Journal of Nonlinear Modeling and Analysis
Volume 8, Number 2, April 2026, 551–569
https://global-sci.com/jnma
DOI:10.12150/jnma.2026.551

A Robust Fixed Point Method to Solve System of Nonlinear Equations

M.R. Amattouch[1,†]

Received 26 February 2025; Accepted 21 October 2025

Abstract Solving nonlinear partial differential equations requires addressing systems of nonlinear algebraic equations. In this article, we propose a new fixed-point method for solving these systems. We assume that the method is both fast and globally convergent. Additionally, this method can be accelerated using Aitken or Anderson acceleration techniques. Several numerical test cases are presented to illustrate the efficiency of the proposed method.

Keywords Brouwer fixed point theorem, system of equations, global convergence

MSC(2010) 65H10 , 65J15, 65Bxx , 35K55 , 35J66.

1. Introduction

Nonlinear partial differential equations are widely used to simulate various industrial and biological phenomena. Discretizing these equations results in a system of nonlinear equations, as seen in [1–3]. Thus, we focus on solving the system of equations

$$F(x) = 0, \tag{1.1}$$

where $F : \mathbb{R}^n \to \mathbb{R}^n$ is a nonlinear, continuously differentiable function, particularly when n is large. We consider cases where $n \geq 3$.

The most popular method for solving (1.1) is the Newton method [4, 5]. This method, known for its physical interpretation and strong convergence properties, poses a major challenge: it requires calculating the Jacobian matrix of F: F', and evaluating its inverse at each iteration, which is computationally expensive, especially for large dimensions n. Another limitation is that convergence is not guaranteed in the presence of singularities, and the method converges slowly for flat functions.

Alternative methods to Newton's method include quasi-Newton and inexact Newton methods, which approximate F'^{-1} to reduce computational costs. Various quasi-Newton methods are discussed in [6–10]. However, these methods still entail high computational costs for many problems. A solution is to accelerate the method's convergence, such as with the BFGS method [11, 12]. Another accelerator is the Krylov method [13, 14]. Yet, these methods also face difficulties with large-scale problems, particularly for flat functions and problems with singularities.

†the corresponding author.
Email address:amattouch36@gmail.com.
[1]Department of Mathematics and Informatics, University AbdelMalik Essaadi, High national school of engineering of Alhoceima, BP 146, Alhoceima, 28806, Morocco

As an alternative to Newton-based methods, accelerated fixed-point methods are often employed. Two common accelerators are Anderson acceleration ([15,16]) and line search methods ([17,18]). These methods offer the advantage of global convergence and are more practical for large-scale problems arising from partial differential equations.

In this article, we introduce a new fixed-point method and provide proofs of both its local and global convergence, as well as an explanation of its hyper-convergence properties. Specifically, we propose a convergence theorem under certain assumptions, modify the general equations to satisfy these assumptions, and achieve fast global convergence through these modifications. This method is highly efficient and applicable to linear systems of equations, infinite-dimensional problems, and optimization tasks. Its computational complexity is favorable, as it only requires basic operations such as matrix multiplication and addition, without necessitating the solution of minimization problems or matrix inversion. Through various test cases, we demonstrate the method's efficiency, particularly for problems involving singularities and large-scale systems. Additionally, the method's performance can be further enhanced using acceleration techniques like Anderson or Aitken methods.

2. The proposed method

Notice that the function F of problem 1.1 can be written as:

$$F = \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_n \end{pmatrix}$$

where $f_1, f_2, ..., f_n$ are functions defined on $\mathbb{R}$.

The Jacobian of the function F is defined by the matrix

$$F'(x) = (f_{i,j}(x))_{1\leq i,j\leq n},$$

where $f_{i,j}(x) = \dfrac{\partial f_i}{\partial x_j}$.

