Stochastic Quasi-Newton Optimization in Large Dimensions Including Deep Network Training

***Abstract*—Our proposal is on a new stochastic optimizer for non-convex and possibly non-smooth objective functions typically defined over large dimensional design spaces. Towards this, we have tried to bridge noise-assisted global search and faster local convergence, the latter being the characteristic feature of a Newton-like search. Our specific scheme – acronymed FINDER (Filtering Informed Newton-like and Derivative-free Evolutionary Recursion), exploits the nonlinear stochastic filtering approach to arrive at a derivative-free update that bears resemblance with the Newton search employing the inverse Hessian of the objective function. Following certain simplifications of the update to enable a linear scaling with dimension and a few other enhancements, we apply FINDER to a range of problems, starting with some IEEE benchmark objective functions to a couple of archetypal data-driven problems in deep networks to certain cases of physics-informed deep networks. The performance of the new method vis-á-vis the well-known Adam and a few others bears evidence to its promise and potentialities for large dimensional optimization problems of practical interest.**

***Index Terms*—Stochastic optimization, FINDER, Ensemble Kalman filter, Quasi-Newton method, Deep neural networks.**

I. Introduction

FROM maximizing energy efficiency in engineering systems to resource allocations in health care, the ubiquitous need for optimization arises across diverse domains. By systematically identifying the bespoke solution within a feasible set, optimization facilitates informed decision-making, improved processes and efficient routes to the desired outcomes. The goal here is to extremize a real-valued objective or loss function f depending on, say, N design variables with or without additional constraints. The landscape of optimization methods may be broadly divided into deterministic and stochastic. Prominent among the deterministic methods are linear and nonlinear programming [1], [2] as well as variants of gradient descent [3]. While effective in cases where f is convex, these methods face challenges when applied to non-convex problems [4], [5]. Newton's methods utilize the Hessian of the objective function f [6]–[10] and enjoy quadratic convergence, thus marking a great leap in deterministic convex optimization. However, the risk of being trapped in a local optimum is significant in such cases. Additionally, the inversion of the Hessian matrix in higher dimensions demands high computational overhead (even with lower-rank approximations of the inverse). Other limitations include possible sensitivity to the initial guess and a risk of divergence in the presence of non-convexity or non-smoothness. Real-world problems are usually non-convex, an instance being the training of Physics-Informed Neural Networks (PINNs) [11], [12]. Addressing this challenge requires carefully crafted strategies that navigate multimodality whilst not sacrificing the convergence rate.

Being noise-driven and thus enabled with global search for non-convex problems, stochastic search has naturally attracted attention [13]. A special note is taken here of meta-heuristic algorithms, where the noisy search is often inspired by biological evolution or some other physical or social phenomena. Unlike derivative-based methods, meta-heuristics operate based only on the objective function values. Though sometimes lacking in mathematical rigor, they have often worked well in navigating complex search spaces. Some notable examples of meta-heuristics include Genetic Algorithms (GA) [14], Particle Swarm Optimization (PSO) [15] etc. However, with increasing dimension of the search space, the number of particles to be sampled may grow exponentially and render such schemes ineffective. A few other derivative-free stochastic techniques are Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES) [16], simulated annealing [17] and differential evolution [18] etc. CMA-ES, in particular, stands out for its effectiveness in optimizing multimodal and non-smooth functions across myriad problem types. Nonetheless, it faces drawbacks, including substantial computational overheads in higher dimensions and the presence of many hyper-parameters in the algorithm. We refer to [19] for a comprehensive account of both deterministic and stochastic optimization methods.

For large-dimensional optimization problems, the Adaptive Moment Estimation (Adam) [20] - a gradient-based scheme - has emerged as a powerful tool, especially for machine learning tasks. Its key strength is in adaptive and dynamic step-size control for each component of the state vector based on past gradients, resulting in faster convergence. Although Adam has advanced the state-of-the-art in optimization, it is not without limitations - the requirement of tuning several hyperparameters and possible convergence to sub-optimal solutions in non-convex and noisy problems, to name a few. Indeed, an optimization method, with twin abilities of linear scaling across dimensions and good performance at global search, remains largely elusive. It is to this cause that the present article is devoted.

The inherent trade-off between faster convergence and quality of solution underscores the complexity of optimization challenges. In this context, despite its limitation on scaling with dimensions, a stochastic approach, aided by the rich machinery of stochastic calculus, should ideally be seen as a potent, if slower, choice. Our present aim is to harness the rapid local convergence of a Newton-like strategy whilst simultaneously allowing a noise-aided exploration of the search space. However, evaluating the Hessian is not just costly but prone to significant errors when done numerically. Drawing inspiration from measure-theoretic ideas in stochastic filtering, we adopt a derivative-free methodology that computes a

Table I
TABLE OF NOTATIONS

Notation	Description
$a, \vec{a}, \boldsymbol{A}$	scalar, vector, matrix
A	a vector random variable
t	time-like quantity
A_t	a random (stochastic) process
$\mathcal{N}$	normal probability distribution
$\mathcal{U}$	uniform probability distribution
A^j	j^{th} realization of A
f	objective (loss) function
∇f	gradient of f (a column vector)
$\langle \vec{u}, \vec{v} \rangle$	inner product of $\vec{u}$ and $\vec{v}$
$\langle \vec{u}, \vec{v} \rangle_{\boldsymbol{A}}$	inner product of $\vec{u}$ and $\boldsymbol{A}\vec{v}$
$\mathbb{E}_{\mathbb{P}}[A]$	expectation of A under measure $\mathbb{P}$
w.r.t. $\vec{a}\uparrow$	with respect to increasing order of $\vec{a}$
argsort($\vec{a}$, $\boldsymbol{A}$, ..)	sort columns of ($\boldsymbol{A}$, ..) w.r.t. $\vec{a}\uparrow$
$\boldsymbol{A}^s$	matrix after sorting columns in $\boldsymbol{A}$
$\overline{\boldsymbol{A}}$	arithmetic mean of all columns vectors of $\boldsymbol{A}$
$\boldsymbol{A}^j$	j^{th} column vector in $\boldsymbol{A}$
$\boldsymbol{A}_{ij}$	$(i,j)^{th}$ entry of $\boldsymbol{A}$
$[a]_{m\times n}$	$m \times n$ matrix with every element being a
$\boldsymbol{I}_n$	rank n identity matrix
$u_{,x_1 \ldots x_n}$	$\frac{\partial^n u}{\partial x_1 \ldots \partial x_n}$
$\boldsymbol{A}^T$	transpose of matrix $\boldsymbol{A}$
$a \wedge b$	minimum of a and b

stochastic mimicking of the inverse Hessian of an objective function. Specifically, we show that an imposition of the requirement of vanishing gradient of f via nonlinear stochastic filtering naturally yields a matrix (a stochastic equivalent of matrix-valued Lagrange multipliers) that may be interpreted as the mimicked inverse Hessian. This circumvents an explicit computation of the Hessian and its inverse. A grounding within the theory of diffusive stochastic processes ensures that the capability for noise-assisted exploration is never sacrificed. Since the original form of the mimicked inverse Hessian requires unwieldy matrix inversion in higher dimensions, we introduce simplifications along with a scalar step-size (learning rate) parameter. Finally, via our numerical implementation, we demonstrate how the method works for a reasonably broad range of optimization problems including a few on the training of deep networks. The last class of examples also features training of PINNs.

The rest of the paper is organized as follows. Section II presents the broad mathematical framework for the dynamics of the hidden design state X by posing it as a nonlinear filtering problem. This section also includes a detailed algorithm for the proposed optimizer. Section III demonstrates the numerical performance of the optimizer by examining a set of IEEE benchmark functions. The working of the optimizer in training deep neural networks is also demonstrated in this section. Finally, some concluding remarks are given in Section IV. Table I gives an outline of the notations used in this article.

II. MATHEMATICAL FORMULATION

A. Problem Statement

We consider the following optimization (minimization) problem:

$$\underset{\vec{x}}{\operatorname{argmin}}\ f(\vec{x}) \text{ where } \vec{x} \in D \subseteq \mathbb{R}^N \tag{1}$$

However, for all practical purposes, we work with the following statement: given a cost function $f : \mathbb{R}^N \to \mathbb{R}^+ \cup \{0\}$,

$$\text{find } \vec{x}^* \in \mathrm{D} \subseteq \mathbb{R}^N \text{ such that } f(\vec{x}^*) \leq \varepsilon_{tol}$$

where ε_{tol} is a user-defined tolerance. Ideally, $f(\vec{x})$ should be differentiable w.r.t. $\vec{x}$, enabling the use of gradient-based optimization techniques. Continuity and smoothness of f are desirable attributes, aiding in convergence and stability during the optimization process.

