 'proportional' control because the quantity $\log h_{n+1} - \log h_n$ is proportional to the error signal $\log r - \log \delta_n$.

At each iteration during optimization, we compute the gradient descent and Heun updates, compute δ_n, and calculate the step size h_{n+1} for the *next* iteration using (7). We then apply the gradient descent update with the step size h_n recommended by the controller on the previous iteration.

We emphasize that the sole purpose of (4) is to compute δ_n and that it is never actually applied to an iterate during optimization. In addition, the use of (4) in the computation of δ_n is a specific choice; any other more accurate discretization of (1) than (2) can be chosen instead.

Setting θ and r. Formally, we have $\theta \in [0, 2]$ [7]. Note that $\theta = 0$ results in a constant step size routine. Increasing θ turns up the gain on the step size control, allowing larger changes between iterations, and potentially lowering the number of iterations required to converge. We use small values of θ, i.e., $\theta \leq 0.01$, which work for functions with condition numbers spanning three orders of magnitude. At larger values of θ we sometimes observe oscillatory behavior. See Fig. A.1 for example trajectories of gradient descent with adaptive P control on a strongly convex function at a range of values of θ.

We set $r = 0.5$ in experiments on strongly convex functions, requiring the P controller to maintain $\delta_n \leq 0.5$. The controller is less sensitive to changes in r; smaller values also work well.

In practice we find it necessary to place a few additional constraints on the P control mechanism (7). These are detailed in Appendix C.

2.2. Second and higher-order algorithms: the heavy ball method

Higher-order differential equations can be written as systems of coupled first-order equations. Using this trick, the method of the previous section can be applied with almost no modification. We illustrate this here with the example of the heavy ball method.

We work with the following continuous-time limit of the heavy ball algorithm:

$$\dot{x}(t) = p(t), \;\; \dot{p}(t) = -\frac{2}{\sqrt{\kappa}} p(t) - f'(x(t)), \tag{8}$$

where we have introduced the (momentum) variable $p = \dot{x}$. In order to retain the stability of (8) at large κ, we use the *semi-implicit* instead of the *standard* Euler discretization. Unlike the latter, the former is stable at all values of κ up to a large step size [9]. The update rule is given by

$$x_{n+1} = x_n + h_n p_{n+1}, \;\; p_{n+1} = p_n + h_n \left(-\frac{2}{\sqrt{\kappa}} p_n - f'(x_n) \right). \tag{9}$$

In contrast with the Euler method, the semi-implicit Euler method uses p_{n+1} (instead of p_n) in x_{n+1}.

In analogy with Section 2.1, we use the following Heun discretization of (8) to calculate δ_n:

$$x^{\mathrm{H}}_{n+1} = x_n + \frac{1}{2} h_n \left(p_{n+1} + p_{n+2} \right) \tag{10a}$$

$$p^{\mathrm{H}}_{n+1} = p_n + \frac{1}{2} h_n \left(-\frac{2}{\sqrt{\kappa}} (p_n + p_{n+1}) - (f'(x_n) + f'(x_{n+1})) \right), \tag{10b}$$

where x_{n+1} and p_{n+1} are given by (9) and p_{n+2} is the semi-implicit Euler update to p_{n+1}. Then $\delta_n = ||(x_{n+1}, p_{n+1}) - (x^{\mathrm{H}}_{n+1}, p^{\mathrm{H}}_{n+1})||$ where (x_n, p_n) is the vector formed by concatenating x_n and p_n, and step sizes are simply given by the P control mechanism (7).

2.3. Meta algorithm

Here we generalize the ideas in the previous two sections to any algorithm that has a well-defined continuous-time representation.

Given a discrete-time algorithm (call it algorithm A), the construction of a P-control-like mechanism for step size selection requires the following four steps: **(1)** Find the continuous-time counterpart of algorithm A. If this is a second or higher-order differential equation with respect to time, rewrite it as a system of coupled first-order equations. **(2)** Identify a higher-order discretization (call this algorithm B) of the continuous-time system. Typically, computational efficiency will guide this choice. **(3)** At each iteration during optimization, compute the difference between the updates of algorithms A and B. Feed this difference to a proportional controller with appropriately set hyperparameters. **(4)** Apply the update of algorithm A with the step size recommended by the controller.

