

An Investigation into the Silver Stepsize Schedule

A special topic for Mathematical Modelling and Scientific Computing in Optimisation for Data Science

George Hildick-Smith

April 25, 2026

© 2026 George Hildick-Smith. This work is licensed under CC BY-NC-ND 4.0, https://creativecommons.org/licenses/by-nc-nd/4.0/. Third party images and quotations remain the property of their owners.

1 Introduction

Gradient descent is one of the most widely used algorithms in modern optimisation, underpinning a vast range of applications in data science, for example training large-scale machine learning models. At its core, the algorithm is simple: at each iteration, move in the steepest descent direction. The key design choice is the *stepsize* — how large a step to take at each iteration. This seemingly simple choice has a profound effect on how quickly the algorithm converges, and choosing it well is the central question of this report.

Consider a μ-strongly convex L-smooth function with condition number $\kappa := L/\mu$: the optimal constant stepsize gives a convergence rate of $\mathcal{O}(\kappa)$ [1]; roughly κ gradient evaluations are needed to reduce the error by a constant factor. For well-conditioned problems this is acceptable, but for ill-conditioned problems this is very slow.

The classical remedy is to add *momentum* or compute *higher derivatives*. However, higher derivatives may not be available or may be computationally expensive. It is also possible that we are in a setting where we do not wish to store the history of previous iterates, and hence we restrict ourselves to momentum-free methods. The Chebyshev stepsize schedule achieves a convergence rate of $\mathcal{O}(\sqrt{\kappa})$ without momentum or higher derivatives, but only for quadratic objectives and only when the horizon n is fixed in advance. Hence, a natural question arises: *can a momentum-free, first-order method, using only gradient information and a clever choice of stepsizes, do strictly better than constant-stepsize gradient descent (GD) on smooth, strongly convex functions?*

This question was answered affirmatively by Altschuler and Parrilo [2], who introduced the Silver Stepsize Schedule (SSS) — a non-monotonic, fractal-like stepsize schedule that achieves a convergence rate of $\mathcal{O}(\kappa^{\log_\rho 2}) \approx \mathcal{O}(\kappa^{0.7864})$, where $\rho = 1 + \sqrt{2}$ is the silver ratio, hence its name. This is strictly better than constant-stepsize GD, resolving a problem that had been open for decades. The schedule is constructed recursively by *hedging* between short and long stepsizes at every scale of the recursion, distributing the roots of the associated error polynomial across the spectral interval in a way that is provably better than any constant choice.

In this project, we explore SSS through numerical experiments, leading us to formalise the Extended Silver Stepsize Schedule (ESSS) for arbitrary iteration budgets. The underlying idea of running Silver schedules for the binary expansion of N was briefly suggested in [2], but not developed further. Our contribution is to give an explicit construction and provide numerical evidence that for a given iteration budget, ESSS produces an iterate closer to the optimum than SSS.

2 First-Order Optimisation Methods

Here we review the classical acceleration techniques for methods that only use information given by the function $f(x)$ and $\nabla f(x)$. Namely, these are first-order methods, as opposed to second-order methods or higher that use, for example, the Hessian $\nabla^2 f(x)$. We investigate their differences, convergence rates, and applicable settings.

One may ask why we would focus on methods that use less information than potentially available to us when more powerful methods exist. The answer is that first-order methods can scale much better to large problems. For example, machine learning models often have millions of parameters. Computing the Hessian matrix has a time complexity of $\mathcal{O}(d^3)$ and to store it, a space complexity of $\mathcal{O}(d^2)$ and [3], where d is the number of parameters, so these higher-order techniques become impractical. In contrast, first-order methods have a complexity of $\mathcal{O}(d)$ for both space and time, making them much more practical for large-scale problems. As the amount of data used in modern applications continues to grow, we must develop more efficient tools. This motivates our study of first-order methods and their variations.

2.1 Problem Setup

Consider the quadratic objective function,

$$f(x) = \tfrac{1}{2}\, x^\top A x, \tag{1}$$

where A is symmetric positive definite (SPD) with eigenvalues contained in $[\mu, L]$, $0 < \mu \leq L$. This means f is μ-strongly convex and L-smooth, with unique minimiser $x^* = 0$. The **condition number** is $\kappa := \frac{L}{\mu} \geq 1$. Unless otherwise stated, this will be the problem setting for the rest of this report.

Remark. We have no linear term '$+b^\top x$' in Equation (1) as it can be removed by a simple change of variables. If x^* is the minimiser of $g(x) = \frac{1}{2}x^\top A x - b^\top x$, such that $Ax^* = b$ and using $y = x - x^*$, we get

$$g(x) = \tfrac{1}{2} y^\top A y - b^\top x^* + \tfrac{1}{2}(x^*)^\top A x^* = f(y) - \tfrac{1}{2}(x^*)^\top b.$$

Since the last term is constant, we obtain

$$\arg\min_x\ g(x) = x^* + \arg\min_y\ f(y).$$

Hence, solving g is equivalent to solving f up to a shift by x^*.

2.2 Classic Gradient Descent

The GD update with a fixed *stepsize* α is

$$x_{k+1} = x_k - \alpha \nabla f(x_k). \tag{2}$$

On the quadratic $f(x) = \frac{1}{2} x^\top A x$ where $\nabla f(x) = Ax$, the update becomes

$$x_{k+1} = x_k - \alpha A x_k = (I - \alpha A)\, x_k. \tag{3}$$

2.2.1 The Error and Convergence Rate

As $x^* = 0$, the error is $e_k := x_k - x^* = x_k$, so after n steps $e_n = (I - \alpha A)^n e_0$. Taking the spectral norm $\|M\| = \sup_{x \neq 0} \|Mx\|/\|x\|$,

$$\|e_n\| \leq \|(I - \alpha A)\|^n \, \|e_0\|. \tag{4}$$

Since A is SPD with eigenvalues $\lambda_i \in [\mu, L]$, the matrix $I - \alpha A$ is symmetric with eigenvalues $\{1 - \alpha\lambda_i\}$. Hence, $\|(I - \alpha A)\| = \rho(\alpha) := \max_{\lambda \in [\mu,\, L]} |1 - \alpha\lambda|$. Note that μ is the strong convexity parameter and L is the smoothness parameter of f.

For the error to decrease after each iteration, we require the *convergence rate* $\rho(\alpha)$ to be less than 1. The smaller $\rho(\alpha)$ is, the faster the convergence. Now we can formulate a second optimisation problem where we want to minimise $\rho(\alpha)$ over $\alpha > 0$. This is the problem of optimising the stepsize to achieve the best convergence.

2.2.2 Optimising the Convergence Rate

As ρ is piecewise linear, its maximum over $[\mu, L]$ is attained at an interval endpoint so,

$$\begin{aligned} \rho(\alpha) &= \max\big\{|1 - \alpha\mu|,\ |1 - \alpha L|\big\} \\ &= \max\big\{1 - \alpha\mu,\ \alpha L - 1\big\}. \end{aligned} \tag{5}$$

Theorem 2.1 (Optimal constant stepsize [1, 4])**.** *We make two claims:*

(i) that $\alpha^* = \dfrac{2}{L + \mu}$ *uniquely minimises* $\rho(\alpha)$ *over all* $\alpha > 0$*, and*

(ii) that $\rho(\alpha^*) = \dfrac{\kappa - 1}{\kappa + 1}$.

Proof. We verify both claims.

(i) For the optimal α, we see that convergence requires $\rho(\alpha) \leq 1$, i.e. $0 \leq \alpha \leq 2/L$, and ρ is minimised when $1 - \alpha\mu = \alpha L - 1$ as shown in Figure 1,

$$1 - \alpha\mu = \alpha L - 1 \implies 2 = \alpha(L + \mu) \implies \alpha^* = \frac{2}{L + \mu}.$$

(ii) Substituting α^* in, $\rho(\alpha^*) = 1 - \alpha^*\mu = \alpha^* L - 1 = \dfrac{L - \mu}{L + \mu} = \dfrac{\kappa - 1}{\kappa + 1}$.

□

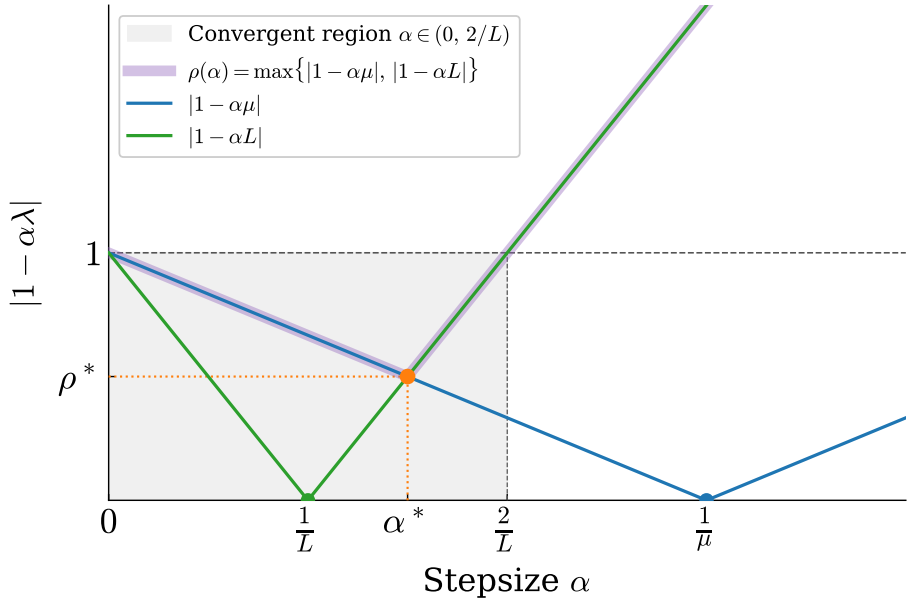


Figure 1: *The convergence rate* $\rho(\alpha) = \max_{\lambda\in[\mu,L]} |1-\alpha\lambda|$ *as a function of the stepsize* α. *The convergent region is shaded and the optimal stepsize* α^* *is indicated.*

For an animation of how the error and trajectories evolve with α, see this video[1].