In a complete probability space $(\Omega, \mathcal{F}, \mathbb{P})$ [21] equipped with the filtration $(\mathcal{F}_t)_{t\geq 0}$ derived from subsets of the sample space Ω, we examine a vector-valued diffusion process $X_t := X(t)$, adapted to $\mathcal{F}_t$ and varying over the time-like interval $t \in [0, T]$. We wish to arrive at a stochastic dynamical scheme for X_t so that $\mathbb{E}_{\mathbb{P}}(X_t) \to \vec{x}^*$ as $t \to \infty$. Here, $X_t = \{(X_t)_i : i = 1, \ldots, N\}$, denotes a weak solution to a stochastic differential equation (SDE) in $\mathbb{R}^N$, driven by an N-dimensional standard Brownian motion B_t:

$$dX_t = \boldsymbol{R}_t dB_t \tag{2}$$

where $\boldsymbol{R}_t$ is the diffusion matrix. Eqn. (2) is referred to as the process dynamics and X_t as the process variable. In order to impose a stochastic equivalent of the constraint $\nabla f = 0$, we use the principle of stochastic filtering to require that ∇f be reduced to a zero-mean martingale, i.e. a stochastic process whose expected future value conditioned on the present is the present value itself [22]. Accordingly, we supplement Eqn. (2) with a constraint or observation equation in terms of the observation variable $Y_t \in \mathbb{R}^N$ such that the consequent update for X_t mimics the one in Newton's algorithm. Specifically, we write the observation or measurement equation for every t in the algebraic form:

$$Y_t = \nabla f + \dot{W}_t \tag{3}$$

and require $Y_t = \vec{0}$ weakly (i.e. with posterior probability 1) for all t. Here $W_t \in \mathbb{R}^N$ denotes the measurement noise – another Brownian motion independent of B_t, so that $\dot{W}_t$ is formally a white noise. Denoting the filtration generated by Y_t by $\mathcal{F}_t^Y$, the estimate $\hat{X}_t$ of X_t under this constraint would be given by the conditional expectation $\mathbb{E}_{\mathbb{P}}(X_t|\mathcal{F}_t^Y)$. Our aim is to find a computationally expedient, t-recursive update map for $\hat{X}_t$.

An inspection of the observation Eqn. (3) reveals that the optimization aim is met, i.e. $\mathbb{E}_{\mathbb{P}}(\hat{X}_t) \to \vec{x}^*$, when ∇f converges in distribution to a zero-mean martingale, which is interpreted as the stochastic equivalent of the condition $\nabla f = 0$. In nonlinear filtering, this is achieved using a change of measures that leads to a generalized Bayes rule (the Kallianpur-Striebel formula) or, equivalently, the Kushner-Stratonovich equation to evolve $\hat{X}_t$ [23]–[25]. A simpler form of this update, as with an ensemble Kalman filter (EnKF), may be arrived at by defining the innovation vector $I_t = \nabla f(X_t)$ [26]. Thus, for a step-size Δt, a recursive update map for $\hat{X}_t$ may be expressed as:

$$\hat{X}_{t+\Delta t} = \hat{X}_t + \tilde{\boldsymbol{G}} I_{t+\Delta t} \tag{4}$$

where $\tilde{\boldsymbol{G}}$ is the gain matrix. Upon a direct comparison with Newton's update that incorporates the inverse Hessian, it is

evident that $-\tilde{\boldsymbol{G}}$ serves as the stochastic mimicking of the inverse Hessian. An explicit expression of $\tilde{\boldsymbol{G}}$ is as follows:

$$\tilde{\boldsymbol{G}} = -\left[\int_{\Omega} \left(\boldsymbol{X} - \overline{\boldsymbol{X}}\right)\left(\boldsymbol{G} - \overline{\boldsymbol{G}}\right)^{T} d\mathbb{P}\right] \left[\int_{\Omega} \left(\boldsymbol{G} - \overline{\boldsymbol{G}}\right)\left(\boldsymbol{G} - \overline{\boldsymbol{G}}\right)^{T} d\mathbb{P} + \boldsymbol{Q}\right]^{-1} \tag{5}$$

where $\boldsymbol{X}$ is the matrix with particles as column vectors; $\boldsymbol{G}$ is the matrix of gradients of cost function for each particle in $\boldsymbol{X}$; $\boldsymbol{Q} = \mathbb{E}_{\mathbb{P}}[dW_t dW_t^T]$ is the measurement noise covariance matrix. By way of a ready reference, a derivation is given in the supplementary material, also available at https://github.com/FINDER-optimizer/FINDER. In practical Monte Carlo implementations involving finite (possibly small) ensembles, we may take the Frobenius norm $||\boldsymbol{Q}||$ to be negligibly small.

Though ∇f could be computed in a derivative-free setup [27], we presently use analytical expressions or automatic differentiation, as appropriate. However, a great difficulty with the method above is the matrix inverse appearing in $\tilde{\boldsymbol{G}}$, which would prevent a linear scaling across dimensions. Yet another computational hurdle is with MC simulations of the equations involved, where the number of particles may need to increase exponentially with dimension in the absence of a trustworthy variance reduction strategy. These call for further modifications and simplifications, which we undertake in Section B.

B. Implementation and Algorithm

In line with Section A, the algorithm begins with Eqn. (2), whose solution requires sampling multiple particles/realizations around a mean, followed by observing gradients ∇f (Eqn. (3)) at these particles. The particles and respective gradients obtain a mimicked inverse Hessian as in Eqn. (5), which is used to update the ensemble via Eqn. (4).

Step 1 – Generate Ensemble: From Eqn. (2), we have $(dB_t)_i \sim \mathcal{N}(0, dt) \Rightarrow (\Delta B_t)_i \sim \mathcal{N}(0, 1)$ upon discretization with $\Delta t = 1$, where subscript i denotes the i^{th} component of the vector. We weakly replace this Gaussian increment with a zero-mean uniform density of spread $[-1, 1]$ with a proper choice of the diagonal matrix $\boldsymbol{R}_t$ so as to match the second moments. Compared with a Gaussian distribution, such a choice leads to less sampling time, especially in large dimensions. Accordingly consider a vector random variable $Z \in \mathbb{R}^N$ such that $Z \sim \prod_{i=1}^{N} \mathcal{U}(-1, 1)$; then we have:

$$X_t^j = X_t^1 + \boldsymbol{R}_t Z^j, \qquad j = 2, \ldots, p \tag{6}$$

where X_0^1 is the initial solution which is iteratively updated. We construct a matrix $\boldsymbol{X}_{N\times p}$ with particles of the ensemble at a given t as column vectors.

Step 2 – Observe Gradients: We determine the gradients for each particle – either through analytical expression or automatic differentiation, and store them columnwise in the matrix $\boldsymbol{G}_{N\times p}$. We also evaluate the cost of each particle and store them in cost vector $\vec{f}_{1\times p}$. Subsequently, we sort columns in $\boldsymbol{X}$ and $\boldsymbol{G}$ in increasing order of $\vec{f}$ to obtain $\boldsymbol{X}^s$ and $\boldsymbol{G}^s$.

