Randomized Newton’s Method for Solving Differential Equations Based on the Neural Network Discretization

Qipin Chen · Wenrui Hao

Received: 18 July 2021 / Revised: 5 May 2022 / Accepted: 10 May 2022 / Published online: 2 July 2022
© The Author(s), under exclusive licence to Springer Science+Business Media, LLC, part of Springer Nature 2022

Abstract
We develop a randomized Newton’s method for solving differential equations, based on a fully connected neural network discretization. In particular, the randomized Newton’s method randomly chooses equations from the overdetermined nonlinear system resulting from the neural network discretization and solves the nonlinear system adaptively. We theoretically prove that the randomized Newton’s method has a quadratic convergence locally. We also apply this new method to various numerical examples, from one to high-dimensional differential equations, to verify its feasibility and efficiency. Moreover, the randomized Newton’s method can allow the neural network to “learn” multiple solutions for nonlinear systems of differential equations, such as pattern formation problems, and provides an alternative way to study the solution structure of nonlinear differential equations overall.

Keywords Random Newton’s method · Convergence analysis · Differential equations · Neural networks

1 Introduction
Partial differential equations (PDEs) have been widely used in physics [8], biology [15, 26], and engineering [1], and have played important roles in modeling from bacterial growth [2] to complex fluid structure interactions [37]. Thus, computing numerical solutions for PDEs has been a research area of long-standing importance in the computational mathematics community. Some efficient numerical methods have already been developed for solving PDEs: for example, the finite difference method [46, 58], the finite element method [32, 60] and the spectral method [22, 43, 44, 59]. Moreover, several numerical techniques, such as the multigrid methods [3, 61], domain decomposition methods [49, 51], and preconditioning techniques have been used to speed up computations and improve computational efficiency,
especially in two- and three-dimensional problems. However, there are still two challenges facing the numerical PDE community: 1) traditional methods become inefficient for solving high-dimensional PDEs, due to such PDEs’ dramatic explosion of grid points. While the sparse grid method [35] has been used to solve high-dimensional PDEs [29, 45] by constructing multidimensional, multilevel basis, this method still becomes inefficient when the dimension is particularly high, owing to the logarithmic term in the complexity [4]; 2) traditional methods are inadequate for computing the multiple solutions that nonlinear PDEs have. While the deflation method [14] has been used to compute the distinct solutions, it can not be guaranteed to find all the possible solutions, due to the artificial singularities that this method introduces. And while homotopy methods coupled with domain decomposition [18], multigrid and spectral methods [54] have been developed for computing multiple solutions, all of these methods become time-consuming for high-dimensional nonlinear PDEs.

Recently, machine learning techniques have been developed for solving PDEs, since machine learning has been experiencing great success in various fields related to artificial intelligence (e.g., computer vision [16], natural language processing [31]). The application of machine learning techniques to PDE problems, however, is usually not straightforward. Some approaches have included the following: The DGM net [48], based on a fully connected network, has been developed to solve high-dimensional PDEs by minimizing the $L^2$ norm, both in the domain and on the boundary; Using ReLU deep neural networks (DNNs) has been developed in [20] to solve differential equations by exploring the relationship between DNNs with rectified linear unit (ReLU) function and continuous piecewise linear functions, from finite element method; A deep learning-based approach [12] has been developed to solve high-dimensional parabolic PDEs by reformulating PDEs as backward stochastic differential equations; Machine learning techniques have also been used to learn governing differential equations by empirical data [41, 55, 57]. All these approaches follow the machine learning optimization framework by minimizing the loss functions constructed according to different approaches. However, these loss functions are usually highly non-convex, and the optimization process is prone to being trapped by some local minima so that the training process takes a long time to stabilize. Thus recent development in Newton’s method brings a faster convergence for the large-scale optimization arising from machine learning [39].

In this paper, thus, we combine two different approaches: namely, we use the neural network to discretize differential equations while we solve a system of nonlinear equations instead of the optimization problem. More specifically, in order to solve the overdetermined system of nonlinear equations that a fully connected neural network discretization yields, we introduce a randomized Newton’s method, which randomly chooses equations from the overdetermined system and solves the nonlinear system adaptively. The remainder of the paper is structured as follows: In Sect. 2, we show the problem setup, address the infeasibility of the traditional collocation method, and introduce the overdetermined nonlinear system that results from neural network discretization. Then, in Sect. 3, we present the detailed algorithm and the convergence analysis of the randomized Newton’s method. Sect. 4 presents several numerical examples, ranging from linear to nonlinear differential equations on both one- and high-dimensional cases, to show the efficiency and feasibility of the randomized Newton’s method. Moreover, we demonstrate with an application to pattern formation how this approach can be used to “learn” multiple solutions by coupling the randomized Newton’s method with neural network discretizations.
2 The Problem Setup

We consider the following Laplace’s equation

\[
\begin{cases}
-\Delta u = f(u) & \text{in } \Omega \\
u = u_0 & \text{on } \partial\Omega,
\end{cases}
\]

where \( \Omega \subset \mathbb{R}^d \) and \( \partial\Omega \) is the boundary of the domain \( \Omega \). We have known that any critical point of the following energy functional is the solution of the Laplace’s Eq. (2.1) [13]

\[
E(u) = \int_\Omega \frac{1}{2} |\nabla u|^2 - F(u) \, dx,
\]

where \( F'(x) = f(x) \) (see example 3 on page 435 in [13]) and \( u \) belongs to the admissible set

\[
\mathcal{A} := \{ u \in C^2(\bar{\Omega}) | u = u_0 \text{ on } \partial\Omega \}.
\]

We apply an \((n+1)\) -layer neural network \( U(x; \theta) \) to approximate the solution to system (2.1), \( u(x) \), namely,

\[
U(x; \theta) = W_{l-1}W_l \sigma(W_{l-1} \cdots \sigma(W_2 \sigma(W_1 x + b_1) + b_2) \cdots + b_{n-1}) + b_n,
\]

where \( \{W_1\}_{i=1}^n \) and \( \{b_i\}_{i=1}^n \) are the weights and bias of the network, and \( \sigma \) is the activation function such as the \( \sin \) function, sigmoid function \( \frac{e^x}{1+e^x} \), or ReLU \( \max(x,0) \) [48]. For simplicity, we denote the set of all parameters as \( \theta = \{W_1, \cdots, W_n, b_1, \cdots, b_n\} \) and the number of all parameters as \( |\theta| \). Then, optimization techniques are successfully used to solve the resulting minimization problem based on (2.2)

\[
\min E(\theta) = \int_\Omega \frac{1}{2} |\nabla U(x; \theta)|^2 - F(U(x; \theta)) \, dx + \int_{\partial\Omega} |U(x; \theta) - u_0(x)|^2 \, dS,
\]

where the second term represents the boundary conditions, which can be Dirichlet or Neumann [40]. The numerical challenges of solving the optimization problem (2.5) is that 1) the computational cost of function evaluations can be very large for high dimensional cases [48], and 2) the solutions are more likely to be trapped by some local minima, since the objective function \( E(\theta) \) is usually highly non-convex [56]. In order to avoid these numerical difficulties, we solve the equation (2.1) directly by using the discretization of (2.4). Therefore, we get the following system of nonlinear equations:

\[
F(\theta) = \begin{cases}
\Delta U(x_i; \theta) + f(U(x_i; \theta)) = 0 & i = 1, \cdots, N \\
U(x_j; \theta) - u_0(x_j) = 0 & j = 1, \cdots, M
\end{cases}
\]