2.2.3 Iteration Complexity

Combining Equation (4) and Theorem 2.1 gives the main convergence result for constant-stepsize GD,

$$\|e_n\| \le \left(\frac{\kappa-1}{\kappa+1}\right)^n \|e_0\| = D_n\|e_0\|. \tag{6}$$

Although derived here on the quadratic for simplicity, this rate holds for any μ-strongly convex, L-smooth function with stepsize $\alpha^* = 2/(L+\mu)$ [4]. Consequently, to achieve $\|e_n\| \le \varepsilon\,\|e_0\|$, using $\log(\frac{\kappa-1}{\kappa+1}) \le -\frac{2}{\kappa+1}$ it suffices to take

$$n \ge \frac{\kappa+1}{2}\log\frac{1}{\varepsilon} \tag{7}$$

iterations, so the iteration count is $\mathcal{O}(\kappa\log 1/\varepsilon)$ for GD[2].

Remark (The fundamental limitation of constant stepsizes)**.** The convergence rate $\rho = (\kappa-1)/(\kappa+1) \to 1$ as $\kappa\to\infty$, meaning the method slows down for ill-conditioned problems as the error contracts by a factor close to 1 at each step.

[1] https://youtu.be/ZQWikG4XRXM

[2] On smooth convex functions without strong convexity ($\mu = 0$), GD with stepsize $\alpha = 1/L$ still converges, but at the slower sublinear rate $f(x_k) - f(x^*) \le L\|x_0 - x^*\|^2/2k$, i.e. $\mathcal{O}(1/\varepsilon)$ iterations required. The $\mathcal{O}(\kappa\log 1/\varepsilon)$ linear rate stated in Table 1 requires $\mu > 0$.

2.3 Gradient Descent with Variable Stepsizes

The GD update with step-dependent stepsizes $\alpha_k > 0$ is

$$x_{k+1} = x_k - \alpha_k \nabla f(x_k) = (I - \alpha_k A)\, x_k. \tag{8}$$

From Equation (1), the error is now $e_n = P_n(A)\, e_0$, where $P_n(A) = \prod_{k=0}^{n-1}(I - \alpha_k A)$. Noting that A is SPD and hence has an eigendecomposition $A = Q\Lambda Q^\top$,

$$P_n(A) = \prod_{k=0}^{n-1}(I - \alpha_k Q\Lambda Q^\top) = Q\left(\prod_{k=0}^{n-1}(I - \alpha_k \Lambda)\right) Q^\top = Q P_n(\Lambda) Q^\top. \tag{9}$$

Here the error can be expressed as

$$\|e_n\| = \|P_n(\Lambda)\, Q^\top e_0\| \leq \max_{\lambda \in [\mu, L]} |P_n(\lambda)| \cdot \|e_0\|. \tag{10}$$

The inequality uses the fact that Λ is diagonal and hence so is $P_n(\Lambda)$, and the spectral norm of a diagonal matrix is bounded by the maximum absolute value of its entries. $P_n(\lambda) = \prod_{k=0}^{n-1}(1 - \alpha_k \lambda)$ is a scalar polynomial of degree n in λ.

Remark. Equation (10) reduces the stepsize design problem to a polynomial approximation problem: *find degree-n polynomials P_n with $P_n(0) = 1$ that are as small as possible on $[\mu, L]$* to minimise the error. Each choice of $\{\alpha_k\}_{k=0}^{n-1}$ corresponds to a unique such polynomial $P_n(\lambda) = \prod_{k=0}^{n-1}(1 - \alpha_k \lambda)$, with roots $\{\lambda_k = 1/\alpha_k\}$ [2]. An example of such a polynomial is shown in Figure 2.

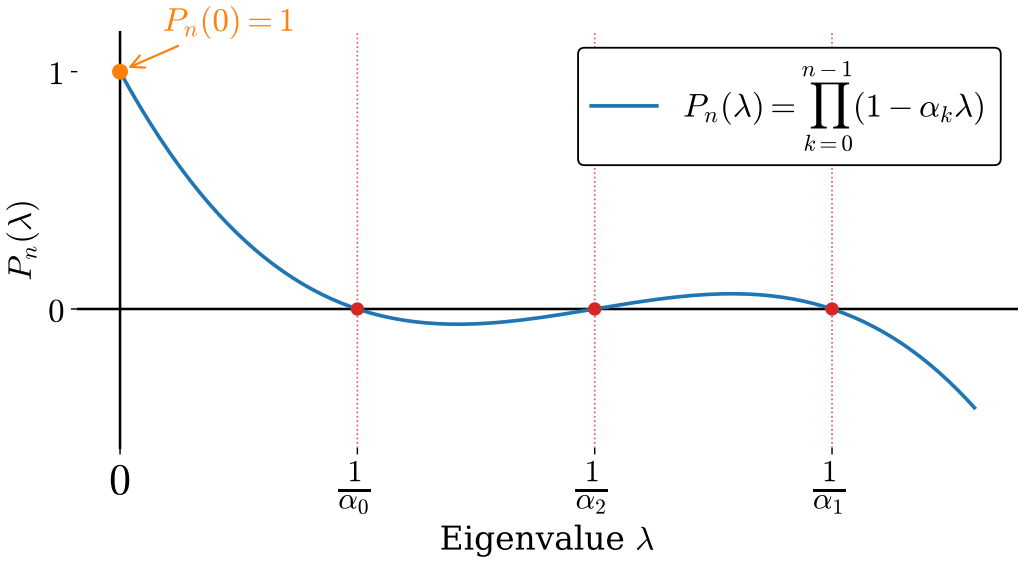


Figure 2: *The polynomial $P_n(\lambda)$ for a particular choice of $\{\alpha_k\}_{k=0}^{n-1}$ stepsizes*[3].

[3]Note that there is no need to have $\{\alpha_k\}_{k=0}^{n-1}$ in any particular order, monotonic or not.

2.4 Polyak Heavy Ball

Pure gradient descent discards the history of iterates: each step depends only on the current gradient. The *Polyak Heavy Ball* method adds a *momentum* term that incorporates information from the previous iterate, analogous to a physical ball rolling down a surface and accumulating inertia [5]. The update rule is

$$x_{k+1} = x_k - \alpha \nabla f(x_k) + \beta(x_k - x_{k-1}), \tag{11}$$

where $\alpha > 0$ is the stepsize, x_0 is the initial point, and $\beta \in [0, 1)$ is the *momentum coefficient*. Setting $\beta = 0$ recovers constant-stepsize GD.

2.4.1 Optimal Parameters and Convergence on Quadratics

On the quadratic objective, Equation (1), with SPD matrix A and spectrum in $[\mu, L]$, the update Equation (11) admits a closed-form optimal parameter pair. Setting

$$\alpha^* = \frac{4}{(\sqrt{L} + \sqrt{\mu})^2}, \qquad \beta^* = \left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1}\right)^2, \tag{12}$$

the error contracts as

$$\|e_n\| \leq C\left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1}\right)^n \|e_0\|, \tag{13}$$

for some constant $C > 0$. This is a genuine improvement over the $\mathcal{O}(\kappa \log 1/\varepsilon)$ rate of constant-stepsize GD. Hence, the heavy ball method has an iteration complexity of $\mathcal{O}(\sqrt{\kappa} \log 1/\varepsilon)$. The proof of Equation (12) and Equation (13) is found in [5].

2.4.2 Limitations Beyond the Quadratic Setting

Despite its strong performance on quadratics, the Heavy Ball method has a well-documented fragility: with the parameters Equation (12) it is known to *diverge* on certain smooth, strongly convex functions that are not quadratic [6]. This is because the optimal momentum coefficient is tuned to the global curvature of the quadratic, and cannot adapt to local variations in curvature in general functions.

Remark. Interpreting the Heavy Ball update Equation (11) in the polynomial framework of Section 2.3, the error after n steps satisfies $e_n = Q_n(A)\, e_0$ for a *degree-n polynomial* $Q_n(\lambda)$. Here $Q_n(\lambda) = (1 - \alpha\lambda)Q_{n-1}(\lambda) - \beta(Q_{n-1}(\lambda) - Q_{n-2}(\lambda))$, with $Q_0(\lambda) = 1$ and $Q_1(\lambda) = 1 - \alpha\lambda$. This couples successive steps and can amplify errors when the quadratic assumption is violated.

2.5 Nesterov Accelerated Gradient

The Nesterov accelerated gradient (NAG) method [7] also uses a *momentum* term to accelerate convergence. Unlike GD, which takes each step from the current iterate x_k, NAG first extrapolates to an intermediate point y_k and then takes the gradient step from there. The update is

$$y_k = x_k + \frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1}(x_k - x_{k-1}), \tag{14}$$

$$x_{k+1} = y_k - \frac{1}{L}\nabla f(y_k). \tag{15}$$

Equation (14) forms the *momentum step*: it extrapolates beyond x_k in the direction of the previous displacement $x_k - x_{k-1}$. Equation (15) then takes a standard gradient step of size $1/L$ from the extrapolated point y_k.

2.5.1 Convergence Rate

From [1], NAG achieves the convergence bound

$$\|e_n\| \leq 2\left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1}\right)^n \|e_0\|, \tag{16}$$

giving an iteration count of $\mathcal{O}(\sqrt{\kappa}\log 1/\varepsilon)$ to reach $\|e_n\| \leq \varepsilon \|e_0\|$. Crucially, the bound in Equation (16) holds for all smooth, strongly convex f [1], not just quadratics[4].

2.5.2 Optimality

Nesterov established a matching lower bound showing that *no* first-order method can achieve fewer than $\Omega(\sqrt{\kappa}\log 1/\varepsilon)$ gradient evaluations on the class of μ-strongly convex L-smooth functions [1, 7]. NAG is therefore *optimal* among first-order methods.