Step 3 – Approximate Inverse Hessian: As already noted, the evaluation of $\tilde{\boldsymbol{G}}$, as in Eqn. (5), could be computationally expensive - and even infeasible, for higher dimensional problems. Hence, for the sake of computational expedience including variance reduction, we approximate $\tilde{\boldsymbol{G}}$ by a diagonal matrix $\boldsymbol{B}$ [see Section 5.7.1 in [28]] whose elements are given by:

$$\boldsymbol{B}_{ii} = \frac{\sum_{j=1}^{p}(\boldsymbol{X}_{ij}^s - \overline{\boldsymbol{X}_i^s})(\boldsymbol{G}_{ij}^s - \overline{\boldsymbol{G}_i^s})}{\sum_{j=1}^{p}(\boldsymbol{G}_{ij}^s - \overline{\boldsymbol{G}_i^s})(\boldsymbol{G}_{ij}^s - \overline{\boldsymbol{G}_i^s})} \tag{7}$$

Note that the second factor (the inverse matrix) in the expression of the gain $\tilde{\boldsymbol{G}}$ is indeed diagonal in principle. This follows from the statistical independence of the scalar components of W_t. However, in a finite-ensemble Monte Carlo simulation, such a diagonal structure is not generally realized. Apart from relieving the computational burden of matrix inversion, this approximation also has the added benefit of variance reduction by ignoring particle scatter due to the off-diagonal entries. Above all, it reduces the number of particles (we use $p = 5$) needed for computing $\boldsymbol{B}$. In this context, we may as well mention that based on the work of Schoenmaker and Milstein [29], one may, in principle, use an appropriate change of measures to achieve 'perfect' variance reduction, i.e. simulation with just one particle. However, given the inherent circularity in the implementation of this principle, we do not attempt it here. We enforce $\boldsymbol{B}$ to be positive semi-definite so that $\langle \boldsymbol{B}\nabla f, \nabla f\rangle \geq 0$. This implies that the direction given by $-\boldsymbol{B}\nabla f$ is still a direction of descent. Accordingly, we set all the negative entries in $\boldsymbol{B}$ to zero. This makes sense, given that for any non-degenerate vector-valued random variable, the diagonal entries of the covariance matrix must be positive – a condition that may possibly be violated during MC simulations with just a few particles. To compensate for the loss of accuracy that accrues due to the approximation of $\tilde{\boldsymbol{G}}$ by a diagonal matrix, we introduce a parameter γ as an exponent of $\boldsymbol{B}$ i.e. $\tilde{\boldsymbol{B}} := \boldsymbol{B}^{\gamma}$ where $0 \leq \gamma \leq 1$ [30]. Substituting $\tilde{\boldsymbol{G}}$ with $\tilde{\boldsymbol{B}}$ in Eqn. (4), it is evident that $\gamma = 0$ implies a gradient descent-like update while $\gamma = 1$ (used in our numerical work) implies a BFGS-mimicking update. Such a substitution also provides a handle to regulate the step size driven by the curvature of the design space topology. An intuitive grasp of the connection between curvature (derived from the inverse Hessian) at a point and the optimal step size suggests that when curvature is substantial, only a small step is necessary. Hence we choose $0 \leq \gamma \leq 1$.

Step 4 – Update Ensemble: We modify the update Eqn. (4) to replace the increment $\tilde{\boldsymbol{G}} I_{t+\Delta t}$ with $\boldsymbol{\Delta}_{t+1}$ given by:

$$\boldsymbol{\Delta}_{t+1} = \theta\boldsymbol{\Delta}_t + \tilde{\boldsymbol{B}}\boldsymbol{G}^s \tag{8}$$

where $\theta \in [0, 1)$ and $\boldsymbol{\Delta}_0 = [0]_{N\times p}$. Since $\Delta t = 1$ and we start the recursion with $t = 0$, we treat t as a non-negative integer. Accordingly, we update the ensemble with the following equation:

$$\hat{\boldsymbol{X}} = \boldsymbol{X}^s - \alpha_t\boldsymbol{\Delta}_{t+1} \tag{9}$$

where $\alpha_t \in (0, 0.1]$ is a step-size multiplier. Eqn. (8) represents a convolution of past increments against exponentially decaying weights, i.e. $\boldsymbol{\Delta}_t = \sum_{k=1}^{t} \theta^k \boldsymbol{\Delta}_{t-k}$. $0 \leq \theta < 1$

ensures that the weights decay in a geometric progression. The convolution step helps accelerate the algorithm in regions with slowly varying gradients, which could otherwise be slow to navigate through. For example, if past increments align in the same direction, it leads to an increase in the value of $\boldsymbol{\Delta}_{t+1}$. θ close to 1 implies higher reliance on past increments, whereas a value near 0 suggests otherwise. Yet another crucial element is the computation of the step-size multiplier α_t. An appropriate α_t in the dominant descent direction $-\boldsymbol{\Delta}^1_{t+1}$ may be obtained iteratively using the Armijo rule [31].

$$f\left((\boldsymbol{X}^s)^1 - \alpha_t \boldsymbol{\Delta}^1_{t+1}\right) \leq f\left((\boldsymbol{X}^s)^1\right) - c_\alpha \alpha_t \langle \boldsymbol{\Delta}^1_{t+1}, (\boldsymbol{G}^s)^1 \rangle \tag{10}$$

The hyper-parameter c_α is typically in the range of 10^{-3} to 10^{-2}. We include $(\boldsymbol{X}^s)^1$ in the updated ensemble matrix $\hat{\boldsymbol{X}}$ to obtain the new ensemble matrix $\boldsymbol{X}'_{N\times(p+1)}$. We obtain the cost vector $\vec{f}'$ for the new ensemble by evaluating the cost of each particle in $\boldsymbol{X}'$ and subsequently find the 'best' particle $(\boldsymbol{X}'^s)^1$, and the 'worst' particle $(\boldsymbol{X}'^s)^{p+1}$ in $\boldsymbol{X}'$. $(\boldsymbol{X}'^s)^1$ serves as the new mean X^1_{t+1} for the next iteration. This ensures that the loss function is always non-increasing.

Step 5 – Update Diffusion Matrix: Next we update $\boldsymbol{R}_t$ using the following expression:

$$\begin{aligned}\Gamma_{t+1} &= (1-c_s)\Gamma_t + c_s\left((\boldsymbol{X}'^s)^{p+1} - (\boldsymbol{X}'^s)^1\right)\\ (\boldsymbol{R}_{t+1})_{ii} &= \begin{cases} |(\Gamma_{t+1})_i| \wedge \zeta_1 \text{ (say } 10^{-2}), & \text{if } (\Gamma_{t+1})_i \neq 0\\ \zeta_2 \text{ (say } 10^{-2}), & \text{otherwise}\end{cases}\end{aligned} \tag{11}$$

where $c_s = 0.1$; $\Gamma_0 = [0]_{N\times 1}$ and $\boldsymbol{R}_0 = 0.1\boldsymbol{I}_N$. The vector Γ takes into account the difference between $(\boldsymbol{X}'^s)^{p+1}$ and $(\boldsymbol{X}'^s)^1$ with a suitable weight $(1-c_s)$ applied to the history of such updates. The idea here is to sample the i^{th} component from the range $|(\boldsymbol{X}'^s)^{p+1}_i - (\boldsymbol{X}'^s)^1_i|$ around $(\boldsymbol{X}'^s)^1_i$ in the next iteration. $c_s = 0.1$ attaches higher weight to the history so the search span does not vary abruptly. Following this idea we wish to choose $|(\Gamma_{t+1})_i|$ as $(\boldsymbol{R}_{t+1})_{ii}$. However, $\boldsymbol{R}_{t+1}$ must have a bounded spectral radius so that all particles are sampled from a bounded neighborhood around the mean. Accordingly, we prescribe an upper limit on $|(\Gamma_{t+1})_i|$, say ζ_1 to obtain $(\boldsymbol{R}_{t+1})_{ii}$. It is also appropriate to avoid an unphysical zero value for $(\boldsymbol{R}_{t+1})_{ii}$ to avoid nil sampling along the i^{th} component. The prescribed limits on the elements of $\boldsymbol{R}_{t+1}$ may themselves be treated as hyper-parameters (ζ_1 and ζ_2) if they aid in finding the minima with a higher degree of precision. As the estimation of $\boldsymbol{B}$ is dependent on the spread of the ensemble around the mean, the performance of FINDER is sensitive to the choices of ζ_1, ζ_2. Higher ζ_1, ζ_2 render the algorithm more exploratory while lower values make it more deterministic. Low diagonal values of $\boldsymbol{R}_{t+1}$ could also be weakly construed as the individual variances arising from zero-mean uniform densities with an appropriate spread. Such a weak replacement of the Gaussian density is admissible [32] as it does not interfere with the local rate of convergence of the Euler-Maruyama method for discretizing the process SDE.

Pseudo-code: The pseudo-code of the proposed optimization scheme is furnished in Algorithm 1.