where \( F : \mathbb{R}^{|\theta|} \rightarrow \mathbb{R}^{N+M} \), \( x_i \) and \( x_j \) are sample points on \( \Omega \) and \( \partial\Omega \) respectively. As the number of neurons goes to the infinity, the numerical solution obtained by neural network converges to the real solution of (2.1) [62]. The collocation method [42] has been normally used to solve the resulting nonlinear system (2.6) by taking \( N+M = |\theta| \), that is, the number of sample points chosen is the same as the number of variables. However, the collocation method can not be used to solve the nonlinear system arising from neural network discretization due to its highly nonlinearity. We will use a simple example to illustrate this reason by considering

\[
\begin{cases}
u_{xx} = -4\pi^2 \sin(2\pi x) & \text{on } (0, 1), \\
u(0) = 0 \text{ and } u(1) = 0.
\end{cases}
\]
We apply a one-hidden-layer neural network discretization, namely,

\[ U(x; \theta) = W_2\sigma(W_1x + b_1) + b_2, \quad (2.8) \]

where \( \theta = \{W_1, W_2, b_1, b_2\} \in \mathbb{R}^4 \). Here we choose the activation function \( \sigma(x) \) simply as \( \sin(x) \), then \( \theta = \{2\pi, 1, 0, 0\} \) is the real solution. Since the number of parameters \( |\theta| \) is 4, we use the collocation method to sample 2 points, \( \{x_1, x_2\} \), in the domain \( (0, 1) \) and 2 points on the boundary. Thus the discretization system becomes:

\[
\begin{pmatrix}
F_1(\theta) \\
F_2(\theta) \\
F_3(\theta) \\
F_4(\theta)
\end{pmatrix} =
\begin{pmatrix}
-W_1^2W_2 \sin(W_1x_1 + b_1) + 4\pi^2 \sin(2\pi x_1) \\
-W_1^2W_2 \sin(W_1x_2 + b_1) + 4\pi^2 \sin(2\pi x_2) \\
W_2 \sin(b_1) + b_2 \\
W_2 \sin(W_1 + b_1) + b_2
\end{pmatrix} = 0. \quad (2.9)
\]

By solving \( (F_1(\theta), F_3(\theta)) = (0, 0) \) for \( W_2 \) and \( b_2 \) in term of \( W_1 \) and \( b_1 \), we have

\[
\begin{cases}
W_2 = \frac{4\pi^2 \sin(2\pi x_1)}{W_1^2 \sin(W_1x_1 + b_1)}, \\
b_2 = -W_2 \sin(b_1) = -\frac{4\pi^2 \sin(2\pi x_1)}{W_1^2 \sin(W_1x_1 + b_1)} \sin(b_1).
\end{cases} \quad (2.10)
\]

Therefore, a simplified system of (2.9) is written as

\[
\begin{pmatrix}
F_2(W_1, b_1) = -\frac{4\pi^2 \sin(2\pi x_1) \sin(W_1x_2 + b_1)}{\sin(W_1x_1 + b_1)} + 4\pi^2 \sin(2\pi x_2) \\
F_4(W_1, b_1) = \frac{4\pi^2 \sin(2\pi x_1) \sin(W_1 + b_1)}{W_1^2 \sin(W_1x_1 + b_1)} - \frac{4\pi^2 \sin(2\pi x_1)}{W_1^2 \sin(W_1x_1 + b_1)} \sin(b_1)
\end{pmatrix} = 0. \quad (2.11)
\]

We chose three groups of collocation points \( \{x_1, x_2\} \):

\[
CL_1 = \{0.1, 0.8\}, \quad CL_2 = \{0.8, 0.9\}, \quad CL_3 = \{0.1, 0.2\}.
\]

Then we employed Newton’s method to solve (2.11) with an initial guess \( (W_1^0 = 1, b_1^0 = 1) \). The solutions of nonlinear systems with three groups of collocation points \( CL_i \) (i=1,2,3) are shown in Fig. 1 (upper left): Newton’s method finds the real solution for the nonlinear system with collocation points \( CL_3 \) while it delivers “fake solutions” for other two systems. The reason is that there might be multiple solutions of (2.11) for any given sample points, although all the systems share one solution which corresponds to the real solution \( u(x) \). Figure 1 shows the multiple roots for different systems with three groups of collocation points.

Therefore, we need to sample many more points than the number of variables, namely, \( N + M > |\theta| \), so that the system (2.6) does not contain “fake solutions” with probability one [27, 50, 53]. Thus the system (2.6) becomes an overdetermined system, since we need many more equations than variables. In this paper, we develop an efficient randomized Newton’s method to solve the overdetermined nonlinear system arising from the neural network discretization of differential equations.
3 Randomized Newton’s Method

We write the overdetermined system of nonlinear equations in the following general form

\[
F(\theta) = \begin{bmatrix}
f_1(\theta) \\
f_2(\theta) \\
\vdots \\
f_n(\theta)
\end{bmatrix} = 0,
\]  

(3.1)

where \( \theta \in \mathbb{R}^m \) and \( F : \mathbb{R}^m \to \mathbb{R}^n \) \((n \gg m)\). In numerical algebraic geometry, the overdetermined system shown in (3.1) can be solved by converting to a square system via solving \( G(\theta) = A_{m \times n} F \), where \( A_{m \times n} \) is an randomized matrix [11]. The drawback of this method is that 1) even the evaluation of this augmented system \( G \) could be problematic for large-scale systems, and 2) the number of sample points might be adaptive, so that constructing the random matrix \( A_{m \times n} \) each time would be time-consuming.
3.1 Algorithm

In this paper, we develop a randomized Newton’s method, namely,

$$\theta^{k+1} = \theta^k - (\nabla \tilde{F}(\theta^k))^{-1} \tilde{F}(\theta^k),$$  \hspace{1cm} (3.2)

where $\tilde{F}$ is a system with randomly chosen $m$ equations from $F$, $\nabla \tilde{F}$ is the Jacobian matrix of $\tilde{F}$ with respect to $\theta$. If $\nabla \tilde{F}(\theta^k)$ is singular, then the Gauss-Newton’s method will be employed.

The randomized Newton’s method can be rewritten with a random variable $\xi$ on a probability space $(\Omega, F, P)$:

$$\xi : \Omega \to \Gamma,$$  \hspace{1cm} (3.3)

where $\Gamma$ is a set with all the combinations of $m$ numbers out of $\{1, 2, \ldots, n\}$. Since $|\Gamma| = \binom{n}{m}$, we denote

$$\Gamma = \{\gamma_1, \gamma_2, \ldots, \gamma_{\binom{n}{m}}\},$$  \hspace{1cm} (3.4)

and assume the random variable $\xi$ follows the uniform distribution, namely, $P(\xi = \gamma_i) = \frac{1}{\binom{n}{m}}$ for $1 \leq i \leq \binom{n}{m}$. Then we rewrite $\tilde{F}$ as

$$\tilde{F} = F(\theta, \gamma_s) := \begin{pmatrix} f_{s_1}(\theta) \\ f_{s_2}(\theta) \\ \vdots \\ f_{s_m}(\theta) \end{pmatrix},$$  \hspace{1cm} (3.5)

where $\gamma_s = \{s_1, \ldots, s_m\}$ and $\{s_1, \ldots, s_m\} \subset \{1, \ldots, n\}$. Similarly, the randomized Newton’s method is rewritten as

$$\theta^{k+1} = \theta^k - (\nabla F(\theta^k))^{-1} \cdot F(\theta^k, \xi_k).$$  \hspace{1cm} (3.6)