Contrast with Heavy Ball

Both NAG and Heavy Ball use momentum and achieve the same asymptotic rate $\mathcal{O}(\sqrt{\kappa}\log 1/\varepsilon)$ on quadratics. The key difference lies in robustness. Heavy Ball uses *fixed* parameters α, β tuned to the global spectrum $[\mu, L]$. NAG's momentum coefficient is also fixed, but the gradient is evaluated at the extrapolated point y_k rather than x_k; this subtle change provides a global descent guarantee that Heavy Ball lacks [1].

[4]The momentum coefficient $(\sqrt{\kappa}-1)/(\sqrt{\kappa}+1)$ requires knowing μ in advance, which is unavailable when either the problem is only smooth convex ($\mu = 0$) or $\mu > 0$ but cannot be computed without forming the full Hessian. So, Nesterov proposed a parameter-free variant [7] that uses the time-varying coefficient $k/(k+3)$ (where k is the iteration number), requiring no knowledge of μ, and achieves the sublinear rate $\mathcal{O}(1/k^2)$ on smooth convex functions. For smooth, strongly convex functions the $\mathcal{O}(\sqrt{\kappa})$ rate is recovered by combining this with periodic restarts [8], where the method is restarted from the current iterate whenever a heuristic criterion (such as a rise in function value) is triggered.

2.6 Chebyshev Stepsize Schedule

The framework of Section 2.3 reduces stepsize design to a minimax problem: find a degree-n polynomial P_n with $P_n(0) = 1$ that minimises $\max_{\lambda\in[\mu,L]} |P_n(\lambda)|$. The classical solution to this problem is the *Chebyshev polynomials* [9]. The Chebyshev polynomial of the first kind, T_n, is defined by $T_n(\cos\theta) = \cos(n\theta)$ [9]. Shifting and scaling to the interval $[\mu, L]$ and enforcing $P_n(0) = 1$ gives

$$P_n(\lambda) = \frac{T_n\left(\dfrac{L+\mu-2\lambda}{L-\mu}\right)}{T_n\left(\dfrac{L+\mu}{L-\mu}\right)}, \tag{17}$$

which achieves the minimax bound given in Equation (10)

$$\max_{\lambda\in[\mu,L]} |P_n(\lambda)| \le 2\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^n, \tag{18}$$

from [9], giving an iteration complexity of $\mathcal{O}(\sqrt{\kappa}\log 1/\varepsilon)$. This optimality was first established by Young [10] for the problem of solving linear systems, and later connected to gradient descent on quadratics by Flanders and Shortley [11]. The stepsizes $\{\alpha_k\}$ are read off from the roots of P_n via $\alpha_k = 1/\lambda_k$, where the roots are given by [3] as

$$\alpha_k = \left[\frac{L+\mu}{2} + \frac{L-\mu}{2}\cos\left(\frac{2k+1}{2n}\pi\right)\right]^{-1}, \qquad k = 0, 1, \ldots, n-1. \tag{19}$$

The stepsizes are non-monotone and **depend on n in advance** but can be taken in any order. In Figure 3 we visualise the Chebyshev polynomial $P_n(\lambda)$, which oscillates on $[\mu, L]$. We see that more roots tend toward the endpoints of the spectral interval [9], which is precisely what minimises the maximum value of $|P_n(\lambda)|$ on $[\mu, L]$.

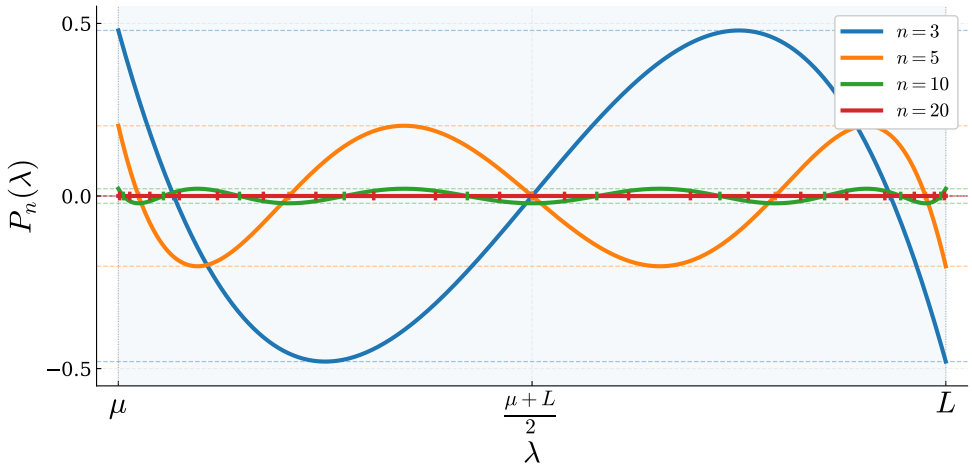


Figure 3: *The Chebyshev polynomials $P_n(\lambda)$ with $\mu = 1$ and $L = 20$ with bounds shown.*

2.7 Summary of Methods

We summarise the methods seen so far in Table 1. We have seen that GD with a constant stepsize is the simplest and slowest method in the general smooth convex setting; adding momentum (Heavy Ball, NAG) or a carefully designed stepsize schedule (Chebyshev) each achieves the optimal $\mathcal{O}(\sqrt{\kappa})$ rate for first-order methods [1].

Method	**Rate**	**Requires**	**Momentum**	**Setting**
GD (constant α)	$\mathcal{O}(\kappa)$	μ, L	No	Smooth Strongly Convex
Heavy Ball	$\mathcal{O}(\sqrt{\kappa})$	μ, L	Yes	Quadratic
NAG	$\mathcal{O}(\sqrt{\kappa})$	μ, L	Yes	Smooth Strongly Convex
Chebyshev	$\mathcal{O}(\sqrt{\kappa})$	μ, L, n fixed	No	Quadratic

Table 1: *Summary of first-order methods analysed. All rates omit a* $\log(1/\varepsilon)$ *factor.*

Both Heavy Ball and the Chebyshev schedule work on quadratics, but neither extends well beyond this setting. Heavy Ball can diverge on general smooth, strongly convex functions [6], and the Chebyshev schedule is restricted to quadratics [10] — it requires knowing both μ and L in advance to compute the stepsizes from Equation (19), and provides no convergence guarantee outside the quadratic setting. NAG, by contrast, achieves the same rate while remaining valid for all smooth, strongly convex objectives, making it the only method in Table 1 that is both optimal and general. We compare how each method behaves in Figure 4.

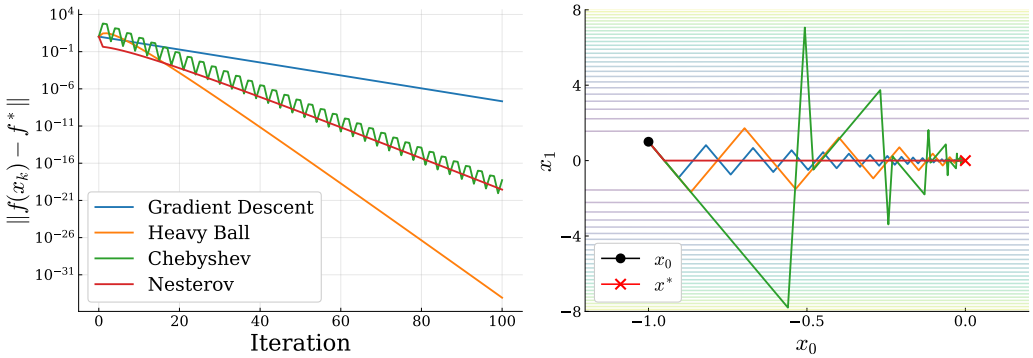


Figure 4: *Comparison of convergence rates for the methods*[5] *in Table 1 on a smooth, strongly convex function* $f(x) = \frac{1}{2}x_0^2 + 10x_1^2$ *with* $\kappa = 20$*. On the left we have the absolute error in function value over iterations and on the right the trajectories in the* (x_0, x_1) *plane.*

[5]Note that the parameters are chosen to better visualise the methods, and hence are not optimal.

3 The Silver Stepsize Schedule

All three **accelerated** methods achieve the same asymptotic rate $\mathcal{O}(\sqrt{\kappa})$, which is optimal for smooth, strongly convex functions by Nesterov's lower bound [7]. Analysing Table 1 we may ask: *can a momentum-free method do strictly better than constant-stepsize GD on smooth, strongly convex functions, and if so, how?*

3.1 The Idea Behind the Silver Stepsize Schedule

To answer this question, we introduce the SSS, a novel method developed by Altschuler and Parrilo [2] that creates a non-monotonic, fractal-like pattern of stepsizes. This new method has a convergence rate of $\mathcal{O}(\kappa^{\log_\rho 2}) \approx \mathcal{O}(\kappa^{0.7864})$, where $\rho = 1 + \sqrt{2}$, which is an improvement over constant-stepsize GD but not the optimal lower bound of $\mathcal{O}(\sqrt{\kappa})$ as shown in Figure 5. The method used to generate these stepsizes is called *hedging*. The idea is to alternate between small stepsizes that reduce the error in the worst-case direction (the eigenvector corresponding to the largest eigenvalue L) and large stepsizes that reduce the error in the best-case direction (the eigenvector corresponding to the smallest eigenvalue μ). However, these stepsizes are adversarial to each other, meaning that the worst-case stepsize for the largest eigenvalue is the best-case stepsize for the smallest eigenvalue, and vice versa. This is why a constant stepsize can only perform so well: it must compromise between the two.

Recall from Section 2.3 that choosing a stepsize schedule $\{\alpha_k\}$ is equivalent to choosing the roots of the polynomial $P_n(\lambda) = \prod_{k=0}^{n-1}(1 - \alpha_k\lambda)$, with roots $\lambda_k = 1/\alpha_k$. A small stepsize places a root near L, suppressing the error in that direction, while a large stepsize places a root near μ. This is exactly the hedging idea: distributing roots across both ends of the spectral interval so that no part of the spectrum is left unaddressed over the full n-step block. The challenge is in choosing which roots to place where, so that P_n remains small across the *entire* interval simultaneously — not just at the roots. It is this global consistency requirement that the Silver construction resolves through its recursive structure, as we now describe.