Algorithm 1 FINDER

Set $p, \theta, \gamma, c_s, c_\alpha, \zeta_1, \zeta_2$ # hyper-parameters
Initialize $X^1_0, \boldsymbol{R}_0, \boldsymbol{\Delta}_0, \Gamma_0, t = 0, \Delta t = 1$ # state variables
while $f > \varepsilon_{tol}$ and $t \leq T$ **do:**
  for $j = 2$ to p **do** : # generate ensemble
    $X^j_t = X^1_t + \boldsymbol{R}_t\ Z^j$ where $Z \sim \prod_{i=1}^{N} \mathcal{U}(-1, 1)$
  end
  $\boldsymbol{X} = [X^1_t, X^2_t, \ldots X^p_t]_{N\times p}$ # ensemble matrix
  $\boldsymbol{G} = [\nabla f(\boldsymbol{X})]_{N\times p}$ # matrix of gradients
  $\vec{f} = [f(\boldsymbol{X})]_{1\times p}$ # cost vector for $\boldsymbol{X}$
  $\boldsymbol{X}^s, \boldsymbol{G}^s \leftarrow$ argsort$(\vec{f}, \boldsymbol{X}, \boldsymbol{G})$ # Sort $\boldsymbol{X}$, $\boldsymbol{G}$ w.r.t. $\vec{f}\uparrow$
  Compute $\tilde{\boldsymbol{B}} = \boldsymbol{B}^\gamma$ # use Eqn.7 & set $\boldsymbol{B}_{ii}$ = 0 if $\boldsymbol{B}_{ii}$ <0
  $\boldsymbol{\Delta}_{t+1} = \theta\boldsymbol{\Delta}_t + \tilde{\boldsymbol{B}}\boldsymbol{G}^{\boldsymbol{s}}$ # compute increment term
  $\alpha_t \leftarrow$ Armijo rule$((\boldsymbol{X}^s)^1, (\boldsymbol{G}^s)^1, \boldsymbol{\Delta}^1_{t+1}, c_\alpha)$ # use Eqn. (10)
  $\hat{\boldsymbol{X}} = \boldsymbol{X}^s - \alpha_t\boldsymbol{\Delta}_{t+1}$ # update the ensemble
  $\boldsymbol{X}' = [\hat{\boldsymbol{X}}, (\boldsymbol{X}^s)^1]_{N\times(p+1)}$ # concatenate $(\boldsymbol{X}^s)^1$ to $\hat{\boldsymbol{X}}$
  $\vec{f}' = [f(\boldsymbol{X}')]_{1\times(p+1)}$ # cost vector for $\boldsymbol{X}'$
  $\boldsymbol{X}'^s \leftarrow$ argsort$(\vec{f}', \boldsymbol{X}')$ # sort $\boldsymbol{X}'$ w.r.t. $\vec{f}'\uparrow$
  $\Gamma_{t+1} = (1-c_s)\Gamma_t + c_s\left((\boldsymbol{X}'^s)^{p+1} - (\boldsymbol{X}'^s)^1\right)$
  for $i = 1$ to N **do:**
    $(\boldsymbol{R}_{t+1})_{ii} = \begin{cases} |(\Gamma_{t+1})_i| \wedge \zeta_1, & \text{if } (\Gamma_{t+1})_i \neq 0\\ \zeta_2, & \text{otherwise}\end{cases}$
  end
  $X^1_{t+1} \leftarrow (\boldsymbol{X}^s)^1$
  $t = t + 1$
Return $\vec{x}^*$ = X^1_t # optimal solution
Output Record and plot the results for comparison.

Remarks on convergence

As already noted, the adopted filtering framework follows the structure of the EnKF [33], which applies to our case after the linearization of the observation equation (see Eqn. (3)). It has been shown in [34] that the EnKF-based update map follows a gradient flow which we now briefly discuss. In doing so, we only focus on the broad framework of our filtering-driven optimization, thereby largely bypassing the simplifications used for scaling with dimensions (e.g. diagonalization of $\tilde{\boldsymbol{G}}$) and other performance-enhancing features (e.g. history dependence). Consider the simplified form of the update as in Eqn. (4) along with the following observation equation which holds almost surely:

$$\vec{0} = \nabla f(X_t) + W_t \tag{12}$$

Fundamental to our discussion is the minimization of the following functional:

$$\phi(\vec{x}) = \frac{1}{2}||\boldsymbol{Q}^{-1/2}\nabla f(\vec{x})||^2 \tag{13}$$

where $\boldsymbol{Q}$ is the covariance of the observation noise W. The posterior measure $\rho(d\vec{x})$ of $\vec{x}$ depends on ϕ and is given by $\rho(d\vec{x}) \propto \exp(-\phi(\vec{x}))\rho_0(d\vec{x})$, where ρ_0 denotes the prior measure. Upon linearization of $\nabla f(\vec{x})$, we may write a locally quadratic approximation of Eqn. (13) as:

$$\phi_L(\vec{x}) = \frac{1}{2}||\boldsymbol{Q}^{-1/2}\boldsymbol{H}\vec{x}||^2 \tag{14}$$

$\boldsymbol{H}$ is the Hessian of f. As shown in [34], the continuous time limit of the recursive update of $\vec{x}$ may be approximated as

$$\frac{d\boldsymbol{X}^j}{dt} = -\boldsymbol{C}(\boldsymbol{X})\nabla_{\vec{x}}\phi_L(\boldsymbol{X}^j) \tag{15}$$

where $\boldsymbol{C}(\boldsymbol{X}) = \frac{1}{p}\sum_{j=1}^{p}(\boldsymbol{X}^j - \overline{\boldsymbol{X}})(\boldsymbol{X}^j - \overline{\boldsymbol{X}})^T$ is the ensemble covariance. The gradient structure as above clearly ensures convergence of the updates whose rate is dictated by the positive eigenvalues of the positive definite matrix $\boldsymbol{C}(\boldsymbol{X})$. More specifically, note that $\phi_L(\vec{x})$ may be interpreted as a Lyapunov function for the dynamical flow defined by Eqn. (15). This is because $\frac{d\phi_L(\vec{x})}{dt} = -\langle\nabla_{\vec{x}}\phi_L(\boldsymbol{X}^j), \nabla_{\vec{x}}\phi_L(\boldsymbol{X}^j)\rangle_{\boldsymbol{C}(\boldsymbol{X})} \leq 0$.

In addition, by defining the particle-wise anomaly $\boldsymbol{E}_t^j = \boldsymbol{X}_t^j - \overline{\boldsymbol{X}_t}$, where $\overline{\boldsymbol{X}_t}$ is the ensemble average at the pseudo-time t, it may be shown that a quadratic form of the locally propagated error, given by the matrix $\boldsymbol{\mathcal{E}}_{ij} = \langle \boldsymbol{H}\boldsymbol{E}^i, \boldsymbol{H}\boldsymbol{E}^j\rangle_{\boldsymbol{Q}}$, evolves according to $\frac{d}{dt}\boldsymbol{\mathcal{E}} = -\frac{2}{p}\boldsymbol{\mathcal{E}}^2$. Hence $\boldsymbol{\mathcal{E}}$ converges to 0 as $t \to \infty$. In other words, all the particles asymptotically collapse to the empirical mean.

Following the discretized form of the filtering dynamics as in Eqn. (4) and a supermartingale structure of the evolving $f(\hat{X}_t)$, it should also be possible to invoke Doob's martingale convergence theorem [35] to prove convergence of our proposed scheme. Indeed, a closer look at the approximations in our gain matrix reveals that the supermartingale structure of $f(\hat{X}_t)$ (or an appropriate subsequence of $f(\hat{X}_t)$), which is an essential element for convergence, is not interfered with. Details of such a proof will be taken up in a future article.

III. Results and Discussion

This section dwells on a numerical demonstration of the feasibility and efficacy of FINDER. In doing so, we choose several IEEE benchmark functions (mostly in relatively high dimensions) and a few problems on training deep neural networks (DNNs). All programs are executed on a system equipped with 32GB RAM and 4GB NVIDIA T1000 GPU, except for the CIFAR10 problem, which is executed on an open-access Google Colab engine with 12 GB RAM and 15GB T4 GPU. The programs, freely available online at https://github.com/FINDER-optimizer/FINDER, are written in Python programming language using standard open-access libraries. We have used the following values of hyper-parameters: $p = 5,\ \theta = 0.9,\ \gamma = 1,\ c_s = 0.1,\ c_\alpha = 0.01$ in all problems. $\zeta_1 = \zeta_2 = 0.01$ unless stated otherwise. In all comparisons with the Adam optimizer [20], a constant learning rate of 10^{-3} is used and the default values of the hyper-parameters are: $\beta_1 = 0.9,\ \beta_2 = 0.999, \epsilon = 10^{-8}$.