**Remark:** We write the formula in (3.6) into a more general form as follows

$$\theta^{k+1} = \theta^k - \eta (\nabla F(\theta^k))^\dagger F(\theta^k) + \sqrt{\eta} R(\theta^k, \xi_k),$$  \hspace{1cm} (3.7)

where $\eta$ is the step-length usually determined by the trust region and the line search algorithms [36]. In the algorithm (3.6), we choose $\eta = 1$. Here $(\nabla F(\theta^k))^\dagger$ is the pseudoinverse of $\nabla F(\theta^k)$ defined by

$$(\nabla F(\theta))^\dagger F(\theta) = \frac{1}{|\Gamma|} \sum_{i=1}^{|\Gamma|} (\nabla F(\theta, \gamma_i))^{-1} F(\theta, \gamma_i),$$  \hspace{1cm} (3.8)

and

$$R(\theta^k, \xi_k) = \sqrt{\eta} ((\nabla F(\theta^k))^\dagger F(\theta^k) - (\nabla F(\theta^k, \xi_k))^{-1} F(\theta^k, \xi_k)).$$

Then the expectation of $R(\theta, \xi)$ is

$$\mathbb{E}[R(\theta, \xi)] = \sqrt{\eta} \left( (\nabla F(\theta))^\dagger F(\theta) - \frac{1}{|\Gamma|} \sum_{i=1}^{|\Gamma|} (\nabla F(\theta, \gamma_i))^{-1} F(\theta, \gamma_i) \right) = 0,$$

and the covariance matrix of $R(\theta, \xi)$ is

\[ \text{Cov}(R(\theta, \xi)) \]
Thus the SDE (3.10) is an approximation of (3.7) in the weak sense introduced in [28].

Then
\[
\|\nabla F(\theta, \xi)\| = \eta \left( E[(\nabla F(\theta, \xi))^{-1} F(\theta, \xi) ((\nabla F(\theta, \xi))^{-1} F(\theta, \xi))^T] - (\nabla F(\theta))\right) \right)^T.
\]

Theorem 1
Suppose \( F(\theta, \xi) \) is the solution to
\[
d\theta_t = b(\theta_t)dt + \sigma(\theta_t)dW_t, \quad \theta_0 = \theta_{\text{init}}.
\]
Then the Euler-Maruyama discretization [23] of (3.10) becomes
\[
\theta^{k+1} = \theta^k + b(\theta^k)\Delta t + \sqrt{\Delta t}\sigma(\theta^k)Z^k
\]
where \( Z^k \sim \mathcal{N}(0, I) \). If we choose \( \Delta t = \eta, b(\cdot) = -(\nabla F(\cdot))^\dagger F(\cdot), \) and \( \sigma(\cdot) = (\Sigma(\cdot))^\frac{1}{2}, \) then Eq. (3.11) becomes
\[
\theta^{k+1} = \theta^k - \eta(\nabla F(\theta^k))\nabla F(\theta^k) + \sqrt{\eta}(\Sigma(\theta^k))^\frac{1}{2} Z^k,
\]
Thus the SDE (3.10) is an approximation of (3.7) in the weak sense introduced in [28].

3.2 Convergence Analysis

Next we define tensor \( \nabla^2 F(\theta, \xi) \) as follows,
\[
[\nabla^2 F(\theta, \xi)]_{ijk} := [\nabla^2 f_i(\theta, \xi)]_{jk}, \quad i, j, k \in \{1, 2, \ldots, m\}
\]
where \( \nabla^2 f_i(\theta, \xi) \) is the Hessian matrix of \( f_i(\theta, \xi) \). Accordingly, we define the multiplication of the tensor with vectors as, for \( \forall a, b \in \mathbb{R}^m, \)
\[
[a^T \cdot \nabla^2 F(\theta, \xi) \cdot b]_i := \sum_{k=1}^{m} \sum_{j=1}^{m} a_j [\nabla^2 f_i(\theta, \xi)]_{jk} b_k, \quad i \in \{1, 2, \ldots, m\}.
\]

Then \( \|\nabla^2 F(\theta, \xi)\| \) is defined as \( \|\nabla^2 F(\theta, \xi)\| = \max_{i \in \{1, \ldots, m\}} \|\nabla^2 f_i(\theta, \xi)\| \). Thus we summarize the local convergence of the randomized Newton’s method in the following theorem.

Theorem 1 Suppose \( \theta^* \) is the solution to \( F(\theta) = 0 \). Assuming that \( \nabla^2 F(\cdot) \) is invertible and continuous, and that \( \nabla^2 A F(\cdot) \) is continuous in a small neighborhood of \( \theta^* \) for any permutation matrix \( A \in \mathbb{R}^{m \times n} \). If the initial guess, \( \theta^0, \) is inside a small neighborhood of \( \theta^0, \) then for scheme (3.6), we have
\[
\mathbb{E}(\|\theta^* - \theta^k\|) \leq C \left( \frac{1}{2} \right)^{2^k-1},
\]
which implies the quadratic convergence.

Proof We consider the Taylor expansion of \( F(\theta, \xi, k) \) at \( \theta^k \) for \( \theta^*: \)
\[
F(\theta^*, \xi_k) = F(\theta^k, \xi_k) + \nabla F(\theta^k, \xi_k) \cdot (\theta^* - \theta^k) + \frac{1}{2} (\theta^* - \theta^k)^T \cdot \nabla^2 F(\theta^k, \xi_k) \cdot (\theta^* - \theta^k),
\]
\[
(3.15)
\]
where \( t_k \) is between \( \theta^* \) and \( \theta^k \). Since that \( \theta^* \) is a solution to both \( F(\theta) = 0 \) and \( F(\theta, \xi_k) = 0 \), (3.15) becomes

\[
0 = F(\theta^*, \xi_k) = F(\theta^k, \xi_k) + \nabla F(\theta^k, \xi_k) \cdot (\theta^* - \theta^k) + \frac{1}{2} (\theta^* - \theta^k)^T \cdot \nabla^2 F(t_k, \xi_k) \cdot (\theta^* - \theta^k).
\]  

(3.16)

By multiplying \( (\nabla F(\theta^k, \xi_k))^{-1} \) on both sides of (3.16), we get

\[
(\nabla F(\theta^k, \xi_k))^{-1} \cdot F(\theta^k, \xi_k) + (\theta^* - \theta^k) = -\frac{1}{2} (\nabla F(\theta^k, \xi_k))^{-1} \cdot (\theta^* - \theta^k)^T \nabla^2 F(t_k, \xi_k)(\theta^* - \theta^k).
\]

(3.17)

By substituting (3.6) into the left-hand side of (3.17), we have

\[
\theta^* - \theta^k + 1 = -\frac{1}{2} (\nabla F(\theta^k, \xi_k))^{-1} \cdot (\theta^* - \theta^k)^T \nabla^2 F(t_k, \xi_k)(\theta^* - \theta^k).
\]

(3.18)

By taking the expectation on both sides of (3.18) with respect to \( \xi_0, \ldots, \xi_{k-1} \), we obtain

\[
\mathbb{E}(\|\theta^* - \theta^k+1\|) = \frac{1}{2} \mathbb{E}(\|\nabla F(\theta^k, \xi_k))^{-1} \cdot (\theta^* - \theta^k)^T \nabla^2 F(t_k, \xi_k)(\theta^* - \theta^k)\|
\]

\[
= \frac{1}{2} \mathbb{E}_{\xi_0, \xi_1, \ldots, \xi_{k-1}}(\|\nabla F(\theta^k, \xi_k))^{-1} \cdot (\theta^* - \theta^k)^T \nabla^2 F(t_k, \xi_k)(\theta^* - \theta^k)\|
\]

\[
= \frac{1}{2} \mathbb{E}\left(\frac{1}{|I|} \sum_{i=1}^{|I|} \|\nabla F(\theta^k, \gamma_i))^{-1} \cdot (\theta^* - \theta^k)^T \nabla^2 F(t_k, \gamma_i)(\theta^* - \theta^k)\|\right)
\]

\[
\leq \frac{1}{2} \frac{1}{|I|} \sum_{i=1}^{|I|} \mathbb{E}(\|\nabla F(\theta^k, \gamma_i))^{-1} \cdot \|\nabla^2 F(t_k, \gamma_i)\| \cdot \|\theta^* - \theta^k\|^2).
\]

(3.19)

In a small neighborhood of \( \theta^* \), our assumptions in the theorem are equivalent to, for any \( i \in \{1, 2, \ldots, |I|\} \),

1. \( \nabla F(\theta^k, \gamma_i) \) is invertible and \( \nabla F(\cdot, \gamma_i) \) is continuous;
2. \( \nabla^2 F(\cdot, \gamma_i) \) is continuous.