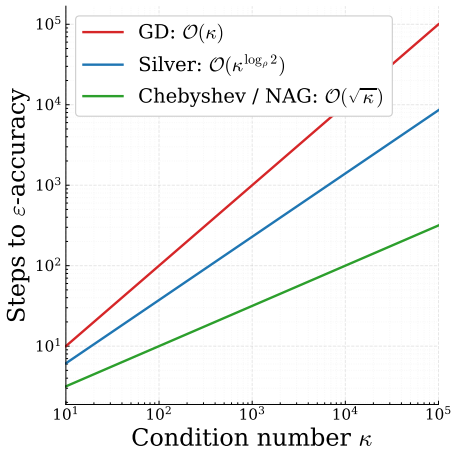


Figure 5: *Comparison of iteration counts as a function of condition number κ.*

3.2 The Algorithm

The exact details and proof of the Silver construction are in [2], hence we only give a sketch of the key ideas. First, we normalise the problem so that the spectral interval is $[\mu/L, 1]$ and $\kappa = L/\mu$. Then we construct a sequence of auxiliary stepsizes y_n, z_n where $y_1 = z_1 = 1/\kappa$. The stepsizes are computed iteratively by Equation (20) and the cobweb plot in Figure 6 visualises the recursive construction. To calculate the auxiliary stepsizes (from [2]) we solve iteratively from $n = 1$ to $n = 2^k$ for $k = 0, 1, \ldots,$

$$y_{2n} z_{2n} = z_n^2, \quad z_{2n} - y_{2n} = 2(z_n - z_n^2).$$

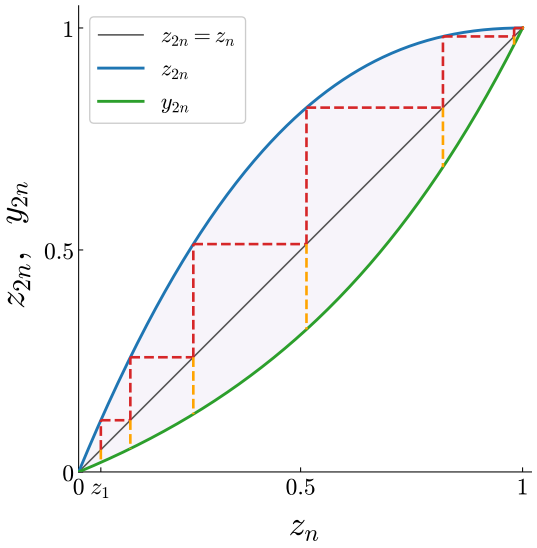


Figure 6: *The cobweb plot for the recursive construction of y_{2n}, z_{2n} from z_n.*

Letting $\xi = 1 - z_n$, they have recursive solutions,

$$y_{2n} = z_n / \left(\xi + \sqrt{1 + \xi^2}\right), \qquad z_{2n} = z_n \left(\xi + \sqrt{1 + \xi^2}\right). \tag{20}$$

Theorem 3.1 (Splitting Inequality)**.** *We claim that $y_{2n} < z_n < z_{2n}$ for every doubling step, so the roots are split across the spectrum at every level of the recursion.*

Proof. We show by induction that $z_{2n} < 1$ at every level, which guarantees that $\xi = 1 - z_n > 0$ and hence the splitting $y_{2n} < z_n < z_{2n}$ at every subsequent doubling.

Base case. $z_1 = \mu/L < 1$ since $\mu < L$ by assumption.

Inductive step. Suppose $z_n < 1$, so $\xi = 1 - z_n \in (0, 1)$. We must show $z_{2n} = z_n(\xi + \sqrt{1 + \xi^2}) < 1$, i.e. that $(1 - \xi)(\xi + \sqrt{1 + \xi^2}) < 1$. Rearranging, this is equivalent to

$$(1 - \xi)\sqrt{1 + \xi^2} < 1 - \xi + \xi^2.$$

Both sides are positive, so squaring and expanding gives

$$\begin{aligned} \text{LHS}^2 &= (1 - \xi)^2(1 + \xi^2) = 1 - 2\xi + 2\xi^2 - 2\xi^3 + \xi^4, \\ \text{RHS}^2 &= (1 - \xi + \xi^2)^2 = 1 - 2\xi + 3\xi^2 - 2\xi^3 + \xi^4, \end{aligned}$$

so $\text{RHS}^2 - \text{LHS}^2 = \xi^2 > 0$, confirming $z_{2n} < 1$. Hence, $\xi > 0$ at every level, so $z_{2n} > z_n$ and then $y_{2n} = z_n^2 / z_{2n} < z_n$, giving $y_{2n} < z_n < z_{2n}$ throughout the recursion. This inequality is clear given the plot in Figure 6 as the blue curve is above the black line, which is above the green curve for the interval $(0, 1)$. □

Hence, the new auxiliary stepsizes y_n, z_n start small and converge to the end of the spectrum. We convert these to the actual stepsizes a_n, b_n to be used in the schedule via the ψ function, which maps the interval $[0, \infty]$ of auxiliary stepsizes to the interval $[1/L, 1/\mu]$ of actual stepsizes[6], where

$$\psi(t) = \frac{1 + \kappa t}{1 + t}, \qquad a_n = \psi(y_n), \qquad b_n = \psi(z_n). \tag{21}$$

Now that we have generated the actual stepsizes a_n, b_n, we construct the exact order in which to use them in the schedule. We note that

$$\psi(0) = 1, \quad \psi(1/\kappa) = \frac{2}{1 + 1/\kappa}, \quad \psi(1) = \frac{1 + \kappa}{2}, \quad \text{and} \quad \psi(\infty) = \kappa. \tag{22}$$

Remark (Rescaling to the original problem)**.** The ψ function maps auxiliary stepsizes in $[0, \infty]$ to values in $[1, \kappa]$, corresponding to the **normalised** problem. For a problem with spectrum $[\mu, L]$, every stepsize h_k must be **divided by L to readjust**.

The only question left is how to order these a_n, b_n stepsizes. Again following from [2], the pattern is defined recursively by

$$h^{(2n)} = [\tilde{h}^{(n)}, a_n, \tilde{h}^{(n)}, b_n] \qquad \text{for } n = 1, 2, 4, 8, \ldots \tag{23}$$

where $h^{(1)} = [a_1]$ and $\tilde{h}^{(n)}$ is all of $h^{(n)}$ except the last element. For example,

$$h^{(1)} = [a_1], \qquad h^{(2)} = [a_2, b_2],$$
$$h^{(4)} = [a_2, a_4, a_2, b_4], \qquad h^{(8)} = [a_2, a_4, a_2, a_8, a_2, a_4, a_2, b_8].$$

Note that $h^{(1)} = [2/(1 + 1/\kappa)]$ from Equation (22); for the scaled problem, this is the optimal stepsize we proved in Theorem 2.1. If we consider the analogous construction of a schedule for the auxiliary stepsizes y_n, z_n, we illustrate both in Figure 7.

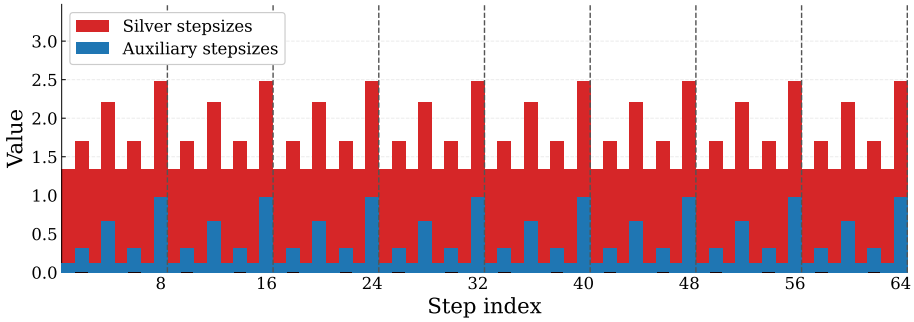


Figure 7: *The Silver Stepsize Schedule* $h^{(n)}$ *and auxiliary schedule for* $n = 8$ *and* $\kappa = 4$.

We also note that a_8 and b_8 are both larger than 2, which is the bound from Theorem 2.1. This bound ensures a monotone decrease in the error norm.

[6]Note that it is λ that is on the scale $[\mu, L]$ while the stepsizes are on the scale $[1/L, 1/\mu]$.

3.3 The Silver Polynomial

Now that we have generated the stepsizes and the schedule, we plot the corresponding polynomial $P_n(\lambda)$ and compare it to the equivalent Chebyshev polynomial with the same number of roots (so that it is a fair comparison). Recall from Equation (18) that the Chebyshev minimax bound on $[\mu, L]$ is

$$C_n = 2\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^n. \tag{24}$$

In Figure 8, the green sections show where the Silver polynomial exceeds C_n in absolute value, which occurs at the ends of the interval. This is because the Chebyshev polynomial is designed to minimise the maximum value across the entire interval. Hence, it places roots near the endpoints to suppress the error there, where it would be largest as polynomials grow quickly away from their roots. Because the Silver polynomial does not have this restriction, it can have values orders of magnitude smaller in the middle. Hence, only if we find ourselves near μ or L will the Silver schedule be worse than Chebyshev. The areas covered and magnitude of the polynomial become more pronounced as the problem becomes more ill-conditioned.