3. Experiments

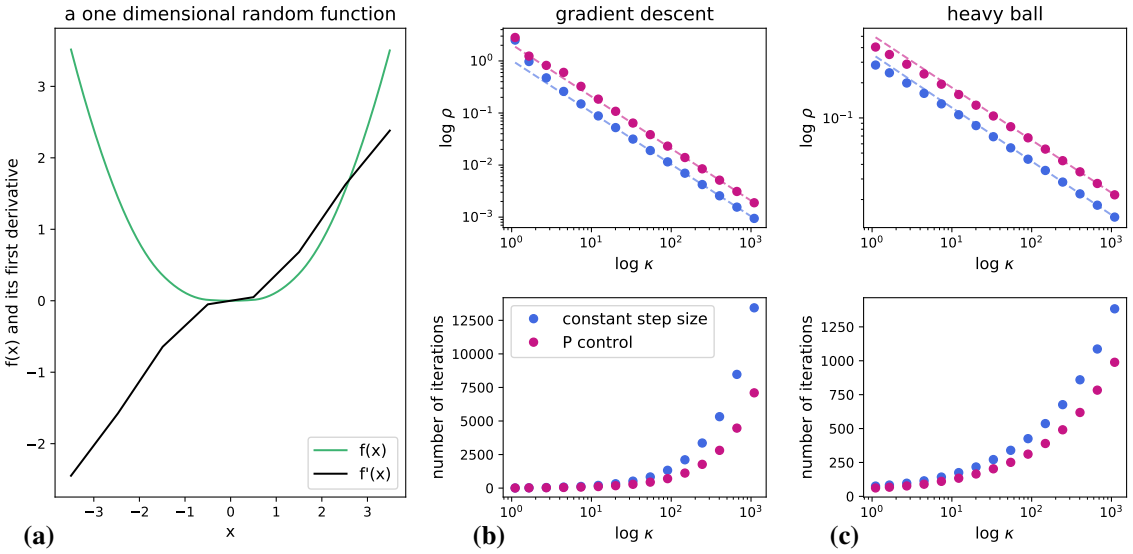

Figure 1: **(a)** A strongly convex function f with condition number 10 generated from a randomly sampled set of second derivatives. Note that $f(0) = 0$ and $f'(0) = 0$. **(b), (c) (Top panels)** Gradient descent and the heavy ball method with P control produce the same convergence behavior as the same algorithms with constant step size routines of 1 and 0.5, respectively. The dashed lines are linear fits of $\log \rho$ against $\log \kappa$ for large values of κ. **(b), (c) (Bottom panels)** For large condition numbers κ, gradient descent (heavy ball) with P control converges in a smaller number of iterations, up to a factor of 2 (1.4) less than the constant step size routine. r and θ are fixed at 0.5 and 0.01, respectively, for all κ.

We compare the performances of gradient descent and the heavy ball method with P control and with constant step sizes that achieve the relevant complexity lower bounds on random strongly convex functions. We find that P control is also able to achieve these bounds and can even improve the constant in many cases - see Fig. 1. In the following, we briefly describe these experiments

and their analysis. In Appendix B we present experiments on the nonconvex problem of principal components analysis.

We choose a range of condition numbers evenly tiling the log scale between 1.1 and 1100. For each value of κ, we generate fifty strongly convex functions $f : \mathbb{R}^{500} \to \mathbb{R}$. Details of how these functions are generated are given in Appendix C. Each function is optimized once using gradient descent with constant step size, and once with P control with the same initial step size as the constant step size method. The two methods begin from different random initializations $x^0 \sim 5 * \text{Unif}[0, 1)^{500}$. Momentum is initialized to the zero vector. Optimization halts when the norm of the difference between successive updates drops below 10^{-8}. We run P control with $\theta = 0.01$ and $r = 0.5$. Additional constraints on the controller are described in Appendix C.