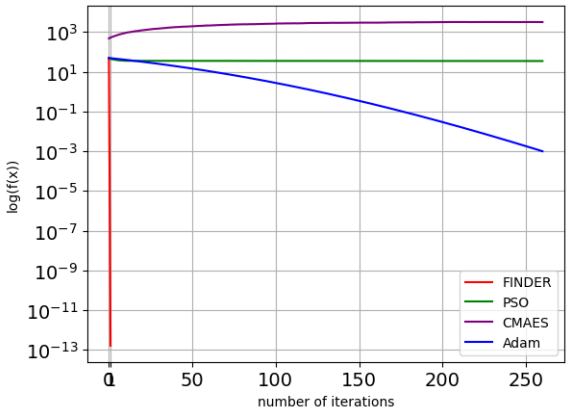

Figure 1. Minimization of 5000-dimensional sphere function

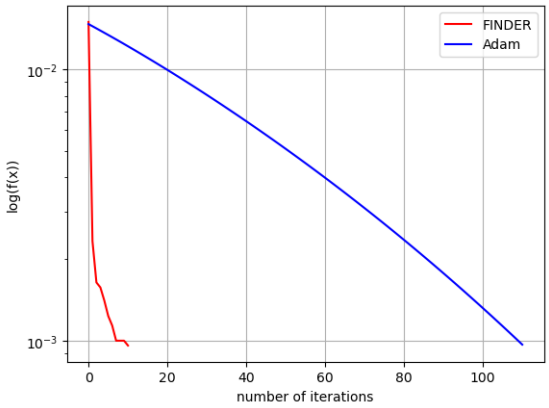

Figure 2. Minimization of the 5000-dimensional Sum of Different Powers function

A. IEEE benchmark problems

Since our optimizer is for very general applications, solving the IEEE benchmark problems [36] is a crucial step in establishing its practical usefulness. Moreover, these benchmark problems being well-established and widely recognized, provide a common ground for comparing different optimization algorithms. We also demonstrate an example showcasing the limitations of probabilistic methods such as the CMAES and PSO in higher dimensions. We demonstrate this while minimizing the sphere function with $N = 5000$. The initial guess $\vec{x}_0$ is randomly drawn from $\mathcal{N}(\mu_N, 10^{-4}I_N)$ distribution, where $\mu_N \in \mathbb{R}^N$ and I_N denotes the rank-N identity matrix. Analytical expressions for the objective function $f(\vec{x})$ and its gradients have been used in these problems. Table II and Figs. 1-6 summarise the comparative performances of FINDER and Adam. From the results on minimizing Ackley and Rosenbrock functions, it is clear that FINDER works well even when the true Hessian has non-zero off-diagonal terms.

B. Classification problems using neural networks

Classification problems are of immense significance in machine learning and artificial intelligence. By accurately recognizing labels (such as digits, animals, cars etc.) from images, they demonstrate a model's ability to understand

Table II
COMPARISON OF FINDER VIS-Á-VIS ADAM IN IEEE BENCHMARK FUNCTIONS ($N = 5000$)

Function Name	Mathematical Expression	μ_N	Order of Loss		Time (in sec.)		Iteration count	
			FINDER	Adam	FINDER	Adam	FINDER	Adam
Sphere	$\sum_{i=1}^{N} x_i^2$	$[0.1]_{N\times 1}$	10^{-12}	10^{-3}	0.0022	13.47	1	260
Sum of Different Powers	$\sum_{i=1}^{N} \lvert x_i\rvert^{i+1}$	$[0.1]_{N\times 1}$	10^{-3}	10^{-3}	0.063	5.71	10	110
Ackley	$-20\exp\left(-0.2\sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i)^2}\right)+20$ $-\exp\left(\frac{1}{N}\sum_{i=1}^{N}\cos(2\pi x_i)\right)+\exp(1)$	$[0.1]_{N\times 1}$	10^{-3}	10^{-3}	0.019	8.02	10	155
High-Conditioned Elliptic	$\sum_{i=1}^{N}\left(10^6\right)^{\frac{i-1}{N-1}} x_i^2$	$[0.1]_{N\times 1}$	10^{-7}	10^{-3}	0.045	33.15	1	417
Rastrigin	$10N+\sum_{i=1}^{N}\left(x_i^2-10\cos(2\pi x_i)\right)$	$[0.1]_{N\times 1}$	10^{-3}	0	0.016	15.99	7	303
Rosenbrock	$\sum_{i=1}^{N-1}\left[100\left(x_{i+1}-x_i^2\right)^2+(1-x_i)^2\right]$	$[1.1]_{N\times 1}$	10^{-3}	10^{-3}	182.28	338.17	1251	2850

Note: These problems have been implemented on CPU only. The Adam algorithm [20] was hand-coded for these problems.

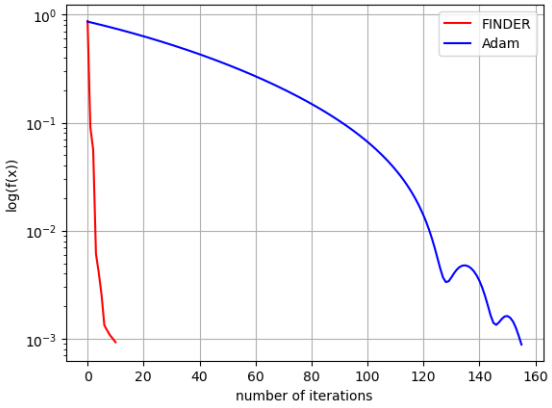

Figure 3. Minimization of 5000-dimensional Ackley function

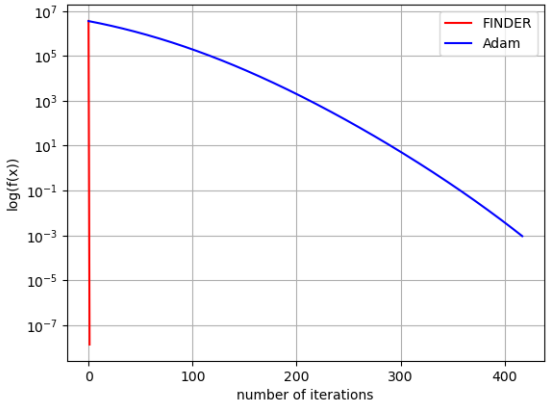

Figure 4. Minimization of 5000-dimensional High-Conditioned Elliptic function

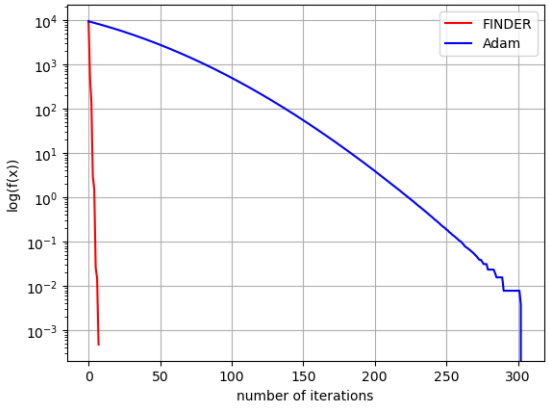

Figure 5. Minimization of 5000-dimensional Rastrigin function

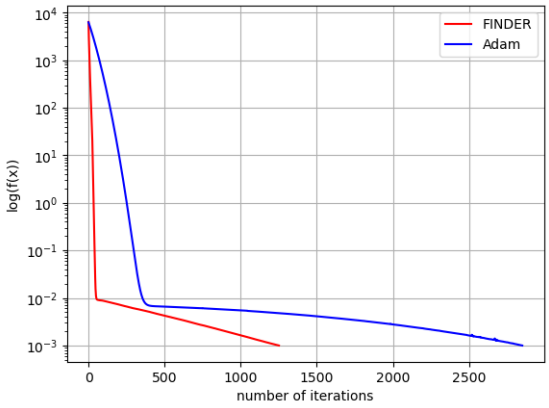

Figure 6. Minimization of 5000-dimensional Rosenbrock function

complex patterns and make decisions based on the input data. In this context, datasets like MNIST [37] & CIFAR10 [38] provide a standardized platform for comparing the efficacy of different algorithms. In our implementation, a single batch of training data has been fed to the networks as input. Gradient computation is presently done through the automatic

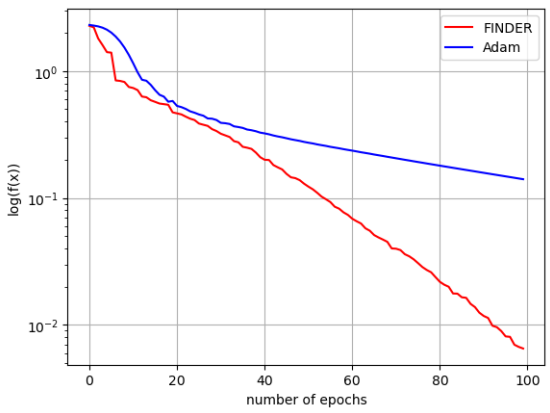

Figure 7. Minimization of the loss function for MNIST problem; $N = 575050$

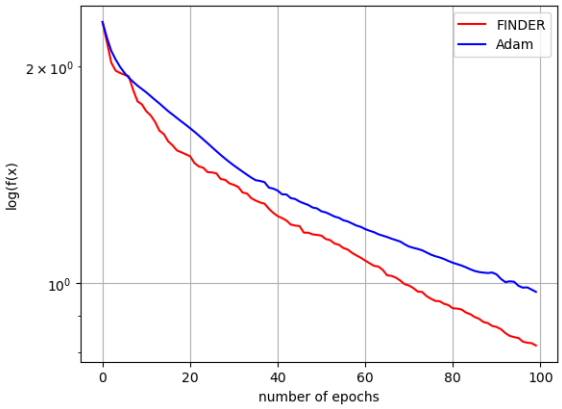

Figure 8. Minimization of the loss function for CIFAR10 classification problem; $N = 775190$

differentiation toolkit 'Autograd' of PyTorch [39]. We have trained for 100 epochs without prescribing any ε_{tol} in these problems.