Then in a small neighborhood of \( \theta^* \), \( \nabla F(\cdot, \gamma_i) \) is invertible. Since \( \nabla F(\cdot, \gamma_i)(\nabla F(\cdot, \gamma_i))^{-1} = I \), we also have \( (\nabla F(\cdot, \gamma_i))^{-1} \) is continuous in a small neighborhood of \( \theta^* \). Therefore, \( \exists M > 0 \) in a neighborhood of \( \theta^* \) such that \( \| (\nabla F(\theta, \gamma_i))^{-1} \cdot \| \| \nabla^2 F(t_k, \gamma_i) \| \| \leq M \) for \( \| \Theta^* \| \leq r \) where \( r \leq \min(\frac{1}{M}, 1) \). Then we have

\[
\mathbb{E}(\|\theta^* - \theta^1\|) \leq \frac{M}{2} \|\theta^* - \theta^0\|^2 \leq \frac{1}{2} \|\theta^* - \theta^0\|,
\]

(3.20)

which implies that \( \theta^1 \) is also inside this small neighborhood with probability one. Similarly, we have

\[
\mathbb{E}(\|\theta^* - \theta^1\|^2) \leq \frac{M^2}{4} \|\theta^* - \theta^0\|^2 \leq \frac{1}{4} \|\theta^* - \theta^0\|^2,
\]

(3.21)

Since \( \theta^1 \) is inside this small neighborhood with probability one, combining with (3.21), we also have

\[
\mathbb{E}(\|\theta^* - \theta^2\|) \leq \frac{M}{2} \mathbb{E}(\|\theta^* - \theta^1\|) \leq \frac{M}{8} \|\theta^* - \theta^0\|^2 \leq \frac{1}{8} \|\theta^* - \theta^0\|,
\]

(3.22)
which implies that $\theta^2$ is also inside this neighborhood. Then by induction, we can prove that all $\theta^k$ are in this neighborhood.

Then (3.19) becomes

$$\mathbb{E}\|\theta^* - \theta^{k+1}\| \leq \frac{M}{2} \mathbb{E}\|\theta^* - \theta^k\|^2. \tag{3.23}$$

Similar to (3.19), we have

$$\mathbb{E}\|\theta^* - \theta^k\|^2 \leq \frac{M^2}{4} \mathbb{E}\|\theta^* - \theta^{k-1}\|^4, \tag{3.24}$$

which implies

$$\mathbb{E}\|\theta^* - \theta^{k+1}\| \leq \frac{M^3}{23} \mathbb{E}\|\theta^* - \theta^{k-1}\|^4 \leq \cdots \leq \frac{M^{2k+1-1}}{2^{k+1-1}} \mathbb{E}\|\theta^* - \theta^0\|^{2k+1}. \tag{3.25}$$

According to the previous assumption, $\|\theta^* - \theta^0\| \leq r$ where $r \leq \frac{1}{M}$ (in a small neighborhood of $\theta^*$), we have

$$\mathbb{E}\|\theta^* - \theta^{k+1}\| \leq \left(\frac{M\|\theta^* - \theta^0\|}{2^{k+1-1}}\right)^{2k+1-1} \|\theta^* - \theta^0\| \leq \|\theta^* - \theta^0\| \left(\frac{1}{2}\right)^{2k+1-1}$$

$$= C \left(\frac{1}{2}\right)^{2k+1-1}, \tag{3.26}$$

where $C = \|\theta^* - \theta^0\|$. This implies the quadratic convergence.

\section*{4 Numerical Experiments}

In this section, we demonstrate the feasibility of the randomized Newton’s method on several examples and choose $\|\tilde{F}(\theta)\| < 5 \times 10^{-3}$ as the stopping criteria. (We set the stopping criteria to $\|\tilde{F}(\theta)\|$ instead of $\|F(\theta)\|$ because of the computational efficiency.) Sigmoid function is employed as the activation function of neural networks for all the examples. We also compare sigmoid, tanh, and softplus activation functions in example 4.1.1.

\subsection*{4.1 1D Examples}

\subsection*{4.1.1 An Example with the Analytical Solution}

First we show the feasibility of the randomized Newton’s method on (2.7) which the traditional collocation method fails. One-hidden-layer neural networks with different numbers of nodes, $U(x; \theta)$, are used to approximate the solution of (2.7). By using three different uniform grids on $[0, 1]$ with step size 0.1, 0.02, and 0.01 (namely, $n = 11, 51$ and 101 respectively in (3.1)), the randomized Newton’s method shows a good agreement with the real solution. For instance, Figure 2 shows the numerical solution versus the real solution with 10 hidden nodes and 101 uniform sample points. More specifically, we list numerical errors between numerical solutions and the real solution with different numbers of nodes and sample points in Table 1. Here the numerical error $Err(U(x; \theta) - u(x))$ is defined as:

$$Err(U(x; \theta) - u(x)) = \sqrt{\int_0^1 |U(x; \theta) - u(x)|^2 dx}. \tag{4.1}$$
Fig. 2 The numerical solution with 10 nodes and 101 sample points V.S. the real solution to system (2.7)

Table 1 Numerical errors of different numerical methods for Example 4.1.1 with different number of nodes and sample points. Both Adam and SGD are run for 3000 iterations.

| # of sample points | # of nodes | # of variables | # of iterations | Errors (RN) | Errors (Adam) | Errors (SGD) |
|--------------------|------------|----------------|-----------------|-------------|--------------|--------------|
| 11                 | 1          | 4              | 18              | 4.1e−4      | 2.0e−3       | 1.6          |
| 2                  | 7          | 24             | 1.2e−4          | 8.8e−4      | 1.5          |
|                    | 3          | 10             | 1.5e−4          | 1.1e−3      | 1.4          |
| 51                 | 1          | 4              | 28              | 2.3e−4      | 1.2e−3       | 4.1e−2       |
|                    | 5          | 16             | 33              | 4.0e−4      | 2.0e−3       | 5.0e−1       |
|                    | 10         | 31             | 37              | 5.6e−5      | 3.3e−3       | 3.9e−1       |
| 101                | 1          | 4              | 22              | 2.4e−4      | 4.0e−4       | 9.9e−2       |
|                    | 5          | 16             | 64              | 8.0e−5      | 1.3e−2       | 4.9e−1       |
|                    | 10         | 31             | 50              | 6.0e−6      | 8.0e−3       | 5.0e−1       |

From Table 1, we note that a better approximation is achieved by increasing nodes due to the universal approximation theory [9]. Moreover, we compare randomized Newton’s method with both SGD and Adam methods in this example. From Fig. 3, we see that both SGD and Adam methods stabilize after 3000 iterations. Then we show the numerical errors of three different methods with the different number of nodes and sample points in Table 1 which demonstrates that randomized Newton’s method finds a better solution with a smaller error.