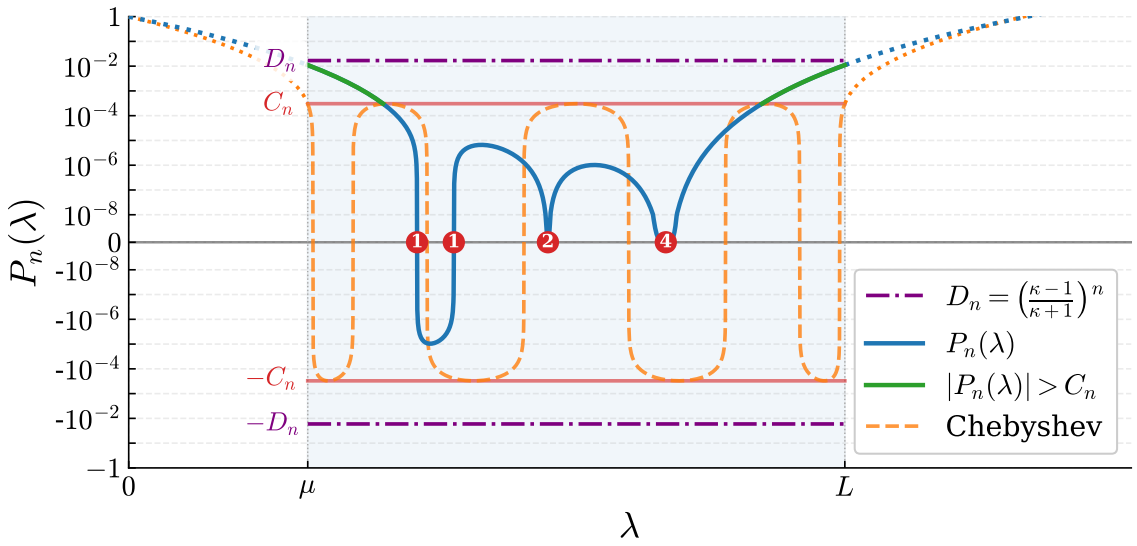


Figure 8: *Comparison of the polynomial $P_n(\lambda)$ generated by the SSS against the Chebyshev polynomial of equal degree. The Chebyshev envelope C_n from Equation (24) and the gradient descent bound D_n from Equation (6) are shown for reference. The Silver polynomial's roots are shown as red dots with multiplicities indicated. The parameters used are as in Figure 7.*

We can also note that the Silver polynomial is always below the gradient descent bound of D_n from Equation (6). So for this particular choice of parameters, Silver outperforms gradient descent (this is true in general [2], but this is not a proof).

4 Numerical experiments

There are many aspects of the method we can now question and explore through numerical experiments. For example, the algorithm generates a series of stepsize schedules $h^{(n)}$ for $n = 1, 2, 4, \ldots$. But how do we know which of these to use? Which is best if we have a fixed budget of N iterations? Alternatively, and more commonly, how many iterations should we run to achieve a certain error threshold?

4.1 Pseudo-stepsize Schedules

We introduce a new naïve pseudo-stepsize schedule, $\hat{h}^{(m)} := h^{(2^{\lceil \log_2 m \rceil})}[:m]$ of length $m \in \mathbb{N}$. For example, $\hat{h}^{(10)}$ would be the first 10 stepsizes of $h^{(16)}$. For each schedule, we loop over it until we have taken $N = 256$ steps. This gives us a continuum of schedules to see what is happening as we vary the cycle length n in Figure 9.

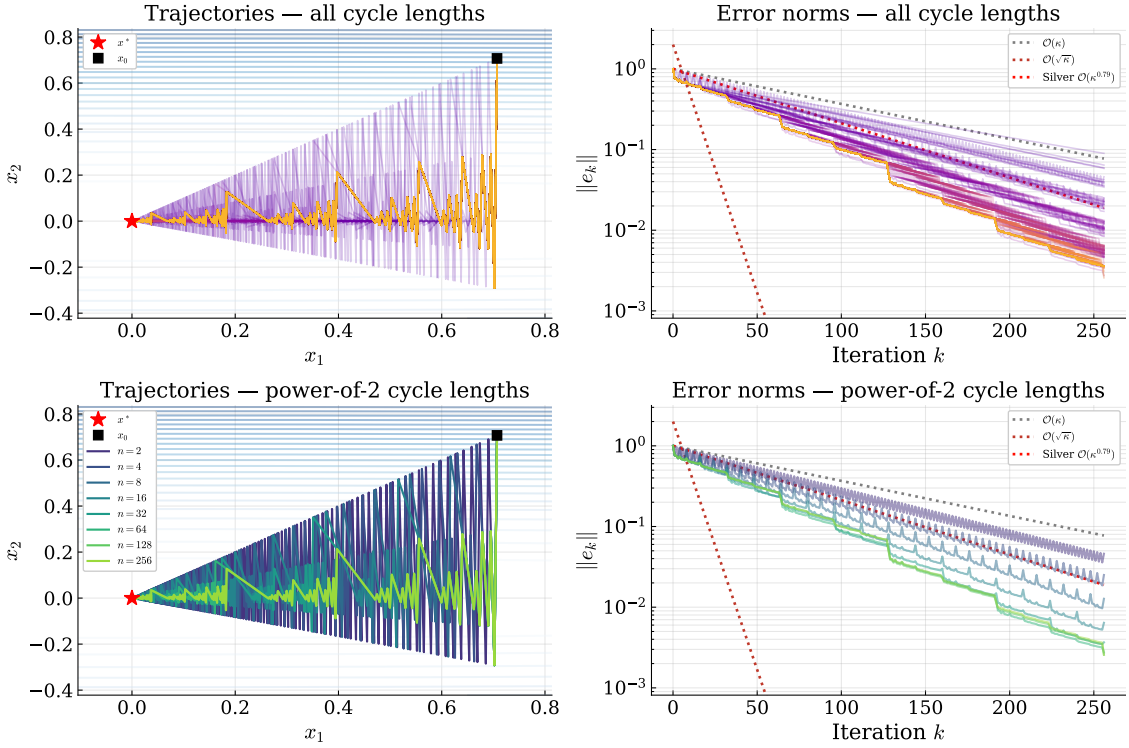


Figure 9: *Trajectories and error norms of the Silver Stepsize Schedule on an ill-conditioned problem*[7] *with* $\kappa = 200$ *over 256 iterations with cycle lengths* $n \in \{1, \ldots, 256\}$. ***Top row:*** *Trajectories and error norms for all cycle lengths using* $\hat{h}^{(n)}$. ***Bottom row:*** *The same plots restricted to power-of-2 cycle lengths* $n \in \{2, 4, 8, 16, 32, 64, 128, 256\}$. *Theoretical bounds are shown for reference in dotted lines.*

[7] To better understand the behaviour of the Silver Stepsize Schedule, we run it on an ill-conditioned problem. We can see this in the contours in the top and bottom left of Figure 9 being nearly horizontal. This is why most of the trajectories move up and down the valley rather than directly towards the minimum (imagine a person skiing in the Olympics down a shallow half pipe).

4.2 Extended Silver Stepsize Schedule

In Figure 10 we see that only when we complete the full cycles of powers of 2 do we get the best performance (shown by red dots), with the best overall being the largest cycle, $n = 256$. This is consistent with [2], which shows that the schedule is designed to be used in full cycles and that when the iteration budget is exactly $N = 2^k$, the optimal schedule is the full $h^{(2^k)}$. This is because the last step of the cycle carries the largest stepsize b_n. Noting that we prefer to complete full cycles, we construct a continuum of better intermediate schedules.

The Silver Stepsize Schedule $h^{(n)}$ is defined for horizons n that are powers of 2. For an arbitrary horizon $N \in \mathbb{N}$, we propose the new *Extended Silver Stepsize Schedule* $\bar{h}^{(N)}$, constructed via the binary decomposition of N.

4.3 Construction of $\bar{h}^{(N)}$

If we write N in its binary expansion as

$$N = 2^{k_1} + 2^{k_2} + \cdots + 2^{k_r}, \qquad \text{where} \qquad k_1 > k_2 > \cdots > k_r \geq 0,$$

then we define $\bar{h}^{(N)}$ by concatenating the complete Silver Stepsize Schedules corresponding to each term in this decomposition, ordered from largest to smallest[8]:

$$\bar{h}^{(N)} := \left[h^{(2^{k_1})},\ h^{(2^{k_2})},\ \ldots,\ h^{(2^{k_r})} \right]. \tag{25}$$

When N is itself a power-of-2, this reduces to the original schedule: $\bar{h}^{(N)} = h^{(N)}$. The previous $\hat{h}^{(m)}$ omits the terminal b_n stepsizes at each recursive level, which are the largest stepsizes and carry the convergence benefit of each level of the recursion. By contrast, $\bar{h}^{(N)}$ ensures every b_{2^k} appears exactly once for k in the decomposition of N (except for b_1 which is never used). For example, for $N = 7$, we have,

$$\bar{h}^{(7)} = [h^{(4)}, h^{(2)}, h^{(1)}] = [a_2, a_4, a_2, b_4, a_2, b_2, a_1],$$

compared to the previous

$$\hat{h}^{(7)} = h^{(2^{\lceil \log_2 7 \rceil})}[:7] = h^{(8)}[:7] = [a_2, a_4, a_2, a_8, a_2, a_4, a_2].$$

To be clear, in Figure 10 we generate the schedules $\hat{h}^{(m)}$ and $\bar{h}^{(m)}$ for all cycle lengths $m \in \{1, \ldots, 256\}$. Then, for each schedule, we loop over it until we have taken 256 steps, and plot the error norm at the end.

[8]The order is from largest to smallest because the larger cycles contain the more powerful b_n stepsizes, so we want to complete those first as they will give us the largest reduction in error. See Appendix B for a discussion on the ordering of the cycles.