We also define the function sign defined on $\mathbb{R}$ as

$$sign(x) = \begin{cases} 1 & \text{if } x > 1. \\ -1 & \text{if } x < 1. \\ 0 & \text{if } x = 0. \end{cases}$$

Consider $x_0 \in \mathbb{R}^n$ as an initial guess. The sequence of our method is constructed as:

$$x_{k+1} = x_k - \frac{1}{d(x_k)} H(x_k)F(x_k), \tag{2.1}$$

where

$$d(x) = \max_{k\in[\![1;n]\!]} \sum_{i=1}^{n} |\frac{\partial f_i(x)}{\partial x_k}| + MN|\frac{\partial f_k}{\partial x_k}|$$

and

$$H(x)=\begin{pmatrix} Msign(\frac{\partial f_1(x)}{\partial x_1}) & sign(\frac{\partial f_1(x)}{\partial x_2}) & sign(\frac{\partial f_1(x)}{\partial x_3}) & \dots & sign(\frac{\partial f_1(x)}{\partial x_n}) \\ sign(\frac{\partial f_2(x)}{\partial x_1}) & Msign(\frac{\partial f_2(x)}{\partial x_2}) & sign(\frac{\partial f_2(x)}{\partial x_3}) & & \vdots \\ sign(\frac{\partial f_3(x)}{\partial x_1}) & \ddots & \ddots & \ddots & \vdots \\ \vdots & & & & \\ sign(\frac{\partial f_n(x)}{\partial x_1}) & \dots & sign(\frac{\partial f_n(x)}{\partial x_{n-2}}) & sign(\frac{\partial f_n(x)}{\partial x_{n-1}}) & Msign(\frac{\partial f_n(x)}{\partial x_n}) \end{pmatrix}$$

where N and M are non-null positive numbers.

So we have the algorithm:

Algorithm 1: General fixed point method

1. **Require** x_0, ε: precision.
2. **Ensure** x^* solution of $F(x)=0$.
3. **State** $d(x)$ and $H(x)$.
4. **State** $dd=0$ and $x=x_0$.
5. **While** $\|F(x)\|\geq\varepsilon$ and $dd<100$.
6. **State** $x=x-\frac{1}{d(x)}H(x)F(x)$.
7. **State** dd=dd+1.
8. **EndWhile**
9. **If** $dd<100$ **State** $x^*=x$.
10. **EndIf**

The matrix $H(x)$ is selected to be diagonally dominant in order to ensure energy super-contraction, thereby guaranteeing the super-linear convergence of our fixed-point method to the solution.

For regularity reasons, we can change the *sign* function to a regular function of order 2 as follows:

$$sign(x)=\begin{cases} 1 \text{ if } x>1+\mu. \\ -1 \text{ if } x<1-\mu. \\ \text{else sign is a polynomial of degree 2 that is null in 0.} \end{cases}$$

Where μ is a very small positive number.

The following theorem provides the local convergence of Algorithm 2.

Theorem 2.1. *Let x^* be a zero of F, and suppose that*

1. *F is C^1-differentiable in a neighborhood of x^*.*
2. $\forall k\in[\![1;n]\!]\ |\frac{\partial f_k(x^*)}{\partial x^k}|>\sum_{i=1,i\neq k}^{n}|\frac{\partial f_k(x^*)}{\partial x^i}|$.
3. $N\geq 1$.

There exists M_0 such that the sequence generated by Algorithm 2 converges to x^* for some initial guess x_0 in a neighborhood of x^*.