4.1.2 An Example with Multiple Solutions

Secondly, we consider the following differential equation with multiple solutions:

\[
\begin{align*}
\frac{d^2 u}{dx^2} &= f(u) \quad \text{on} \ (0, 1), \\
\frac{du}{dx}(0) &= 0 \quad \text{and} \quad u(1) = 0,
\end{align*}
\] (4.2)
Fig. 3 Numerical error v.s. the number of iterations for Adam and SGD optimizers in Example 4.1.1. The neural network has one hidden layer with 10 nodes. The optimization problem (2.5) is evaluated on 101 sample points

where

$$f(u) = -\lambda (1 + u^p), \quad p \in \mathbb{N} \text{ and } \lambda \geq 0. \quad (4.3)$$

When $p = 4$, we have known that there are two solutions if $\lambda < \lambda^* (\lambda^* \approx 1.30107)$, and one solution if $\lambda = \lambda^*$ [18, 21]. The real solutions of (4.2) can be determined by solving the nonlinear equation below for $u(0)$:

$$G(u_0) := \int_0^{u_0} \frac{ds}{\sqrt{F(u_0) - F(s) - \sqrt{2}}} = 0, \quad (4.4)$$

where $F(u) = \int_0^u \lambda (1 + s^p) ds$ and $u_0 = u(0)$. Then we test the randomized Newton’s method for both $\lambda = 1.2$ and $\lambda = \lambda^*$ by employing a one-hidden-layer neural network with two nodes and 101 uniform sample points on $[0, 1]$. More specifically, the numerical solution is written as

$$U(x; \theta) = W_2 \sigma (W_1 x + b_1) + b_2, \quad (4.5)$$

where $W_1, b_1 \in \mathbb{R}^{2 \times 1}$, $W_2 \in \mathbb{R}^{1 \times 2}$, $b_2 \in \mathbb{R}$ and $\theta = \{W_1, b_1, W_2, b_2\}$. For $\lambda = 1.2$, we choose two different initial values

$$\theta_0^1 = \{(1, 1)^T, (1, 1)^T, (1, 1), 1\} \text{ and } \theta_0^2 = \{(5, 0.5)^T, (1, -3)^T, (1, -27), 2\}$$

and obtain two numerical solutions, as shown in Fig. 4a; For $\lambda = \lambda^*$, the two initial values yield the same numerical solution, as shown in Fig. 4b. Moreover, we also test the algorithm on neural networks with different structures as well as different activation functions and show numerical errors in Table 2.
Table 2  Numerical errors for neural networks with different structures for Example 4.1.1 (the first two rows are neural networks with a 1-hidden layer, the third and fourth rows are neural networks with 2-hidden layers, and the last two rows are neural networks with 3-hidden layers. The index on the second column stands for the width of each layer, for example, (3, 2) means that the width of the first hidden layer is 3 and the width of the second hidden layer is 2)

| # of sample points | Width of hidden layers | # of variables | Errors (sigmoid) | Errors (tanh) | Errors (softplus) |
|--------------------|------------------------|----------------|------------------|--------------|------------------|
| 101                | 3                      | 10             | 8.6e−4           | 2.9e−3       | 4.7e−3           |
| 5                  | 16                     | 9.0e−4         | 9.9e−4           | 2.2e−3       |
| (2, 2)             | 13                     | 3.0e−4         | 2.5e−4           | 3.3e−4       |
| (3, 2)             | 17                     | 9.0e−4         | 4.3e−4           | 7.1e−4       |
| (3, 2, 2)          | 23                     | 1.2e−4         | 2.2e−4           | 1.7e−4       |
| (3, 3, 2)          | 29                     | 2.0e−4         | 1.7e−3           | 1.6e−3       |

Fig. 4 Numerical solutions V.S. real solutions of system (4.2)

4.1.3 The 1D Burgers’ Equation

Next we consider the 1D Burgers’ equation with a viscosity term:

\[
\begin{align*}
-\epsilon u_x + (u^2/2)_x &= \sin(x) \cos(x) \text{ on } (0, \pi), \\
u(0) &= 0 \text{ and } u(\pi) = 0,
\end{align*}
\]

where \(\epsilon\) is the viscosity coefficient. We use a one-hidden-layer neural network with ten nodes to approximate the solution of (4.6) and uniform sample points on \([0, \pi]\) with step size 0.01 (\(m=314\)). When \(\epsilon = 1\), the solution is unique and converges to the entropy solution of the 1D Burgers’ equation as \(\epsilon \to 0\) [10]. The analytical solution of system (4.6) when \(\epsilon = 0\) has the following form [6, 7, 19]

\[
u(x) = \begin{cases} 
\sin(x) & 0 \leq x < x_0, \\
-\sin(x) & x_0 < x \leq \pi,
\end{cases}
\]

where \(x_0 \in [0, \pi]\) is the shock location. Specifically when \(x_0 = \pi/2\), it becomes the entropy solution due to the symmetry. Therefore, we use the randomized Newton’s method to solve (4.6) as tracking \(\epsilon\) from 1 to 0. In order to test the randomized Newton’s method, we employ
Fig. 5 Numerical solutions of two tracking methods V.S. real solutions of system (4.6). (Here we only plot solutions with $\epsilon = 1, 0.2,$ and 0. The real solutions in (a) and (b) are defined in (4.7) with $x_0 = \pi/2$ and $x_0 = \pi$, respectively.)

Table 3 Numbers of iterations of the randomized Newton’s method and condition numbers of (4.6) when tracking $\epsilon$ from 1 to 0 for Example 4.1.3

| $\epsilon$ | Homotopy tracking | With a random initial guess |
|------------|------------------|----------------------------|
|            | # of iterations  | Condition numbers | # of iterations  | Condition numbers |
| 1          | 46               | 1.2e4               | 1867            | 1.4e3            |
| 0.8        | 11               | 1.2e4               | 73              | 8.6e3            |
| 0.6        | 5                | 2.3e4               | 1185            | 8.0e3            |
| 0.4        | 21               | 2.3e4               | 269             | 5.7e3            |
| 0.2        | 25               | 1.2e4               | 392             | 1.3e6            |
| 0.1        | 116              | 4.6e4               | 318             | 8.1e5            |
| 0.05       | 43               | 6.4e4               | 3471            | 8.1e4            |
| 0.01       | 285              | 3.1e8               | 371             | 2.0e5            |
| 0          | 2                | 1.0e10              | 57              | 5.0e4            |

two tracking methods: 1) homotopy tracking [19], namely, using the previous solution as the initial guess; 2) the randomized initial guess for each $\epsilon$. We list numerical performance of two tracking methods based on the randomized Newton’s method in Table 3, which shows numbers of iterations and condition numbers for each $\epsilon$. Obviously, homotopy tracking converges much faster and also captures the singularity at $\epsilon = 0$ (see more details in [19]). The numerical solutions are plotted in Figs. 5a and b, which shows that the homotopy tracking obtains the entropy solution while the other method converges to an artificial steady state of $x_0 = \pi$ when $\epsilon \to 0$. The numerical errors at $\epsilon = 0$ are $3.6 \times 10^{-3}$ for homotopy tracking and $4.1 \times 10^{-3}$ for the tracking with a random initial guess. This example demonstrates that the randomized Newton’s method can be coupled with different tracking methods and computes different solutions.
4.2 2D Examples

4.2.1 An Example with the Analytical Solution

We consider the following 2D example on a rectangular domain $\Omega_1 = [0, \pi] \times [0, \pi]$:

$$
\begin{align*}
\Delta u &= -2u & \text{in } \Omega, \\
u(x, y) &= \sin(x + y) & \text{on } \partial \Omega.
\end{align*}
$$

(4.8)

Obviously, the real solution is $u(x, y) = \sin(x + y)$. We use a one-hidden-layer neural network with six nodes to approximate the solution, namely,

$$
U((x, y)^T; \theta) = W_2\sigma(W_1(x, y)^T + b_1) + b_2,
$$

(4.9)

where $W_1 \in \mathbb{R}^{6 \times 2}$, $b_1 \in \mathbb{R}^{6 \times 1}$, $W_2 \in \mathbb{R}^{1 \times 6}$, $b_2 \in \mathbb{R}$ and $\theta = \{W_1, b_1, W_2, b_2\}$. The randomized Newton’s method is used to solve Eq. (4.8) with uniform sample points on $[0, \pi]^2$ with step size 0.01 ($m = 98596$). Figure 6a plots the numerical solution versus the real solution, while Fig. 6b plots the numerical error which shows a good agreement with the real solution by using the randomized Newton’s method.