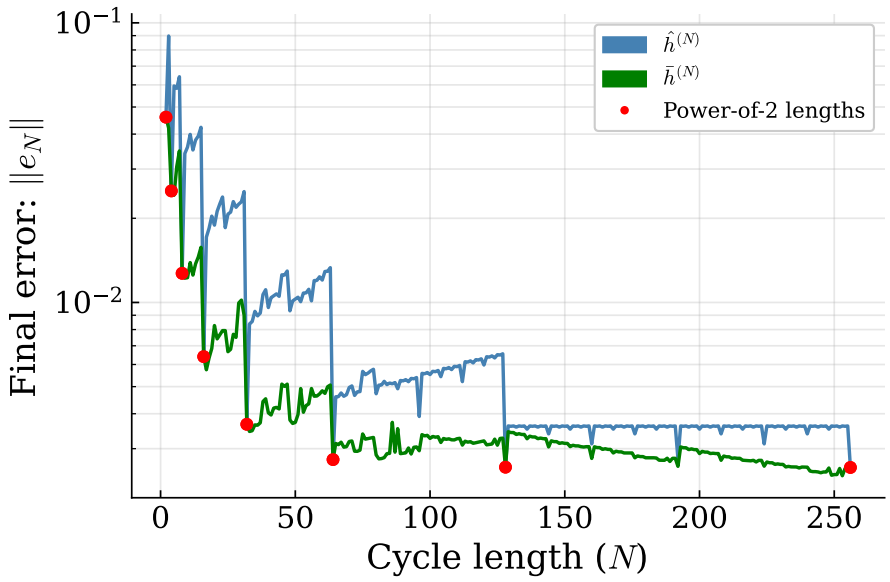


Figure 10: *Final error as a function of cycle length. Powers of two are marked with circles.*

From Figure 10 we see that $\bar{h}^{(m)}$ achieves a final error less than or equal to that of $\hat{h}^{(m)}$ for all cycle lengths m, with equality precisely at the power-of-2 lengths. This confirms the motivation behind the construction: ensuring b_{2^k} terms appear. For $\hat{h}^{(m)}$, the error is larger between consecutive powers of 2 and only drops sharply when a new b_n term is finally reached. As for $\bar{h}^{(m)}$, the growth between power-of-2 lengths is far less, and in some cases the error even drops between consecutive powers of 2. Note that in Figure 10, $\bar{h}^{(253)}$ achieves a lower error than the full $h^{(256)}$. This observation motivated Conjecture 4.3.

Conjecture (Extended Silver Stepsize Schedule Improvement)**.** For sufficiently ill-conditioned problems, cycling the ESSS $\bar{h}^{(N^*)}$ to complete N steps achieves a final error less than that of $h^{(N)}$ for all $N = 2^r$, with $r \geq 3$. Here $N^* = N - 3$; since N is a power-of-2, its binary decomposition is $N^* = 2^r - 3 = 11\ldots101_2$ in binary and is r bits long. Hence, we write

$$\bar{h}^{(N^*)} = \left[h^{(2^{r-1})},\ h^{(2^{r-2})},\ \ldots,\ h^{(2^2)},\ h^{(2^0)}\right], \quad N^* = \sum_{i=0, i\neq 1}^{r-1} 2^i, \quad k_1 > \cdots > k_r \geq 0.$$

Then the stepsize schedule computed is expressed as $\left[\bar{h}^{(N^*)},\ \bar{h}^{(N^*)}[:3]\right]$, compared to the original schedule of $h^{(N)}$. We can also use the construction of $\bar{h}^{(N^*)}$ given in Equation (25) more generally to construct a schedule for any N. This conjecture is supported by the numerical experiments in Figure 11.

4.4 Analysis of Optimal Cycle Length

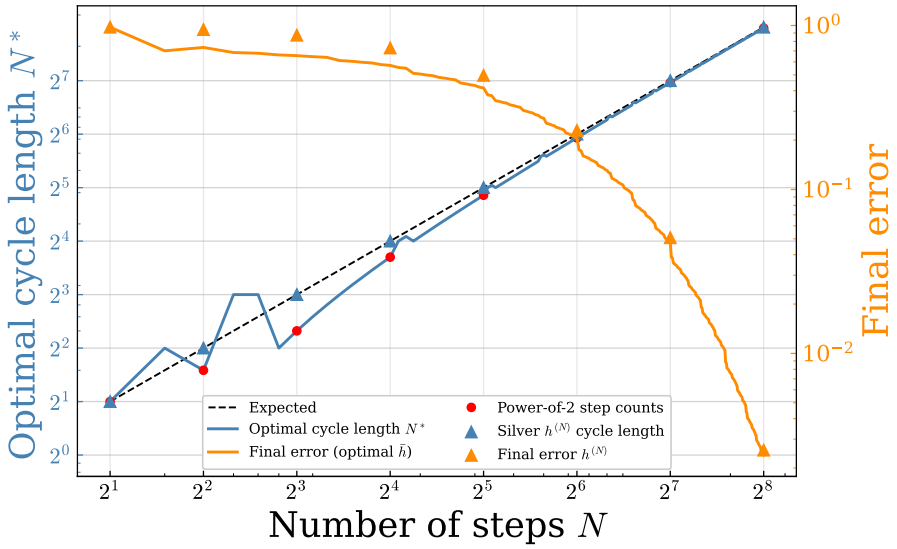


Figure 11: *Optimal cycle length N^* as a function of the number of steps. Computed by iterating over different cycle lengths and choosing the one that minimises the final error. The parameters here are $\mu = 0.005$ and $L = 1$*

From Figure 11, we draw several conclusions about the optimal cycle length N^* as a function of the number of steps N. The blue line shows that the optimal cycle length broadly tracks the expected $N^* = N$ line (dashed), confirming that longer Silver schedules are generally better as the budget grows. However, the red dots at power-of-2 steps lie *below* the blue triangles, meaning that at these budgets the optimal cycle length N^* is strictly less than N. In other words, cycling a shorter $\bar{h}^{(N^*)}$ schedule achieves a lower final error than running the full certified $h^{(N)}$ schedule exactly once[9]. The orange line confirms that the final error decreases monotonically with N as expected, with drops at power-of-2 budgets where a new full Silver schedule becomes available. The orange triangles lying above the orange line at these points further confirm that $h^{(N)}$, when used as a cycling schedule, is suboptimal compared to the shorter optimal $\bar{h}$.

We leave a formal proof of this conjecture as future work: a rigorous argument would likely require extending the recursive gluing certificate of [2] to non-power-of-2 horizons, which is a substantial theoretical undertaking beyond the scope of this project. The conjecture is instead supported by the numerical evidence seen here[10] with Figure 11 given as an example, which we consider sufficient motivation.

[9]Note that in the cycling process, we only cycle the first three steps of $\bar{h}^{(N^*)}$. These will always be $[a_2, a_4, a_2]$. This could be marginally improved by running $\bar{h}^{(3)} = [h^{(2)},\ h^{(1)}] = [a_2, b_2, a_1]$ instead, but the difference is likely negligible for N large.

[10]https://youtu.be/FSRPs0QzBuY

4.5 Rosenbrock Function

One important feature of the Silver Stepsize Schedule is that it works beyond the quadratic setting. To explore this, we apply it to the Rosenbrock function [12],

$$f(x) = (1 - x_1)^2 + 100(x_2 - x_1^2)^2, \tag{26}$$

which has a unique global minimum at $x^* = (1, 1)$ with $f(x^*) = 0$. The function is a standard benchmark for optimisation algorithms due to its narrow curved valley (see Figure 12), which makes it highly ill-conditioned and difficult to navigate. Altschuler and Parrilo prove in their companion paper [13] that the SSS also works for smooth, non-convex functions, for example the Rosenbrock function. Hence, we apply the Silver schedule to this problem and compare it to other methods.

Since the Silver schedule requires estimates of the strong convexity parameter μ and smoothness constant L, which are not globally available for Rosenbrock, we use fixed estimates $\mu = 1$ and $L = 1001$, giving $\kappa = 1001$. These are obtained from the smallest and largest eigenvalues of the Hessian at x^*. We compare four methods: constant-stepsize GD, parameter-free NAG with time-varying momentum $k/(k+3)$ [7], the Silver schedule $\hat{h}^{(N)}$, and our newly proposed Extended Silver Schedule $\bar{h}^{(N^*)}$.

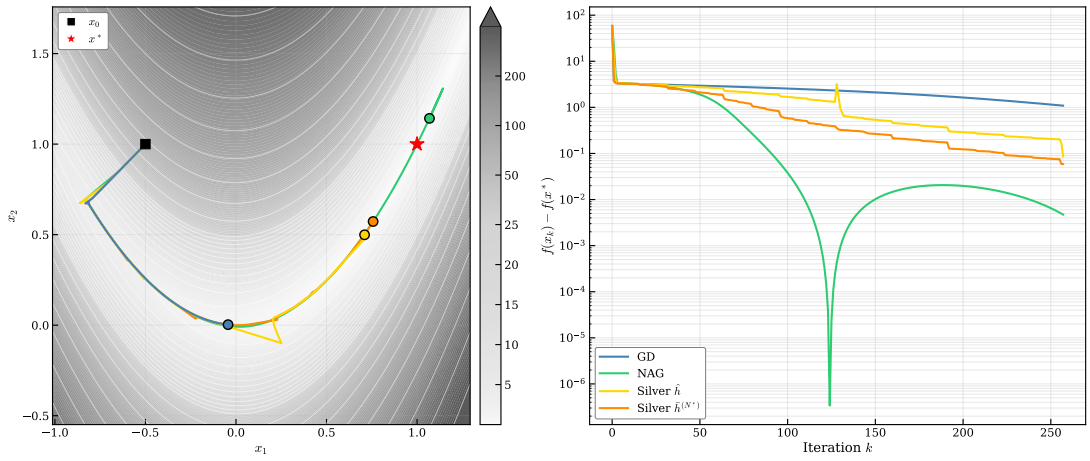


Figure 12: *Comparison of optimisation methods on the Rosenbrock function Equation (26).* ***Left:*** *Trajectories overlaid on the function contours.* ***Right:*** *Function value $f(x_k)$ over iterations. Initialised at $x_0 = (-0.5, 1.0)$ and run for 256 iterations.*

From Figure 12 we see that NAG converges fastest, as expected, and both Silver schedules perform better than GD, with $\bar{h}^{(N^*)}$ slightly outperforming $\hat{h}^{(N)}$. For an animation of how the trajectories evolve over time, see the accompanying video[11].

[11] https://youtu.be/P2ChTCWAUNw

5 Existing Literature and Related Work

Since the Silver Stepsize Schedule was introduced, several works have proposed alternative schedules with different convergence properties. We compare $\hat{h}$ and $\bar{h}$ with three such schedules that all also noticed the flaw in the original SSS in that it only works for powers of two. These schedules were tested on the Rosenbrock function with the same estimated parameters, as shown in Figure 13. We do not go into the details of these methods as they are beyond the scope of this project.