1) MNIST classification problem: The goal is to train a DNN to correctly identify the digit in each image from the MNIST dataset. We attempt to train a DNN of [764-512-256-128-64-10] architecture with Rectified Linear Unit ($relu$) activation in hidden layers and $softmax$ activation in the output layer. Fig. 7 shows a comparison with the Adam optimizer after training for 100 epochs.

2) CIFAR10 classification problem: The goal here is to correctly categorize each image from the CIFAR10 dataset based on the object captured in that image. We select a Convolutional Neural Network (CNN) as our model, comprising a single convolutional layer featuring 16 kernels and filters sized 5×5. This layer undergoes a max-pooling operation with $relu$ activation, followed by a feed-forward neural network with 240, 84, and 10 neurons; each hidden layer employs $\tanh$ activation and the output layer employs $softmax$ activation. The loss function evolutions over 100 epochs are shown in Fig. 8

C. Training Physics Informed Neural Networks (PINNs)

Motivated by the smoothness of the functional approximants and their derivatives in neural networks, PINNs use physics information, such as squared residuals of the governing ordinary or partial differential equations (ODEs/PDEs) and initial/boundary conditions (ICs/BCs) to construct the loss function. This loss function is then minimized to determine the network parameters. We solve the Burgers' equation, a plane-strain elasticity problem, and a strain gradient plasticity (SGP) problem in this sub-section. We have used (see Eqn. (11)) $\zeta_1 = \zeta_2 = 10^{-4}$ for the first two problems. The SGP problem involves an innovative use of ζ_1, ζ_2, discussed later.

1) Burgers' Equation: This PDE is used to model shock wave phenomena. In situations characterized by low viscosity parameters, this equation may lead to a shock formation, a phenomenon notoriously difficult to resolve using traditional numerical methods. We specifically solve the following problem [11] using PINN:

$$\begin{aligned} u_{,t} + uu_{,x} &= \frac{0.01}{\pi} u_{,xx} \\ \text{Initial condition (IC): } u(0,x) &= -\sin(\pi x) \\ \text{Boundary conditions (BCs): } u(t,-1) &= u(t,1) = 0 \end{aligned} \tag{16}$$

where $x \in [-1,1]$ and $t \in [0,1]$. The field variable u is treated as the output of a DNN with the architecture $[2, 20, 20, 20, 20, 20, 20, 20, 20, 20, 1]$. The DNN has $\tanh$ activation in its hidden layers and no activation in the output layer. The input (x,t) to this network is a set of interior points, boundary points ($x = -1$ and $x = 1$), and initial points at $t = 0$ in the domain $[-1,1] \times [0,1]$. 10000 interior points are drawn from the domain using the distribution $\mathcal{U}(-1,1) \times \mathcal{U}(0,1)$. 50 initial points and 50 boundary points (25 each for boundaries at $x = -1$ and $x = 1$) are also drawn with uniform distribution. The loss function is the sum of initial loss, boundary loss, and residue loss. The initial loss is the mean squared loss of predicted u at initial points against the prescribed IC. Similarly, the boundary loss is the mean squared loss of u predicted at boundary against prescribed BCs. The residue loss is the mean squared loss of PDE residue evaluated at the interior points. Fig. 9 shows the comparative evolutions of the loss via FINDER and Adam.

2) 2D Elasticity: In the mechanics of solids, the 2D plain strain elasticity problem describes the deformation of a two-dimensional body under external forces, assuming no deformation in the third dimension. We attempt to solve one such problem as given in [12] using PINN. The governing PDEs,

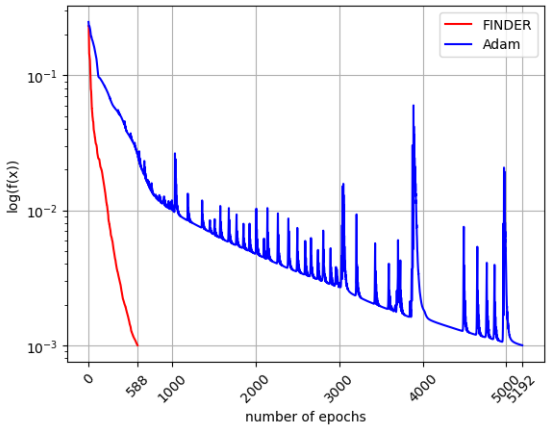

Figure 9. Minimization of the loss function for solving Burgers' equation; $N = 3441$

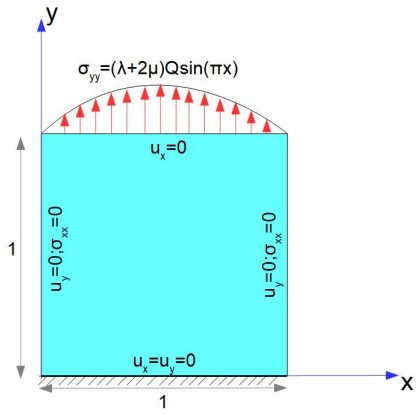

Figure 10. A schematic of domain, loading, and BCs for the elasticity problem

constitutive laws, and kinematic relations are as follows:

$$
\begin{aligned}
\text{Momentum balance laws: } & \sigma_{xx,x} + \sigma_{xy,y} + b_x = 0 \\
& \sigma_{yx,x} + \sigma_{yy,y} + b_y = 0 \\
& \sigma_{xy} = \sigma_{yx} \\
\text{Constitutive laws: } & \sigma_{xx} = (\lambda + 2\mu)\varepsilon_{xx} + \lambda\varepsilon_{yy} \\
& \sigma_{yy} = (\lambda + 2\mu)\varepsilon_{yy} + \lambda\varepsilon_{xx} \\
& \sigma_{xy} = 2\mu\varepsilon_{xy} \\
\text{Kinematic relations: } & \varepsilon_{xx} = u_{,x}; \varepsilon_{yy} = v_{,y} \\
& \varepsilon_{xy} = \frac{1}{2}\left(u_{,y} + v_{,x}\right) \\
b_x = & \lambda(4\pi^2 \cos(2\pi x)\sin(\pi y) - \pi\cos(\pi x)\mathcal{Q}y^3) \\
& +\mu(9\pi^2 cos(2\pi x)sin(\pi y) - \pi cos(\pi x)\mathcal{Q}y^3) \\
b_y = & \lambda(-3sin(\pi x)\mathcal{Q}y^2 + 2\pi^2 sin(2\pi x)cos(\pi y)) \\
& +\mu(-6sin(\pi x)\mathcal{Q}y^2 + 2\pi^2 sin(2\pi x)cos(\pi y)) \\
& +\mu\pi^2 sin(\pi x)\mathcal{Q}y^4/4
\end{aligned}
\tag{17}
$$

where σ denotes the Cauchy stress tensor, b the resultant body force, u, v displacements along x, y directions and ε the infinitesimal strain tensor. We also take $\lambda = 1$, $\mu = 0.5$ and $\mathcal{Q} = 4$. A schematic of the BCs is shown in Fig. 10. The PINN that we employ has separate DNNs for $u, v, \sigma_{xx}, \sigma_{yy}$ and σ_{xy} with $[2, 100, 100, 100, 100, 1]$ architecture. The input (x, y) to the model is a set of boundary and interior points. The DNNs have $\tanh$ activation for all the layers except for the last. The domain is discretized as a 100×100 grid resulting in 9604 interior nodes and 400 boundary nodes. The loss function is the sum of residue and boundary losses. The residue loss is the mean squared error of the residues of PDEs, constitutive laws and kinematic equations evaluated at the interior points. The boundary loss is the mean squared error of displacements/tractions predicted at boundary points against the prescribed BCs. Evolutions of the loss function via the two optimizers are plotted in Fig. 11.