4.2.2 The 2D Burgers’ Equation

Next we consider the 2D Burgers’ equation with a viscosity term on $\Omega = [0, \pi/\sqrt{2}] \times [0, \pi/\sqrt{2}]$ [6, 7, 19]:

$$
\begin{align*}
\left(\frac{1}{\sqrt{2}} \frac{u_x^2}{2}\right)_x + \left(\frac{1}{\sqrt{2}} \frac{u_y^2}{2}\right)_y - \epsilon \Delta u &= \sin\left(\frac{x+y}{\sqrt{2}}\right) \cos\left(\frac{x+y}{\sqrt{2}}\right) & \text{in } \Omega, \\
u(x, y) &= 0 & \text{on } \partial \Omega.
\end{align*}
$$

(4.10)

Eq. (4.10) recovers the one-dimensional problem in Example 3 if we restrict the solution along the diagonal line. In order to approximate the solution, we use a one-hidden-layer neural network with three nodes with uniform sample points on $[0, \pi/\sqrt{2}]^2$ with step size 0.01 ($m = 49284$). Similar to Example 3, we use two tracking methods with respect to $\epsilon$ from 1 to 0 coupled with the randomized Newton’s method and list their numerical performance in Table 4 which shows that the randomized Newton’s method converges faster with the
Table 4  Numbers of iterations of the randomized Newton’s method and condition numbers of (4.10) when tracking $\epsilon$ from 1 to 0 for Example 4.2.2

| $\epsilon$ | Homotopy tracking |           | Condition numbers |           | With a random initial guess |           |
|------------|--------------------|-----------|-------------------|-----------|----------------------------|-----------|
|            | # of iterations    | Condition numbers |                  |           | # of iterations    | Condition numbers |                  |
| 1          | 75                 | 3.4e2     | 149               | 5.0e4     | 0.8                     | 15             | 5.2e2           | 25             | 1.1e4          | 0.6                     | 8               | 5.2e2           | 119            | 1.6e3          | 0.4                     | 30              | 1.5e4           | 66             | 4.0e3          | 0.2                     | 22              | 5.2e3           | 32             | 3.8e4          | 0.05                    | 182             | 1.5e3           | 337            | 8.4e4          | 0.02                    | 233             | 1.1e5           | 494            | 2.5e4          | 0.01                    | 126             | 2.6e4           | 137            | 1.7e3          | 0                       | 928             | 1.7e6           | 176            | 3.2e4          |

Fig. 7  Numerical solutions of two tracking methods V.S. real solutions of system (4.10). (Here we only plot solutions with $\epsilon = 1, 0.2, 0.05$ and 0.)

homotopy tracking. The numerical solutions of two tracking methods are plotted in Figs. 7a and b which show that the homotopy tracking yields the entropy solution while the random initial guess leads to an artificial steady state. The number of iterations with the homotopy tracking is higher than that with the random initial guess because the homotopy tracking converges to the entropy solution which is singular in this example (with a larger condition number shown in Table 4). However, the method with a random initial guess converges to a trivial steady solution which is nonsingular (with a small condition number shown in Table 4). In order to better compare with the real solution, here we plot solutions along the diagonal line of $\Omega$. The numerical errors are $1.8 \times 10^{-2}$ for the solution with the homotopy tracking and $1.6 \times 10^{-2}$ for the solution with a random initial guess.
Fig. 8 **Left:** numerical solutions vs. real solutions of system (4.11) in the radial direction; **Middle:** the numerical solution for \( n = 2 \); **Right:** the numerical solution on the \( x - y \) plane for \( n = 3 \)

Table 5  Numerical errors of different \( n \) for solving Eq. (4.11) in Example 4.3

| \( n \) | Width of hidden layers | # of variables | Errors  |
|-------|------------------------|----------------|--------|
| 2     | 10                     | 41             | 1.1e−3 |
| 3     | 35                     | 176            | 2.1e−3 |
| 4     | 80                     | 481            | 5.0e−3 |
| 5     | 100                    | 701            | 4.2e−3 |
| 6     | 100                    | 801            | 4.1e−3 |

4.3 A High-dimensional Example

We consider the following Laplace’s equation on the unit \( n \)-ball, namely, \( \Omega = \{ x \in \mathbb{R}^n : \| x \| \leq 1 \} \)

\[
\begin{aligned}
- \Delta u &= \| x \| \quad \text{in } \Omega, \\
u(x) &= 1 \quad \text{on } \partial \Omega.
\end{aligned}
\]  

(4.11)

For the radial symmetric case, the real solution has the following form

\[
u(r; n) = \frac{-r^3 + 3n + 4}{3n + 3},
\]  

(4.12)

where \( 0 \leq r \leq 1 \) and \( n \geq 2 \).

Numerically, we use one-hidden-layer neural networks with different nodes and \( 10^{2n} \) uniform sample points on \( \Omega \) to solve Eq. (4.11) by employing the randomized Newton’s method. Figure 8 shows a good agreement of numerical solutions with real solutions for \( n = 2 \) and \( n = 3 \). For higher dimensional case \( (n > 3) \), we list numerical errors different \( n \) in Table 5, which shows that the randomized Newton’s method can be used for solving high-dimensional differential equations.

4.4 An Application to the Pattern Formation

The pattern formation, as an important problem in physics and biology, involves nonlinear differential equations in various mathematical models. One of the key questions is to compute the nonuniform steady states of nonlinear differential equations, which are the so-called the stationary spatial patterns [17, 24, 30, 52]. However, numerical methods of solving these nonlinear systems, e.g, Newton’s method, are normally sensitive to the initial guesses that are hard to construct for the pattern formation models. In this paper, we will use the neural
network discretization to “learn” these nonuniform patterns. Although the accuracy of neural network discretization is low, it will provide an alternative way to compute the multiple solutions, which is hard for the traditional discretizations such as the finite difference or finite element methods. We use the Gray-Scott model [25, 38] to illustrate the idea:

\[
\begin{align*}
\frac{\partial A}{\partial t} &= D_A \Delta A + SA^2 - (\mu + \rho)A, \\
\frac{\partial S}{\partial t} &= D_S \Delta S - SA^2 + \rho(1 - S),
\end{align*}
\]

(4.13)

where \( A \) is the concentration of an activator and \( S \) is the concentration of a substrate. The growth of \( A \) reacts with \( S \) fed from the activator with a rate \( \rho \), and \( S \) is converted to an inert product at the rate \( \mu \). \( D_A \) and \( D_S \) are the diffusion coefficients of \( A \) and \( S \), respectively. In our computations, we consider the steady state system, namely \( \frac{\partial A}{\partial t} = \frac{\partial S}{\partial t} = 0 \) in (4.13), and choose \( D_A = 2.5 \times 10^{-4} \), \( D_S = 5 \times 10^{-4} \), \( \rho = 0.04 \) and \( \mu = 0.065 \).