We characterize the convergence of our algorithms using $||x_n|| \sim B||x_0||e^{\rho n}$, where B is a positive constant and ρ is the convergence rate, which depends on the condition number κ. To probe this dependency, for each trajectory, we perform a linear fit of the log norm of the iterate (scaled by the norm of the initial iterate) to the iteration number. The slope ρ produced by this fit is recorded, along with the number of iterations required to reach the stopping criterion. This data is graphed in the top panel of Fig. 1(b,c) as a function of condition number. We see that $\log \rho$ exhibits roughly linear behavior with $\log \kappa$. It is therefore natural to estimate the asymptotic dependence of ρ on κ for each algorithm with a linear fit to the data in Fig. (1)(b,c) (top panel) for large κ.

For gradient descent, the complexity lower bound for the convergence rate ρ (see Appendix C) is κ^{-1}, i.e., $\rho \sim \mathcal{O}(\kappa^{-1})$ [12]. A constant step size of 1 ($1/L$ with $L = 1$) achieves this bound. Linear fits estimating the asymptotic convergence rates of gradient descent with this constant step size and with P control return (slope, constant) tuples $(-0.999(1), 0.013(3))$ and $(-1.000(1), 0.317(3))$, respectively.[1] Both step size schemes achieve the lower bound. The larger constant produced by P control indicates convergence within a smaller number of iterations. See Fig. 1(b).

Similarly, continuous-time analysis of the heavy ball equation (8) establishes a complexity lower bound of $\kappa^{-0.5}$ for ρ [11]. The same holds true in discrete time for large κ [12]. Experiments indicate that both P control and a constant step size routine with step size $1/2$ perform close to this lower bound, with asymptotic convergence rates of $-0.450(2)$ and $-0.459(3)$, respectively. Once again, as with gradient descent, P control has a better constant ($-0.290(6)$ versus $-0.453(8)$), which manifests as a smaller number of iterations to convergence. See Fig. 1(c). The choice of 0.5 as the step size for the constant step size routine is motivated by the fact that the algorithm achieves a rate $\rho \sim 1/(2\sqrt{\kappa})$ on quadratic functions, and step sizes larger than $2(\sqrt{2} - 1) \approx 0.828$ lead to instability on quadratics.

4. Conclusion

We have shown how to construct an adaptive step size control mechanism for optimization rooted in ideas from the numerical analysis of differential equations. Our method applies to any algorithm with a well-defined continuous-time representation and has the advantage of being computationally cheap and easy to tune. Experiments on strongly convex functions indicate that for ill-conditioned problems our approach is able to reduce the number of iterations compared to the state of the art, and is competitive across all problem instances. Experiments on principal components analysis further highlight the potential of the approach even for nonconvex optimization problems.

1. Standard deviations are reported in brackets; for example, writing $0.013(3)$ indicates a parameter of 0.013 with a standard deviation of 0.003.

Acknowledgements

We thank the German Research Foundation and the Branco Weiss Fellowship, administered by ETH Zürich, for the generous support. We also thank the Google PhD Fellowship program, and the Office of Naval Research under grant number N00014-18-1-2764.

References

[1] Larry Armijo. Minimization of functions having Lipschitz continuous first partial derivatives. *Pacific Journal of Mathematics*, 16(1):1–3, 1966.

[2] Akshay Balsubramani, Sanjoy Dasgupta, and Yoav Freund. The fast convergence of incremental PCA. *Advances in Neural Information Processing Systems*, pages 3174–3182, 2013.

[3] Yair Carmon, John C. Duchi, Oliver Hinder, and Aaron Sidford. Lower bounds for finding stationary points II: first-order methods. *Mathematical Programming*, 2019. Preprint available online.

[4] Jelena Diakonikolas and Michael I. Jordan. Generalized momentum-based methods: A Hamiltonian perspective. *arXiv:1906.00436 [math.OC]*, pages 1–30, 2019.

[5] R. Fletcher. *Practical Methods of Optimization*. John Wiley & Sons, second edition, 1987.

[6] E. Hairer, S. P. Nørsett, and G. Wanner. *Solving Ordinary Differential Equations I*. Springer, second edition, 1993.

[7] E. Hairer, , and G. Wanner. *Solving Ordinary Differential Equations II*. Springer, second edition, 1996.

[8] Walid Krichene, Alexandre M. Bayen, and Peter L. Bartlett. Accelerated mirror descent in continuous and discrete time. *Advances in Neural Information Processing Systems 28*, pages 2845–2853, 2015.