Zhang anytime [14]. This schedule achieves $\mathcal{O}(\kappa^{0.893})$ convergence for *any* stopping time. This decrease in performance is the price to pay for the anytime property.

Grimmer OBS-F [15]. A dynamic programming approach that constructs the stepsize schedule of any length by recursively combining shorter schedules via a single join stepsize, achieving the same $\mathcal{O}(\kappa^{0.7864})$ with an improved constant over Silver.

Grimmer long-step [16]. A fractal schedule with the same recursive structure as Silver but with different join stepsizes, achieving the same $\mathcal{O}(\kappa^{0.7864})$ iteration complexity as Silver but with an improved constant factor.

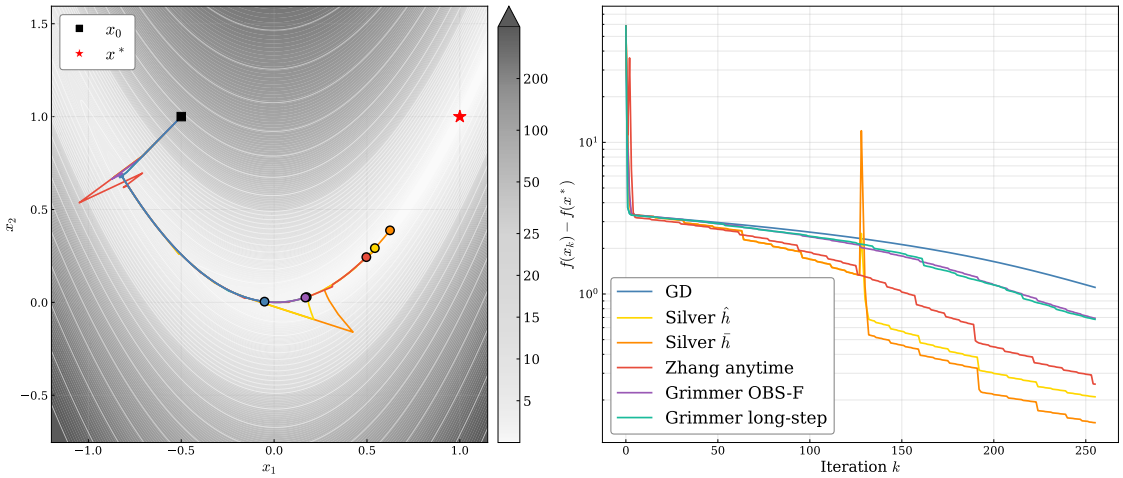


Figure 13: *Trajectories and error norms for each schedule on the Rosenbrock function.*

From Figure 13, all methods converge on the Rosenbrock function. The Extended Silver Schedule $\bar{h}$ achieves the lowest final error. The Zhang anytime schedule performs reasonably, while OBS-F and the Grimmer long-step schedule perform comparably to constant-stepsize GD. We also note that the $\hat{h}$ schedule outperforms these more sophisticated schedules. Plots of the methods' stepsizes can be found in Appendix C. A more rigorous comparison of these methods would require testing across a range of problems and parameters, which would be an interesting avenue for future work.

6 Conclusions

This report has investigated the Silver Stepsize Schedule, a momentum-free gradient descent method that works in the smooth and strongly convex setting. It achieves a convergence rate of $\mathcal{O}(\kappa^{\log_\rho 2}) \approx \mathcal{O}(\kappa^{0.7864})$, strictly between the unaccelerated rate $\mathcal{O}(\kappa)$ of constant-stepsize GD and the optimal first-order rate $\mathcal{O}(\sqrt{\kappa})$ of NAG and Chebyshev. Notably, Silver is the first known momentum-free method proven to improve over constant-stepsize GD on smooth, strongly convex functions, resolving a question that had been open since Nesterov's seminal work [7]. We reviewed the classical first-order methods and their limitations, before introducing the Silver construction and its recursive, fractal-like structure. Through the polynomial framework of Section 2.3, we saw that the Silver schedule hedges between small and large stepsizes to distribute roots across the spectral interval, achieving a better worst-case polynomial than constant-stepsize GD. Because it does not have the restriction of minimising the maximum value across the entire interval like Chebyshev, it can achieve much smaller values in the middle, leading to a much larger reduction in error for certain steps.

Restarting strategies, as used with NAG [8], were also considered for improved performance. However, restarting is beneficial for NAG precisely because it accumulates a momentum term across iterations, and resetting this to zero removes accumulated 'bad history'. The Silver schedule has no such momentum state: each step depends only on the current iterate and stepsize, so we did not pursue this further.

In extending the original SSS via the naïve $\hat{h}^{(m)}$ approach, numerical experiments of Section 4 showed us that the performance of the schedule is highly sensitive to the choice of cycle length. Only at power-of-2 lengths does the full convergence benefit materialise, as it is at the final step of each recursive level that the large b_n stepsize delivers its payoff. This motivated our new ESSS $\bar{h}^{(N^*)}$, constructed by concatenating complete power-of-2 sub-schedules ordered by binary decomposition. Empirically, $\bar{h}^{(N^*)}$ consistently outperforms the naïve prefix truncation $\hat{h}^{(m)}$ for all non-power-of-2 cycle lengths and matches it for power-of-2 lengths. Most strikingly, Figure 11 suggests that for power-of-2 budgets, cycling a shorter $\bar{h}^{(N^*)}$ schedule with $N^* = N - 3$ outperforms the theoretically certified $h^{(N)}$ schedule. It remains open whether this holds in general, and a proof would shed light on the broader conjecture of [2] that $\kappa^{\log_\rho 2}$ is optimal among all momentum-free stepsize schedules. Finally, we applied both Silver schedules to the non-quadratic Rosenbrock function, where ESSS outperformed other extensions of SSS. The SSS is a powerful new method that opens up many avenues for further research, both theoretical and numerical, in the design of momentum-free first-order optimisation algorithms with accelerated rates.

References

[1] Yurii E. Nesterov. *Introductory Lectures on Convex Optimization: A Basic Course*. Vol. 87. Applied Optimization. New York: Springer, 2004. ISBN: 978-1-4020-7553-7. DOI: 10.1007/978-1-4419-8853-9.

[2] Jason M. Altschuler and Pablo A. Parrilo. "Acceleration by stepsize hedging: Multi-step descent and the silver stepsize schedule". In: *Journal of the ACM* 72.2 (2025), pp. 1–38.

[3] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. 4th. Johns Hopkins University Press, 2013.

[4] Guillaume Garrigos and Robert M. Gower. *Handbook of Convergence Theorems for (Stochastic) Gradient Methods*. 2023. arXiv: 2301.11235 [math.OC]. URL: https://arxiv.org/abs/2301.11235.

[5] Boris T. Polyak. "Some methods of speeding up the convergence of iteration methods". In: *USSR Computational Mathematics and Mathematical Physics* 4.5 (1964), pp. 1–17. DOI: 10.1016/0041-5553(64)90137-5.

[6] Laurent Lessard, Benjamin Recht, and Andrew K. Packard. "Analysis and Design of Optimization Algorithms via Integral Quadratic Constraints". In: *SIAM Journal on Optimization* 26.1 (2016), pp. 57–95. DOI: 10.1137/15M1009597.

[7] Yurii E. Nesterov. "A method for solving the convex programming problem with convergence rate $O(1/k^2)$". In: *Doklady Akademii Nauk SSSR* 269.3 (1983). English translation in *Soviet Mathematics Doklady*, 27(2):372–376, 1983, pp. 543–547.

[8] Brendan O'Donoghue and Emmanuel J. Candès. "Adaptive Restart for Accelerated Gradient Schemes". In: *Foundations of Computational Mathematics* 15.3 (2015), pp. 715–732. DOI: 10.1007/s10208-013-9150-3.

[9] Lloyd N. Trefethen. *Approximation Theory and Approximation Practice*. Extended. Philadelphia: SIAM, 2019. ISBN: 978-1-611975-93-0.

[10] David M. Young. "Iterative methods for solving partial difference equations of elliptic type". In: *Transactions of the American Mathematical Society* 76.1 (1954), pp. 92–111.

[11] Donald A. Flanders and George H. Shortley. "Numerical determination of fundamental modes". In: *Journal of Applied Physics* 21.12 (1950), pp. 1326–1332.

[12] Howard H. Rosenbrock. "An automatic method for finding the greatest or least value of a function". In: *The Computer Journal* 3.3 (1960), pp. 175–184. DOI: 10.1093/comjnl/3.3.175.

[13] Jason M. Altschuler and Pablo A. Parrilo. "Acceleration by stepsize hedging: Silver Stepsize Schedule for smooth convex optimization". In: *Mathematical Programming* 213 (2025), pp. 1105–1118. DOI: 10.1007/s10107-024-02164-2.

[14] Zihan Zhang, Jason Lee, Simon Du, and Yuxin Chen. "Anytime Acceleration of Gradient Descent". In: *Proceedings of Thirty Eighth Conference on Learning Theory*. Vol. 291. Proceedings of Machine Learning Research. PMLR, 2025, pp. 5991–6013. URL: https://proceedings.mlr.press/v291/zhang25a.html.

[15] Benjamin D. Grimmer, Kevin Shu, and Alex L. Wang. *Composing Optimized Stepsize Schedules for Gradient Descent*. 2024. arXiv: 2410.16249 [math.OC]. URL: https://arxiv.org/abs/2410.16249.

[16] Benjamin D. Grimmer. *Provably Faster Gradient Descent via Long Steps*. 2023. arXiv: 2307.06324 [math.OC]. URL: https://arxiv.org/abs/2307.06324.

Appendix

A Glossary

Acronym	Definition
ESSS	Extended Silver Stepsize Schedule
GD	gradient descent
NAG	Nesterov accelerated gradient
SPD	symmetric positive definite
SSS	Silver Stepsize Schedule

B Ordering of Sub-Schedules in $\bar{h}^{(N)}$

The construction of $\bar{h}^{(N)}$ in Equation (25) concatenates Silver schedules from largest to smallest, giving

$$\bar{h}^{(N)} = \left[h^{(2^{k_1})},\ h^{(2^{k_2})},\ \ldots,\ h^{(2^{k_r})}\right], \qquad k_1 > k_2 > \cdots > k_r \geq 0.$$

Here we compare this against the reversed (smallest-to-largest) ordering

$$\bar{h}^{(N)}_{\text{small-first}} = \left[h^{(2^{k_r})},\ h^{(2^{k_{r-1}})},\ \ldots,\ h^{(2^{k_1})}\right]$$

on both a quadratic and the Rosenbrock function, to justify the choice of ordering.