First, we consider the 1D case with \([0, 1]\) as the domain and use a one-hidden-layer neural network with ten nodes to discretize \( A \) and \( S \), namely,

\[A(x) = W_2^T \sigma(W_1 x + b_1) + b_2, \quad S(x) = W_4^T \sigma(W_3 x + b_3) + b_4,\]

(4.14)

where \( W_i \in \mathbb{R}^{10} \) \((i = 1, \ldots, 4)\), \( b_1 \in \mathbb{R}^{10} \), \( b_3 \in \mathbb{R}^{10} \), \( b_2 \in \mathbb{R} \), and \( b_4 \in \mathbb{R} \). Based on the following four initial guesses, we obtained different steady patterns by using the randomized Newton’s method and show the results in Fig. 9:

\[
\begin{align*}
\text{IG1: } (A, S) &= \left(\frac{3}{10} \cos(3\pi x) + \frac{1}{2}, -\frac{3}{10} \cos(3\pi x) + \frac{1}{2}\right), \\
\text{IG2: } (A, S) &= \left(-\frac{3}{10} \cos(3\pi x) + \frac{1}{2}, \frac{3}{10} \cos(3\pi x) + \frac{1}{2}\right), \\
\text{IG3: } (A, S) &= \left(\frac{3}{10} \cos(\pi x) + \frac{1}{2}, -\frac{3}{10} \cos(\pi x) + \frac{1}{2}\right), \\
\text{IG4: } (A, S) &= \left(-\frac{3}{10} \cos(\pi x) + \frac{1}{2}, \frac{3}{10} \cos(\pi x) + \frac{1}{2}\right).
\end{align*}
\]

(4.15)

We use 101 uniform sample points on \([0, 1]\) for 1D simulations.

Secondly, we consider the 2D case with the domain as \([0, 1]^2\) and also use a one-hidden layer neural network with ten nodes as the discretization, namely,

\[A(x, y) = W_2^T \sigma(W_1 (x, y)^T + b_1) + b_2 \quad \text{and} \quad S(x, y) = W_4^T \sigma(W_3 (x, y)^T + b_3) + b_4\]

(4.16)

where \( W_1, W_3 \in \mathbb{R}^{10 \times 2} \), \( W_2, W_4 \in \mathbb{R}^{10} \), \( b_1, b_3 \in \mathbb{R}^{10} \), \( b_2, b_4 \in \mathbb{R} \). In the 2D case, we run the randomized Newton’s method many times with the same initial guess in order to “learn” the multiple steady patterns. For instance, the initial guess shown in Fig. 10 (left) yields two stable patterns shown in Fig. 10 (right); Fig. 11 shows four steady patterns can be “learned” from one initial guess. Thus the randomized Newton’s method can be used to compute the multiple solutions of nonlinear differential equations. We use 10201 uniform sample points on \([0, 1] \times [0, 1]\) for 2D simulations.

5 Conclusion

In this paper, we develop a randomized Newton’s method for solving differential equations based on the fully connected neural network discretization. This proposed method is designed
Fig. 9 Steady patterns of the 1D Gray-Scott model by using different initial guesses shown in Eq. (4.15) with residual specifically to solve an overdetermined nonlinear system, since the number of sample points in such a system is much larger than the number of variables. For each iteration, we randomly choose equations from the nonlinear system and apply the classical Newton’s method repeatedly, and we prove theoretically that the randomized Newton’s method has a local quadratic convergence. Using several examples, we also demonstrate, numerically, that the randomized Newton’s method for solving both linear and nonlinear equations is indeed efficient and feasible. Moreover, the method developed here can be used to solve high-dimensional differential equations that are otherwise hard to solve by traditional numerical methods. Another advantage of this method is that it allows for computing the multiple solutions of nonlinear differential equations, such as pattern formation problems. In future work, we will apply the other types of neural networks to discretize differential equations (e.g., convolutional neural networks) and aim to reduce the redundancy of neural network discretization, in order to improve the convergence of the randomized Newton’s method. We will also apply the randomized Newton’s method to other overdetermined nonlinear systems such as function approximations and data-driven problems. Another future direction is to speed up the computation of Newton’s method by employing the Jacobian free Newton-GMRES solver to
Fig. 10  **Left:** The initial guess of $A(x, y)$; **Right:** steady patterns of $A(x, y)$ for the 2D Gray-Scott model

steady state solution 1 with residual $4.9 \times 10^{-3}$

IG1

steady state solution 2 with residual $4.9 \times 10^{-3}$

Fig. 11  **Left:** The initial guess of $A(x, y)$; **Right:** steady patterns of $A(x, y)$ for the 2D Gray-Scott model

steady state solution 1 with residual $4.9 \times 10^{-3}$

steady state solution 2 with residual $4.9 \times 10^{-3}$

steady state solution 3 with residual $4.7 \times 10^{-3}$

steady state solution 4 with residual $4.9 \times 10^{-3}$

IG2
handle deep neural networks with millions of parameters. More specifically, the Jacobian free Newton-GMRES solver can speed up the computation of the matrix inverse and reduce the memory of storing the Jacobian matrix for large neural networks.

**Funding** The authors have not disclosed any funding.

**Data availability** Data sharing not applicable to this article as no datasets were generated or analysed during the current study.