[9] M. Muehlebach and M. I. Jordan. Optimization with momentum: Dynamical, control-theoretic, and symplectic perspectives. *Journal of Machine Learning Research*, 22:1–50, 2021.

[10] Michael Muehlebach and Michael I. Jordan. A dynamical systems perspective on Nesterov acceleration. *Proceedings of the International Conference on Machine Learning*, pages 1–7, 2019.

[11] Michael Muehlebach and Michael I. Jordan. Continuous-time lower bounds for gradient-based algorithms. *Proceedings of the International Conference on Machine Learning*, pages 1–13, 2020.

[12] Yurii Nesterov. *Introductory Lectures on Convex Optimization*. Springer, 2004.

[13] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer Science and Business Media, second edition, 2006.

[14] Erkki Oja and Juha Karhunen. On stochastic approximation of the eigenvectors and eigenvalues of the expectation of a random matrix. *Journal of Mathematical Analysis and Applications*, 106, 1985.

[15] Weijie Su, Stephen Boyd, and Emmanuel J. Candès. A differential equation for modeling Nesterov's accelerated gradient method: Theory and insights. *Journal of Machine Learning Research*, 17(153):1–43, 2016.

[16] Andre Wibisono, Ashia C. Wilson, and Michael I. Jordan. A variational perspective on accelerated methods in optimization. *Proceedings of the National Academy of Sciences*, 113(47): E7351–E7358, 2016.

[17] Philip Wolfe. Convergence conditions for ascent methods. *SIAM Review*, 11(2):226–235, 1969.

Appendix A. P Control Hyperparameters

In Fig. A.1 we illustrate how the choice of θ at fixed r changes the behavior of the proportional controller (7).

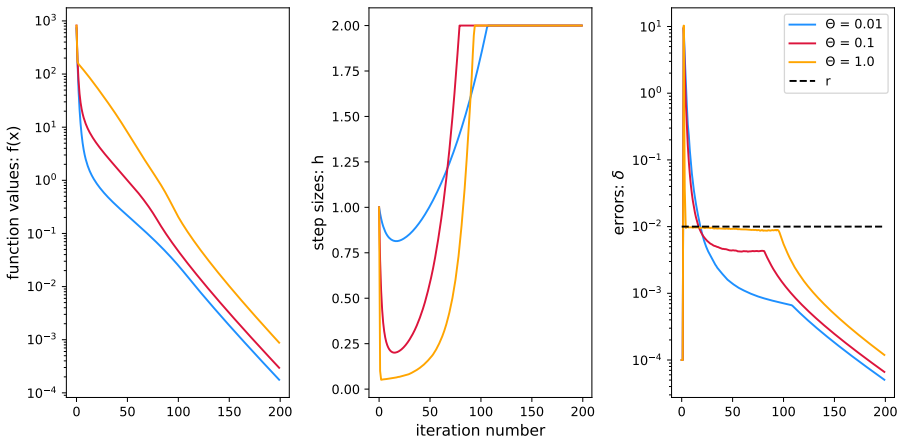

Figure A.1: **Gradient descent with adaptive P control.** Example trajectories of the function value (left panel), the step size (middle panel), and the error δ_n (right panel) for gradient descent with step sizes set using proportional control are shown for three different values of θ. Here, $f : \mathbb{R}^{500} \to \mathbb{R}$ is a randomly generated strongly convex function with a condition number of 100. See Appendix C for a description of how the function f is generated. Each value of θ produces a different trajectory. After a few tens of iterations, the control mechanism is able to force δ_n below the specified value of r. Large values of θ produce the largest differences between successive step sizes and vice versa.

Appendix B. A Nonconvex Problem: PCA

We demonstrate the use of P control to set step sizes for a simple instance of the PCA problem: given a positive semidefinite symmetric matrix $A \in \mathbb{R}^{d\times d}$ with distinct eigenvalues, the goal is to learn its first principal component (eigenvector) . The corresponding optimization problem

$$\underset{||x||=1}{\operatorname{argmin}} \; -x^\top A x, \tag{11}$$

where $x \in \mathbb{R}^d$, is nonconvex but still admits a unique computable solution.