On the quadratic objective Equation (1), the error after all N steps is determined by the polynomial $P_N(\lambda) = \prod_{k=0}^{N-1}(1-\alpha_k\lambda)$, which is a product and therefore independent of the ordering of $\{\alpha_k\}$. The two orderings must produce identical final errors, and this is confirmed in Figure 14. Although the intermediate errors $\|e_k\|$ differ substantially between the two orderings — with the difference reaching $\sim 10^{-1}$ in the bottom panel — the final errors agree to machine precision ($\sim 10^{-16}$). This is seen in the very bottom right of the bottom panel in Figure 14.

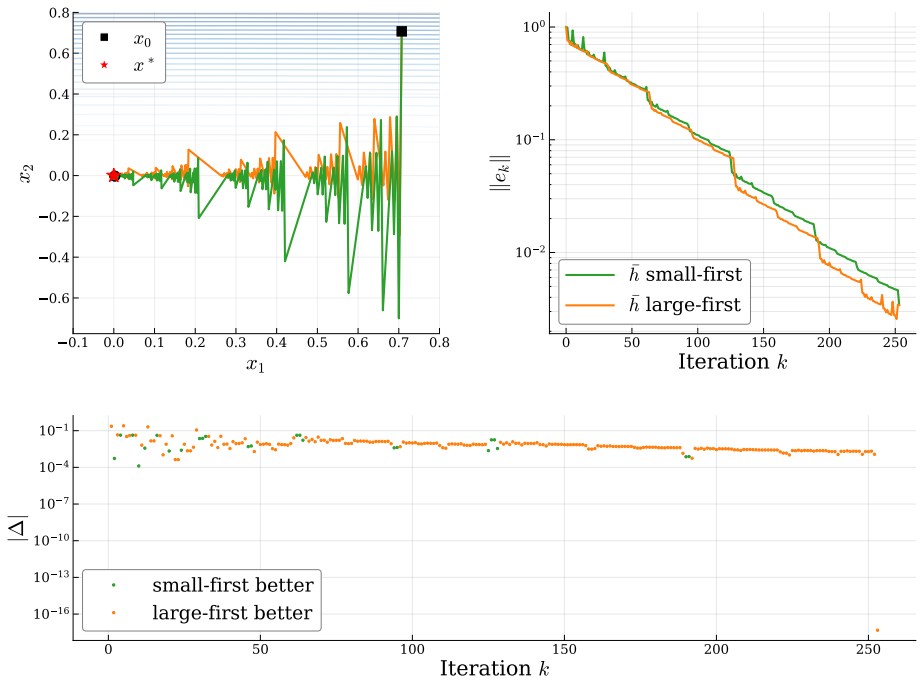


Figure 14: *Ordering comparison on a quadratic with $\kappa = 200$ and $N = 253$ steps. The intermediate errors differ visibly, but the final errors are identical. The bottom panel confirms that the absolute difference $|\Delta|$ drops to machine precision ($\sim 10^{-16}$) by the final iteration.*

On the Rosenbrock function, however, the ordering genuinely matters. Since the objective is non-quadratic, the commutativity of the error polynomial no longer applies. As shown in the bottom panel of Figure 15, the small-first ordering performs better at early iterations because it completes full sub-cycles sooner, while the large-first ordering only completes its first sub-cycle at $k = 128$. After this point, the large-first ordering dominates, ultimately achieving a lower final function value. This supports the heuristic that executing the larger sub-schedules first yields a greater overall reduction in error, after which the smaller sub-schedules refine the iterate.

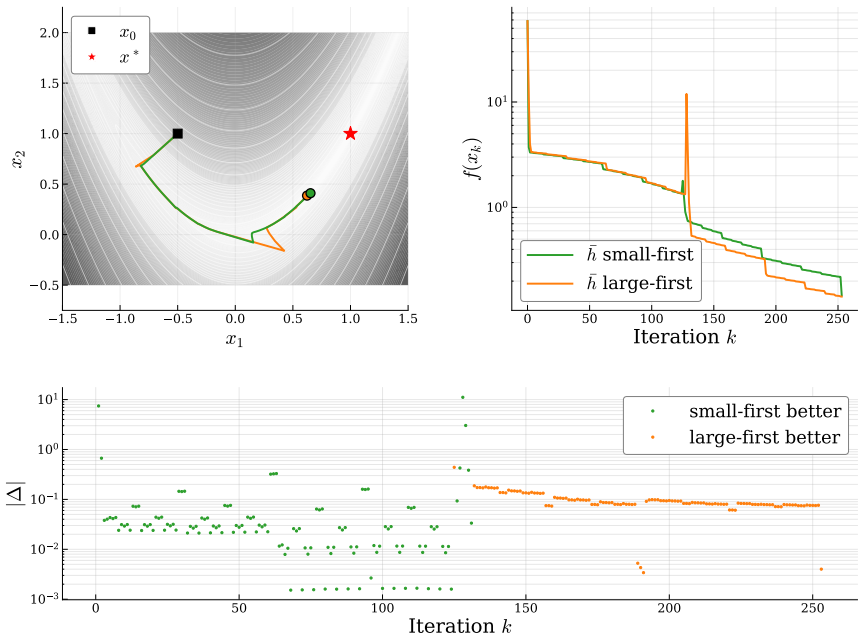


Figure 15: *Ordering comparison on the Rosenbrock function with $\mu = 1$, $L = 1001$, and $N = 253$ steps. Despite a transient spike, the large-first ordering achieves a lower final function value.*

Remark. Note that in Figure 15 in the top left image, although the green dot looks closer to the minimum than the orange dot, it actually has a higher function value. The contour lines are very close in the valley, so a small change in position may lead to a large change in function value.

C Stepsize Plots

Here we plot the stepsizes for $\hat{h}$, $\bar{h}$, and the other methods discussed in Section 5. Note that the optimal α from Theorem 2.1 is approximately $2/L$ as we use $\mu = 1$ and $L = 1001$, hence $2/(\mu + L) \approx 2/L$.

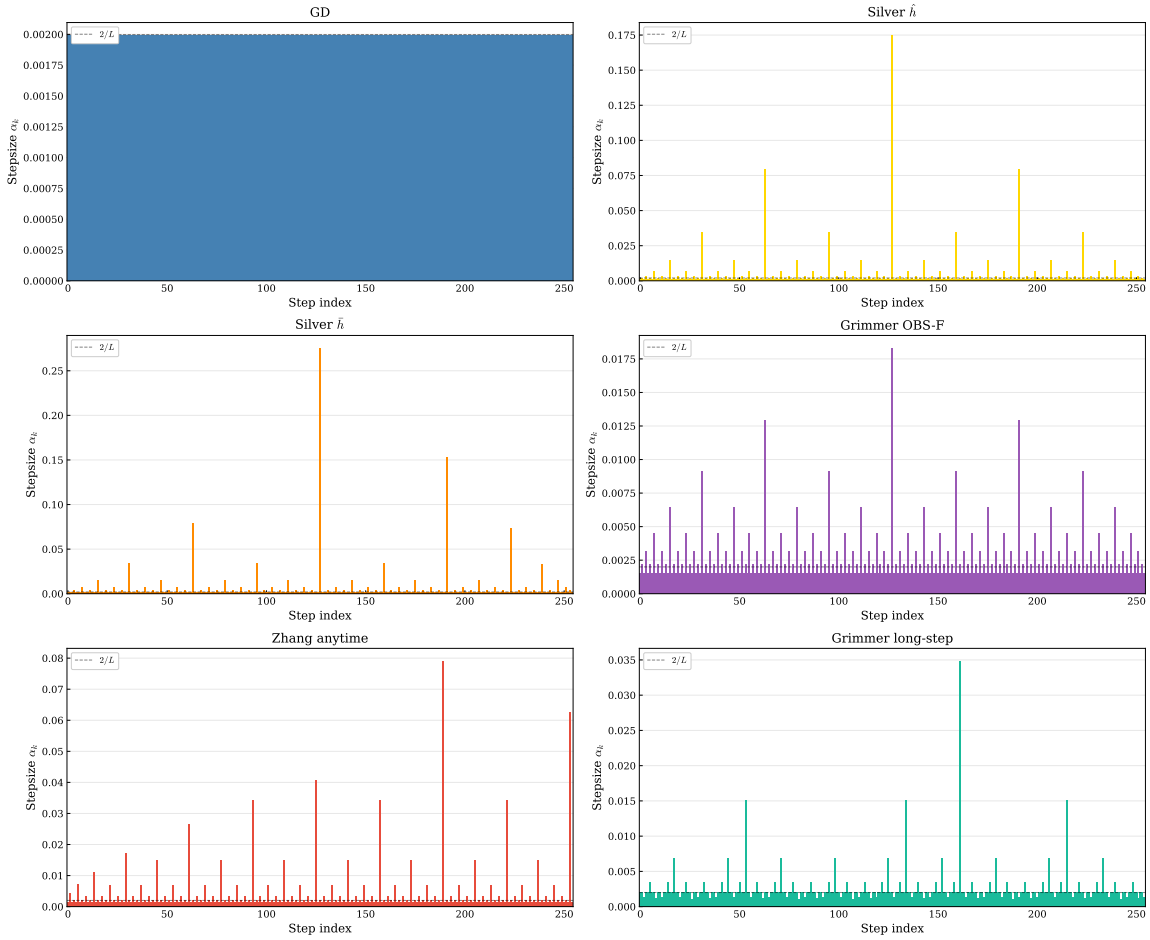


Figure 16: *Stepsizes for each schedule where* $\mu = 1$ *and* $L = 1001$.