Proof. The strategy is to apply Brouwer's fixed-point theorem ([19, 20]) to the map $\Phi : \mathbb{R}^n \to \mathbb{R}^n$ defined by

$$\Phi(x) = x - \frac{1}{d(x)} H(x)F(x).$$

The Jacobian of this map at x^* is

$$\Phi'(x^*) = I_n - \frac{1}{d(x^*)} H(x^*)F'(x^*).$$

Consider λ as an eigenvalue of the matrix $\Phi'(x^*)$ and x as an associated eigenvector. Let k be such that $|x(k)| = \max\limits_{i\in[\![1;n]\!]} |x(i)| \neq 0$. We then have:

$$\Phi'(x^*)x = x - \frac{1}{d(x^*)} H(x^*)F'(x^*)x = \lambda x.$$

This implies the following for the kk-th component:

$$\lambda x(k) = \left(1 - \frac{M \mid \frac{\partial f_k(x^*)}{\partial x_k} \mid - \sum\limits_{i=1, i\neq k}^{n} \mid \frac{\partial f_i(x^*)}{\partial x_k} \mid}{d(x^*)} \right) x(k)$$
$$- \frac{\sum\limits_{j=1, j\neq k}^{n} \left(\sum\limits_{i=1, i\neq k}^{n} sign(\frac{\partial f_i(x^*)}{\partial x_k}) \frac{\partial f_i(x^*)}{\partial x_j} - M sign(\frac{\partial f_k(x^*)}{\partial x_k}) \frac{\partial f_k(x^*)}{\partial x_j} \right) x(j)}{d(x^*)}.$$

Taking absolute values, we obtain:

$$|\lambda| \mid x(k) \mid \leq \left| \left(1 - \frac{M \mid \frac{\partial f_k(x^*)}{\partial x_k} \mid + \sum\limits_{i=1, i\neq k}^{n} \mid \frac{\partial f_i(x^*)}{\partial x_k} \mid}{d(x^*)} \right) \right| \mid x(k) \mid$$
$$+ \frac{\sum\limits_{j=1, j\neq k}^{n} \left(\sum\limits_{i=1, i\neq k}^{n} \mid \frac{\partial f_i(x^*)}{\partial x_j} \mid + M \mid \frac{\partial f_k(x^*)}{\partial x_j} \mid \right) \mid x(j) \mid}{d(x^*)}.$$

Since $M \mid \frac{\partial f_k(x^*)}{\partial x_k} \mid + \sum\limits_{i=1, i\neq k}^{n} \mid \frac{\partial f_i(x^*)}{\partial x_k} \mid \leq d(x^*)$ and $\left| \frac{x(j)}{x(k)} \right| \leq 1$, we get:

$$|\lambda| \leq \left(1 - \frac{M \mid \frac{\partial f_k(x^*)}{\partial x_k} \mid + \sum\limits_{i=1, i\neq k}^{n} \mid \frac{\partial f_i(x^*)}{\partial x_k} \mid}{d(x^*)} \right) + \frac{\sum\limits_{j=1, j\neq k}^{n} \left(\sum\limits_{i=1, i\neq k}^{n} \mid \frac{\partial f_i(x^*)}{\partial x_j} \mid + M \mid \frac{\partial f_k(x^*)}{\partial x_j} \mid \right)}{d(x^*)}$$

$$\leq 1 - \frac{M\left(\mid \frac{\partial f_k(x^*)}{\partial x_k} \mid - \sum_{i=1,i\neq k}^{n} \mid \frac{\partial f_k(x^*)}{\partial x_j}\right) + \sum\limits_{i=1,i\neq k}^{n} \mid \frac{\partial f_i(x^*)}{\partial x_k} \mid - \sum\limits_{j=1,j\neq k}^{n} \sum\limits_{i=1,i\neq k}^{n} \mid \frac{\partial f_i(x^*)}{\partial x_j} \mid}{d(x^*)}.$$

Choosing $M > M_0 = \dfrac{-\sum\limits_{i=1,i\neq k}^{n} \mid \frac{\partial f_i(x^*)}{\partial x_k} \mid + \sum\limits_{j=1,j\neq k}^{n} \sum\limits_{i=1,i\neq k}^{n} \mid \frac{\partial f_i(x^*)}{\partial x_j} \mid}{\mid \frac{\partial f_k(x^*)}{\partial x_k} \mid - \sum\limits_{i=1,i\neq k}^{n} \mid \frac{\partial f_k(x^*)}{\partial x_j} \mid}$, the magnitude of the eigenvalue $\mid \lambda \mid$ becomes less than 1, guaranteeing that $\Phi'(x^*)$ is a contraction. Thus, by Brouwer's fixed-point theorem, the sequence generated by Algorithm 2 converges to x^*

□

Next, we present assumptions regarding the global convergence of Algorithm 2. The application Φ is considered in the proof of Theorem 2.1.