A well-established algorithm that solves (11) is Oja's rule [14], given by the update

$$x_{n+1} = \frac{(I + h_n A)x_n}{||(I + h_n A)x_n||}, \tag{12}$$

where I is the identity matrix in d dimensions and h_n is the n^{th} step size.

Let the eigengap of A, gap(A), be defined as the difference between the largest and second-largest eigenvalues of A. Define a positive constant c. A recent convergence proof guarantees that the error of (12), measured by the quantity $1 - (x_n^\top x^*)^2$ where x^* is the principal eigenvector of

A, is $\mathcal{O}(1/n)$ for a step size routine of c/n if $c \geq 1/(2 * \text{gap}(A))$ [2]. However, in most real-world problems, gap(A) would not normally be known ahead of time, complicating the choice of c. Indeed, in [2], this is posed as a problem for practitioners. We are able to offer a solution in the form of P control, which avoids the problem entirely because it requires no knowledge of the gap. We compare Oja's rule with a c/n step size scheme with different values of c to P control (with fixed r and θ) across a range of eigengaps and find that the latter performs competitively.

We note that the convergence result of Balsubramani et al. [2] is proved in the streaming setting, where instead of a fixed matrix A we have a collection of vectors y_i, sampled i.i.d. from a mean-zero distribution with covariance A, that we access one at a time. In such a scenario, A is replaced by the rank one matrices $y_n y_n^\top$ in (12). Since our goal here is simply to evaluate the adaptive step size routine on a nonconvex optimization problem, we confine ourselves to the simplest possible, and hence deterministic, setting even though P control could also be applied in the streaming setting.

P control for Oja's rule. The continuous-time system corresponding to Oja's rule has a particularly simple form. It is gradient flow on the unit sphere in $d-1$ dimensions, which we can write as the differential equation

$$\dot{x}(t) = \left(A - \frac{x(t)^\top A x(t)}{x(t)^\top x(t)} I \right) x(t). \tag{13}$$

We note that the original convergence proof of (12) was done in continuous time by analyzing the behavior of (13) [14].

It is not difficult to see that the local error of (12) is quadratic in h_n (see the discussion around (3)). In order to compute δ_n for (7), we need a more accurate discretization of (13) than (12). We use a Heun update $x^{\text{H}}_{n+1} = x_n + 0.5 h_n (g(x_n) + g(x_{n+1}))$, where g is given by the right-hand side of (13) and x_{n+1} is given by (12). Then $\delta_n = ||x_{n+1} - x^{\text{H}}_{n+1}||$, as before. To calculate the step size for Oja's rule, we use this value of δ_n in the P controller (7) at each iteration.

As before, the Heun method allows us to reuse the computation of x_{n+1} at the next iteration. Hence the adaptive step size routine does not alter the computational cost per iteration.

We interrogate a range of eigengaps between 10^{-5} and 10^{-1}. At each value of the eigengap we generate twenty diagonal matrices $A \in \mathbb{R}^{100 \times 100}$, and apply random orthogonal transformations to introduce off-diagonal elements. On each matrix, we run Oja's rule (12) four times: once with P control, and thrice with a c/n step size routine with $c = 0.5/\text{gap}(A), 5/\text{gap}(A)$, and $50/\text{gap}(A)$. We parametrize the P controller with $(r, \theta) = (10^{-4}, 10^{-4})$. Iterates are initialized from the standard uniform distribution. The stop criterion is either $||x_{n+1} - x_n|| \leq 10^{-8}$ or $n = 2.5 \times 10^5$, whichever occurs first.

Results. The P control mechanism produces an algorithm that is competitive with a c/n step size routine for moderate and large values of c, and that outperforms the smallest allowed value of c, as measured by the number of iterations required to converge (see Fig. B.2). As mentioned previously, the utility of this result derives from the fact that the eigengap is not normally known ahead of time, making it difficult in practice to set a good value of c for the c/n step size routine such that the algorithm achieves its $\mathcal{O}(1/n)$ rate of convergence.

Thus we see that P control can readily be adapted to problems that are not strongly convex, and can produce algorithms that perform well compared to the state of the art.