**Declarations**

**Conflict of interest** This research is supported by NSF via DMS-1818769. There is no conflict of interest.

**References**

1. Ames, W.F.: Nonlinear partial differential equations in engineering, vol. 18. Academic press (1965)
2. Baranyi, J., Roberts, T.A., McClure, P.: A non-autonomous differential equation to model bacterial growth. Food Microbiol. 10(1), 43–59 (1993)
3. Bramble, J.H., Pasciak, J.E., Wang, J., Xu, J.: Convergence estimates for multigrid algorithms without regularity assumptions. Math. Comput. 57(195), 23–45 (1991)
4. Büngartz, Hans-Joachim., Griebel, Michael: Sparse grids. Acta Numer 13, 147–269 (2004)
5. Chen, J., Lopes, M.: Estimating the error of randomized newton methods: A bootstrap approach. In: International Conference on Machine Learning, pp. 1649–1659. PMLR (2020)
6. Chou, C., Shu, C.: High order residual distribution conservative finite difference weno schemes for steady state problems on non-smooth meshes. J. Comput. Phys. 214(2), 698–724 (2006)
7. Chou, C., Shu, C.: High order residual distribution conservative finite difference weno schemes for convection-diffusion steady state problems on non-smooth meshes. J. Comput. Phys. 224(2), 992–1020 (2007)
8. Courant, R., Hilbert, D.: Methods of Mathematical Physics: Partial Differential Equations. John Wiley & Sons (2008)
9. Cybenko, G.: Approximation by superpositions of a sigmoidal function. Math. Control Signals Systems 2(4), 303–314 (1989)
10. De Lellis, C., Otto, F., Westdickenberg, M.: Minimal entropy conditions for burgers equation. Q. Appl. Math. 62(4), 687–700 (2004)
11. Della-Dora, J.: Numerical linear algorithms and group theory. Linear Algebra Appl. 10(3), 267–283 (1975)
12. Han, W.E.J., Jentzen, A.: Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations. Communications in Mathematics and Statistics 5(4), 349–380 (2017)
13. Evans, L.C.: Partial Differential Equations. Graduate studies in mathematics. American Mathematical Society (2010)
14. Farrell, P.E., Birkisson, A., Funke, S.W.: Deflation techniques for finding distinct solutions of nonlinear partial differential equations. SIAM J. Sci. Comput. 37(4), A2026–A2045 (2015)
15. Fedosov, D.A., Lei, H., Caswell, B., Suresh, S., Karniadakis, G.E.: Multiscale modeling of red blood cell mechanics and blood flow in malaria. PLoS Comput. Biol. 7(12), e1002270 (2011)
16. Forsyth, D.A., Ponce, J.: Computer vision: a modern approach. Prentice Hall Professional Technical Reference (2002)
17. Gierer, A., Meinhardt, H.: A theory of biological pattern formation. Kybernetik 12(1), 30–39 (1972)
18. Hao, W., Hauenstein, J.D., Hu, B., Sommese, A.J.: A bootstrapping approach for computing multiple solutions of differential equations. J. Comput. Appl. Math. 258, 181–190 (2014)
19. Hao, W., Hauenstein, J.D., Shu, C., Sommese, A.J., Xu, Z., Zhang, Y.: A homotopy method based on weno schemes for solving steady state problems of hyperbolic conservation laws. J. Comput. Phys. 250, 332–346 (2013)
20. He, J., Li, L., Xu, J., Zheng, C.: Relu deep neural networks and linear finite elements. arXiv preprint arXiv:1807.03973 (2018)
21. Hu, B.: Blow-up theories for semilinear parabolic equations. Springer (2011)
22. Karniadakis, G., Sherwin, S.: Spectral/hp element methods for computational fluid dynamics. Oxford University Press (2013)
23. Kloeden, P.E., Platen, E.: Numerical solution of stochastic differential equations, vol. 23. Springer Science & Business Media (2013)
24. Kondo, S., Miura, T.: Reaction-diffusion model as a framework for understanding biological pattern formation. Science 329(5999), 1616–1620 (2010)
25. Lee, K.J., McCormick, W.D., Ouyang, Q., Swinney, H.L.: Pattern formation by interacting chemical fronts. Science 261(5118), 192–194 (1993)
26. Lei, H., Karniadakis, G.E.: Quantifying the rheological and hemodynamic characteristics of sickle cell anemia. Biophys. J. 102(2), 185–194 (2012)
27. Leykin, A.: Numerical algebraic geometry. Journal of Software for Algebra and Geometry 3(1), 5–10 (2011)
28. Li, Q., Tai, C., W.E.: Stochastic modified equations and adaptive stochastic gradient algorithms. In: Proceedings of the 34th International Conference on Machine Learning-Volume 70, pages 2101–2110. JMLR.org (2017)
29. Liu, Y., Cheng, Y., Chen, S., Zhang, Y.: Krylov implicit integration factor discontinuous galerkin methods on sparse grids for high dimensional reaction-diffusion equations. J. Comput. Phys. 388, 90–102 (2019)
30. Maini, P.K., Woolley, T.E., Baker, R.E., Gaffney, E.A., Lee, S.S.: Turing’s model for biological pattern formation and the robustness problem. Interface focus 2(4), 487–496 (2012)
31. Manning, C.D., Schütze, H.: Foundations of statistical natural language processing. MIT press (1999)
32. Marion, M., Xu, J.: Error estimates on a new nonlinear galerkin method based on two-grid finite elements. SIAM J. Numer. Anal. 32(4), 1170–1184 (1995)
33. Medler, D.A., Dawson, M.R.W.: Using redundancy to improve the performance of artificial neural networks. In: Proceedings of the Biennial Conference-Canadian Society for Computational Studies of Intelligence, pages 131–138. CANADIAN INFORMATION PROCESSING SOCIETY (1994)
34. Nobile, F., Tempone, R., Webster, C.G.: A sparse grid stochastic collocation method for partial differential equations with random input data. SIAM J. Numer. Anal. 46(5), 2309–2345 (2008)
35. Nocedal, J., Yuan, Y.: Combining trust region and line search techniques. In: Advances in nonlinear programming, pages 153–175. Springer (1998)
36. Pearson, J.E.: Complex patterns in a simple system. Science 261(5118), 189–192 (1993)
37. Pilanci, M., Wainwright, M.: Newton sketch: A near linear-time optimization algorithm with linear-quadratic convergence. SIAM J. Optim. 27(1), 205–245 (2017)
38. Piscopo, M.L., Spannowsky, M., Waite, P.: Solving differential equations with neural networks: Applications to the calculation of cosmological phase transitions. Phys. Rev. D 100, 016002 (2019)
39. Qin, T., Wu, K., Xiu, D.: Data driven governing equations approximation using deep neural networks. J. Comput. Phys. 395, 620–635 (2019)
40. Russell, R.D., Shampine, L.F.: A collocation method for boundary value problems. Numer. Math. 19(1), 1–28 (1972)
41. Schumack, M.R., Schultz, W.W., Boyd, J.P.: Spectral method solution of the stokes equations on non-staggered grids. J. Comput. Phys. 94(1), 30–58 (1991)
42. Shen, J., Tang, T., Wang, L.: Spectral methods: algorithms, analysis and applications, vol. 41. Springer Science & Business Media (2011)
43. Shu, C.: High-order finite difference and finite volume weno schemes and discontinuous galerkin methods for cfd. International Journal of Computational Fluid Dynamics 17(2), 107–118 (2003)
44. Siegel, Jonathan W., Xu, Jinchao: High-order approximation rates for neural networks with ReLU activation functions. arXiv preprint arXiv:2012.07205 (2020)
45. Sirignano, J., Spiliopoulos, K.: Dgm: A deep learning algorithm for solving partial differential equations. J. Comput. Phys. 375, 1393–1364 (2018)
46. Smith, Barry, Bjorstad, Petter, Gropp, William: Domain decomposition: parallel multilevel methods for elliptic partial differential equations. Cambridge University Press (2004)
47. Sommese, A.J., Verschelde, J., Wampler, C.W.: Introduction to numerical algebraic geometry. In: Solving polynomial equations, pages 301–337. Springer (2005)
48. Toselli, A., Widlund, O.: Domain decomposition methods-algorithms and theory, vol. 34. Springer Science & Business Media (2006)
52. Turing, A.M.: The chemical basis of morphogenesis. Bull. Math. Biol. 52(1–2), 153–197 (1990)
53. Wampler, C.W., Sommese, A.J.: Numerical algebraic geometry and algebraic kinematics. Acta Numer 20, 469–567 (2011)
54. Wang, Y., Hao, W., Lin, G.: Two-level spectral methods for nonlinear elliptic equations with multiple solutions. SIAM J. Sci. Comput. 40(4), B1180–B1205 (2018)
55. Wang, Y., Lin, G.: Efficient deep learning techniques for multiphase flow simulation in heterogeneous porous media. Journal of Computational Physics, page 108968 (2019)
56. Weickert, J., Schnörr, C.: A theoretical framework for convex regularizers in pde-based computation of image motion. Int. J. Comput. Vision 45(3), 245–264 (2001)
57. Wu, K., Xiu, D.: Data-driven deep learning of partial differential equations in modal space. arXiv preprint arXiv:1910.06948 (2019)
58. Xing, Y., Shu, C.: High order finite difference weno schemes with the exact conservation property for the shallow water equations. J. Comput. Phys. 208(1), 206–227 (2005)
59. Xiu, D.: Numerical methods for stochastic computations: a spectral method approach. Princeton University Press (2010)
60. Xu, J., Zikatanov, L.: A monotone finite element scheme for convection-diffusion equations. Mathematics of Computation of the American Mathematical Society 68(228), 1429–1446 (1999)
61. Xu, J., Zikatanov, L.: On an energy minimizing basis for algebraic multigrid methods. Comput. Vis. Sci. 7(3–4), 121–127 (2004)
62. Xu, Jinchao: The finite neuron method and convergence analysis. arXiv preprint arXiv:2010.01458 (2020)

Publisher’s Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.