3) Strain Gradient Plasticity: Strain gradient plasticity is a mathematical model that acknowledges the significance of plastic strain gradients and hence intrinsic (microscopic) length scales in modeling plastic deformation in solids, e.g.

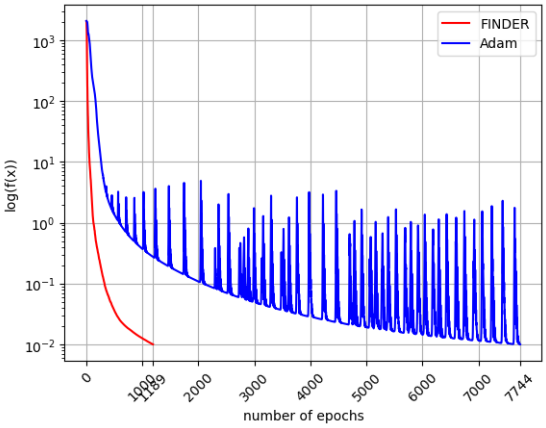

Figure 11. Minimization of the loss function for the 2D elasticity problem; $N = 153505$

metals. Classical plasticity theories, which do not incorporate such length scales, exhibit spurious mesh-dependence of solutions (viz. shear bands) when implemented numerically, thereby limiting our understanding of the phenomena. These include strain softening/hardening, where some materials, upon stretching, lose strength while others gain; size effect, where smaller material samples exhibit higher strength than the bigger ones; inhomogeneous plastic flow, where materials do not stretch evenly; cases of very large plastic deformation etc. These features are far too macroscopically manifest to be efficiently simulated using molecular dynamics. This creates a gap between computational feasibility and theoretical modelling. With its multi-scale features, the strain gradient plasticity model [40] has been an attempt to bridge the gap. The multi-scale and extremely stiff character of the model is captured by the governing macroscopic and microscopic balance laws. Specifically, the microscopic balance is designed to evolve the plastic strain rate $\dot{\gamma}^p$ with τ^p denoting the stress conjugate. This balance law also involves a term $\nabla\dot{\gamma}^p$ – the gradient of plastic strain rate, and the k^p is the associated stress conjugate. Attempts at directly solving these partial differential equations (balance laws) via the finite element method have generally been unsuccessful. A PINN-based approach, on the other hand, has recently shown promise [41].

We now use the two optimizers to solve one such stiff displacement boundary value problem (see Appendix A of [42]) via PINN and compare their relative performance. The resulting loss function has multiple local minima in proximity, leading to a possible variation in results with different initializations of network parameters. The governing equations and ICs/BCs are as follows:

$$
\begin{aligned}
\text{Macroforce balance: } & \tau_{,y} = 0 \\
\text{Microforce balance: } & \tau = \tau^p - k^p_{,y} \\
\text{Constitutive law: } & \tau = \mu(u_{,y} - \gamma^p) \\
\text{Microstress: } & \tau^p = S_0 \left(\frac{d^p}{d_0}\right)^m \frac{\dot{\gamma}^p}{d^p} \\
\text{Gradient microstress: } & k^p = SL^2\gamma^p_{,y} + S_0 l^2 \left(\frac{d^p}{d_0}\right)^m \frac{\dot{\gamma}^p_{,y}}{d^p} \\
\text{Resistance to plastic flow: } & \dot{S} = H(S)d^p \ ; \ S(y,0) = S_0 \\
\text{Effective flow rate: } & d^p = \sqrt{|\ \dot{\gamma}^p\ |^2 + l^2\ |\ \dot{\gamma}^p_{,y}\ |^2} \\
\text{Imposed shear strain: } & E(t) = \frac{u^+(t)}{h} \\
\text{Displacement BCs: } & u(0,t) = 0;\ u(h,t) = u^+(t) \\
\text{Plastic strain rate BCs: } & \dot{\gamma}^p(0,t) = \dot{\gamma}^p(h,t) = 0 \\
\text{Displacement ICs: } & u(y,0) = 0 \\
\text{Plastic strain ICs: } & \gamma^p(y,0) = 0
\end{aligned}
\tag{18}
$$

where τ denotes shear stress and $y \in [0, h]$. The problem involves a long (height h along y), thin body (infinite in x and z) undergoing plane-strain shearing along x, so that the displacement $u := u(y,t)$. We use the initial yield strength $S_0 = 100$ MPa, shear modulus $\mu = 100$ GPa, hardening/softening function $H = 0$, reference flow rate $d_0 = 0.1$, rate sensitivity parameter $m = 0.02$, energetic length scale $L = 10$, dissipative length scale $l = 0$ and $h = 10$. We employ separate DNNs for u and γ^p, each with [2-64-128-64-1] architecture and $\tanh$ activation in the hidden layers. The BCs and ICs have been 'hard-coded' by modifying the network output itself. We also scale the inputs t, y as well the outputs u, γ^p before constructing the loss function, which is then just the mean squared error residue of the given equations evaluated at 930 collocation points, uniformly distributed in the domain. The strategy to deal with multiple local minima is to use larger values of $\zeta_1, \zeta_2 : \zeta_1 = \zeta_2 = 0.1$ to begin with, while reducing them to 10^{-4} once an appropriately low loss value is reached. Fig. 12 shows evolutions of the loss function via FINDER and Adam.

Table III further summarizes the comparative performances of FINDER and Adam, in training DNNs for the stated classification and PINN examples. Though comparable, the GPU times taken by FINDER are generally a bit higher, mainly caused by the computational overhead of simulating multiple particles within a Monte Carlo framework. We expect that a GPU-based explicit parallelization of the multi-particle simulation would render FINDER even more computationally efficient in future.

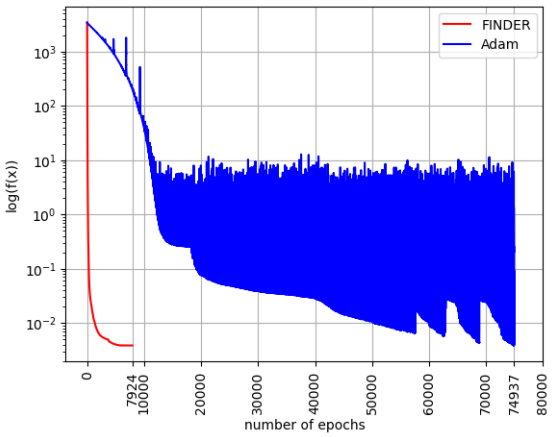

Figure 12. Minimization of the loss function for the strain gradient plasticity problem; $N = 33666$

Table III
COMPARING FINDER VIS-Á-VIS ADAM IN NEURAL NETWORKS BASED IMPLEMENTATIONS

Problem	N	Loss		Time (in sec.)		Iteration Count	
		FINDER	Adam	FINDER	Adam	FINDER	Adam
Burgers'	3441	10^{-3}	10^{-3}	59.57	65	588	5192
Plasticity	33666	3.8×10^{-3}	3.8×10^{-3}	766.73	615.60	7924	74937
2D elasticity	153505	10^{-2}	10^{-2}	485.10	420.45	1189	7744
MNIST	575050	6.5×10^{-3}	1.4×10^{-1}	847.72	708.4	100	100
CIFAR10	775190	8.2×10^{-1}	9.7×10^{-1}	1821.2	1507	100	100

IV. CONCLUSIONS

Motivated in part by the fast convergence of quasi-Newton schemes over a locally convex part of an objective or loss function, we have used a measure-theoretic means to a stochastic mimicking of the inverse Hessian en route to a novel optimization scheme, FINDER. In the process, we have used certain simplifying approximations that enable the method to linearly scale with dimension. Since a probabilistic foundation of the method also implies a noise-driven exploratory character, FINDER could be thought of as an attempt to be astride the best of both worlds. The measure-theoretic feature, which enables a distributional – and not pointwise – interpretation of quantities such as the inverse Hessian and even the solution, derives from the stochastic filtering route, wherein the gradient of the loss function is driven to a zero-mean martingale (and not to zero, as with most deterministic methods) via a change of measures. Among others, this distributional interpretation of quantities comes in handy while training physics-informed deep networks wherein the presence of higher order derivatives may render the loss function spiky thereby making a pointwise interpretation of the gradient of this function less meaningful. This probably explains why Adam, which uses the classical notion of gradient, exhibits (unlike FINDER) spiky evolutions for PINN implementations.