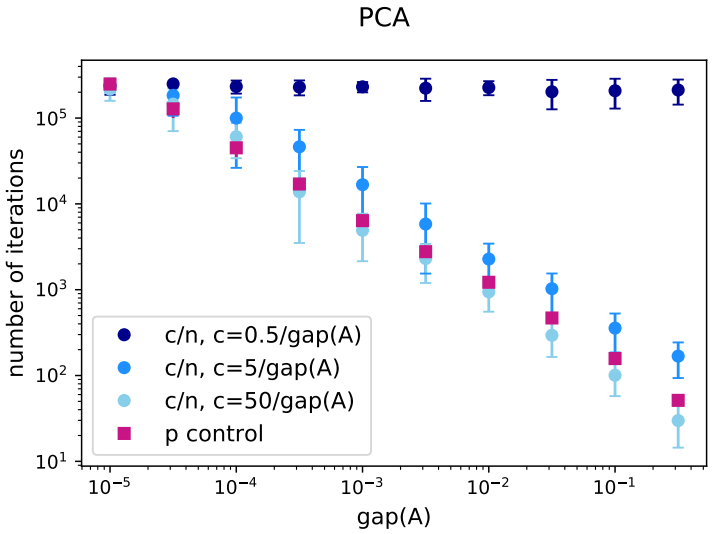

Figure B.2: **Oja's rule with P control performs competitively with a $1/n$ step size routine.** Setting step sizes in Oja's rule using P control with r and θ both set to 10^{-4} produces an algorithm that converges, on average, in about as many iterations as a $1/n$ step size routine with a moderate to large premultiplier c. On the y-axis we have the mean number of iterations to convergence over twenty repeats, and on the x-axis we have gap(A), the difference between the two largest eigenvalues of the matrix A. Error bars indicate one standard deviation. The error bars for P control are too small for visibility. See text for experimental details.

Appendix C. Methods

These are some additional details of the methods for the experiments presented in Fig. 1.

C.1. Random generation of strongly convex functions

We divide the real line into a finite number N of discrete intervals. For a given value of the condition number κ, we sample a set of N numbers $\kappa^{-1} + z(1 - \kappa^{-1})$, where $z \sim \text{Unif}[0, 1)$, that serve as the piecewise-constant second derivatives of a strongly convex function. The function is then constructed numerically from the second derivative and the conditions $f(0) = 0$, $f'(0) = 0$. Multidimensional functions $f : \mathbb{R}^d \to \mathbb{R}$ are obtained by summing d one-dimensional functions. We ensure that at $x^* = (0, ..., 0)$, the smallest eigenvalue of the Hessian of f is $1/\kappa$, and the largest eigenvalue is 1 (recall that we fix $L = 1$). An example of such a function is given in Fig. 1(a) with $N = 7$ and $\kappa = 10$. We fix $N = 7$ in all our experiments.

C.2. P control details

P control is run with $\theta = 0.01$ and $r = 0.5$. We impose the following two additional constraints that are slightly different for gradient descent and the heavy ball method.

Gradient descent. If the factor $(r/\delta_n)^{\theta/2}$ in (7) is ≥ 10 or ≤ 0.1, we set it equal to 10 or 0.1, respectively. If, furthermore, $h_n \geq 2$ or ≤ 0.01, we set it equal to 2 or 0.01, respectively. The upper

bound of 2 on the step size is motivated by the fact that this is the stability boundary for gradient descent on a 1-Lipschitz convex function. We note that it is possible to remove these constraints by fine-tuning r and θ to specific ranges of condition numbers, but too much tuning defeats the purpose of the adaptive algorithm, so we feel it is reasonable to introduce the constraints. We are then able to fix r and θ for all κ in the range 1.1 to 1100.

Heavy ball method. If the factor $(r/\delta_n)^{\theta/2}$ is ≥ 5 or ≤ 0.05, we set it equal to 5 or 0.05, respectively. And if $h_n \geq 0.8$ or ≤ 0.01, we set it equal to 0.8 or 0.01, respectively. The upper bound of 0.8 on the step size is motivated by the fact that this is roughly the stability boundary for the heavy ball algorithm on quadratic functions.