As with most other optimizers, FINDER too has limitations and issues in its current form. While it shows a consistently faster local convergence, the noise-assisted avoidance of local

traps in FINDER appears to be less effective than the exploitation of the so-called variance of the stochastic gradient in Adam. Borrowing such concepts from Adam or using Levy noises [43] with or without annealing to avoid local entrapment could be interesting future extensions. Curiously enough, the gain matrix multiplying the vector-valued innovation (i.e. the error process or the gradient of the loss function being driven to a zero-mean martingale) to form the update term is then identifiable as the mimicked inverse Hessian. Moreover, based on the structure of our process and observation dynamics that form the backbone of the filtering problem, we may take the covariance of the estimated states to be diagonally dominant. This in turn would justify the diagonal approximation of the gain matrix by just neglecting the low-rank cross-covariance component. The limited results presented in this work show the promise of the method, perhaps uniquely so for a Monte Carlo search, across a spectrum of continuous optimization problems. The authors invite the readership to explore this optimizer for any continuous optimization requirements.

Supplementary Material

Python codes including a plug-and-play version are on GitHub; https://github.com/FINDER-optimizer/FINDER.

References

[1] D. G. Luenberger, Y. Ye *et al.*, *Linear and nonlinear programming*. Springer, 1984, vol. 2.

[2] S. P. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge University Press, 2004.

[3] S. Ruder, "An overview of gradient descent optimization algorithms," 2017.

[4] M. Afshar and A. Faramarzi, "Size optimization of truss structures by cellular automata," *Journal of Computer Science and Engineering*, vol. 3, no. 1, pp. 1–9, 2010.

[5] A. Faramarzi and M. Afshar, "A novel hybrid cellular automata–linear programming approach for the optimal sizing of planar truss structures," *Civil Engineering and Environmental Systems*, vol. 31, no. 3, pp. 209–228, 2014.

[6] D. P. Bertsekas, "Projected Newton methods for optimization problems with simple constraints," *SIAM Journal on control and Optimization*, vol. 20, no. 2, pp. 221–246, 1982.

[7] J. Ford and I. Moghrabi, "Multi-step quasi-Newton methods for optimization," *Journal of Computational and Applied Mathematics*, vol. 50, no. 1-3, pp. 305–323, 1994.

[8] M. N. Thapa, "Optimization of unconstrained functions with sparse Hessian matrices—Quasi-Newton methods," *Mathematical Programming*, vol. 25, no. 2, pp. 158–182, 1983.

[9] A. F. Izmailov and M. V. Solodov, *Newton-type methods for optimization and variational problems*. Springer, 2014, vol. 3.

[10] B. T. Polyak, "Newton's method and its use in optimization," *European Journal of Operational Research*, vol. 181, no. 3, pp. 1086–1096, 2007.

[11] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations," *arXiv preprint arXiv:1711.10561*, 2017.

[12] E. Haghighat, M. Raissi, A. Moure, H. Gomez, and R. Juanes, "A physics-informed deep learning framework for inversion and surrogate modeling in solid mechanics," *Computer Methods in Applied Mechanics and Engineering*, vol. 379, p. 113741, 2021.

[13] H. Robbins and S. Monro, "A stochastic approximation method," *The annals of mathematical statistics*, pp. 400–407, 1951.

[14] M. Srinivas and L. M. Patnaik, "Genetic algorithms: A survey," *computer*, vol. 27, no. 6, pp. 17–26, 1994.

[15] D. Wang, D. Tan, and L. Liu, "Particle swarm optimization algorithm: an overview," *Soft computing*, vol. 22, no. 2, pp. 387–408, 2018.

[16] N. Hansen, "The CMA evolution strategy: a comparing review," *Towards a new evolutionary computation: Advances in the estimation of distribution algorithms*, pp. 75–102, 2006.

[17] R. A. Rutenbar, "Simulated annealing algorithms: An overview," *IEEE Circuits and Devices magazine*, vol. 5, no. 1, pp. 19–26, 1989.

[18] R. Storn and K. Price, "Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces," *Journal of global optimization*, vol. 11, pp. 341–359, 1997.

[19] D. Roy and G. V. Rao, *Elements of classical and geometric optimization*. CRC Press, 2024.

[20] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[21] B. Oksendal, *Stochastic differential equations: an introduction with applications*. Springer Science & Business Media, 2013.

[22] M. Saxena, S. Sarkar, and D. Roy, "Inverse Hessian by stochastic projection and application to system identification in nonlinear mechanics of solids," *International Journal of Non-Linear Mechanics*, p. 104762, 2024.

[23] H. J. Kushner, "On the differential equations satisfied by conditional probablitity densities of Markov processes, with applications," *Journal of the Society for Industrial and Applied Mathematics, Series A: Control*, vol. 2, no. 1, pp. 106–119, 1964.

[24] R. L. Stratonovich, R. N. McDonough, N. B. McDonough *et al.*, "Conditional Markov processes and their application to the theory of optimal control," *IEEE Transactions on Automatic Control 13(1):137-138*, 1968.

[25] S. Sarkar, S. R. Chowdhury, M. Venugopal, R. M. Vasu, and D. Roy, "A Kushner–Stratonovich Monte Carlo filter applied to nonlinear dynamical system identification," *Physica D: Nonlinear Phenomena*, vol. 270, pp. 46–59, 2014.

[26] D. Roy and G. V. Rao, *Stochastic dynamics, filtering and optimization*. Cambridge University Press, 2017.

[27] M. Saxena, S. Sarkar, and D. Roy, "A microstructure-sensitive and derivative-free continuum model for composite materials: applications to concrete," *International Journal of Solids and Structures*, p. 112051, 2022.

[28] D. Bertsekas and R. Gallager, *Data networks*. Athena Scientific, 2021.

[29] G. Milstein and J. Schoenmakers, "Monte Carlo construction of hedging strategies against multi-asset European claims," *Stochastics and Stochastic Reports*, vol. 73, no. 1-2, pp. 125–157, 2002.

[30] Z. Yao, A. Gholami, S. Shen, K. Keutzer, and M. W. Mahoney, "ADAHESSIAN: An Adaptive Second Order Optimizer for Machine Learning," *AAAI (Accepted)*, 2021.

[31] J. Nocedal and S. J. Wright, *Line Search Methods*. New York, NY: Springer New York, 2006, pp. 30–65. [Online]. Available: https://doi.org/10.1007/978-0-387-40065-5_3

[32] G. N. Milstein, *Numerical integration of stochastic differential equations*. Springer Science & Business Media, 2013, vol. 313.

[33] G. Evensen, "The ensemble Kalman filter: Theoretical formulation and practical implementation," *Ocean dynamics*, vol. 53, pp. 343–367, 2003.

[34] C. Schillings and A. M. Stuart, "Analysis of the Ensemble Kalman Filter for Inverse Problems," *SIAM Journal on Numerical Analysis*, vol. 55, no. 3, pp. 1264–1290, 2017. [Online]. Available: https://doi.org/10.1137/16M105959X

[35] F. C. Klebaner, *Introduction to stochastic calculus with applications*. World Scientific Publishing Company, 2012.

[36] V. Plevris and G. Solorzano, "A collection of 30 multidimensional functions for global optimization benchmarking," *Data*, vol. 7, no. 4, p. 46, 2022.

[37] Y. LeCun, C. Cortes, and C. Burges, "MNIST handwritten digit database," *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, vol. 2, 2010.

[38] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.

[39] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.

[40] M. E. Gurtin and L. Anand, "A theory of strain-gradient plasticity for isotropic, plastically irrotational materials. Part I: Small deformations," *Journal of the Mechanics and Physics of Solids*, vol. 53, no. 7, pp. 1624–1649, 2005.

[41] A. Tyagi, U. Suman, M. Mamajiwala, and D. Roy, "Physics Informed Deep Learning for Strain Gradient Continuum Plasticity," *arXiv preprint arXiv:2408.06657*, 2024.

[42] S. P. Lele, "On a class of strain gradient plasticity theories: formulation and numerical implementation," Ph.D. dissertation, Massachusetts Institute of Technology, 2008.

[43] D. Applebaum, *Lévy processes and stochastic calculus*. Cambridge University Press, 2009.