Theorem 2.2. *Let $F : U \to U$ where U is a compact set of $\mathbb{R}^n$ and suppose that*

1. *$\Phi(U) \subset U$.*
2. *F is C^1-differentiable On U.*
3. *F has at least one zero x^*.*
4. *$\forall x \in U, \forall k \in [\![1; n]\!];\ |\frac{\partial f_k(x)}{\partial x^k}| > \sum\limits_{i=1,i\neq k}^{n} |\frac{\partial f_k(x)}{\partial x^i}|$.*

There exist M_0 and N_0 such that the sequence generated by Algorithm 2 converges to a zero of F.

Proof. Since F is C^1-differentiable on U we define:

$$M_0 = \max_{x\in U} \frac{\sum\limits_{i=1,i\neq k}^{n} \mid \frac{\partial f_i(x)}{\partial x_k} \mid + \sum\limits_{j=1,j\neq k}^{n} \sum\limits_{i=1,i\neq k}^{n} \mid \frac{\partial f_i(x)}{\partial x_j} \mid}{\mid \frac{\partial f_k(x)}{\partial x_k} \mid - \sum\limits_{i=1,i\neq k}^{n} \mid \frac{\partial f_k(x)}{\partial x_j} \mid}$$

and

$$N_0 = \frac{4 \max\limits_{x\in U} \max\limits_{i\in[\![1;n]\!]} \mid \frac{\partial f_i(x)}{\partial x_i} \mid}{\min\limits_{x\in U} \max\limits_{i\in[\![1;n]\!]} \mid \frac{\partial f_i(x)}{\partial x_k} \mid}.$$

Considering the application: $\Lambda : U \to \mathbb{R}^n$ defined by

$$\forall x \in U;\ \Lambda(x) = H(x)F(x),$$

we have

$$\forall x \in U;\ \Lambda'(x) = H(x)F'(x).$$

Let λ be an eigenvalue of the matrix $\Lambda'(x)$ and v be one of its eigenvector. Let k, such that $|v(k)| = \max\limits_{i\in[\![1;n]\!]} |v(i)| \neq 0$. We have:

$$\Lambda'(x)v = H(x)F'(x)v = \lambda v.$$

From this identity, we obtain the following:

$$\lambda v(k) = \left(M \mid \frac{\partial f_k(x)}{\partial x_k} \mid + \sum_{i=1, i\neq k}^{n} \mid \frac{\partial f_i(x)}{\partial x_k} \mid \right) v(k) + \sum_{j=1, j\neq k}^{n} \left(\sum_{i=1, i\neq k}^{n} sign(\frac{\partial f_i(x)}{\partial x_k}) \frac{\partial f_i(x)}{\partial x_j} + M sign(\frac{\partial f_k(x)}{\partial x_k}) \frac{\partial f_k(x)}{\partial x_j} \right) v(j).$$

$$|\lambda| \mid v(k) \mid \leq \left| \left(M \mid \frac{\partial f_k(x)}{\partial x_k} + \mid \sum_{i=1, i\neq k}^{n} \mid \frac{\partial f_i(x)}{\partial x_k} \mid \right) \right| \mid v(k) \mid + \sum_{j=1, j\neq k}^{n} \left(\sum_{i=1, i\neq k}^{n} \mid \frac{\partial f_i(x)}{\partial x_j} \mid + M \mid \frac{\partial f_k(x)}{\partial x_j} \mid \right) \mid v(j) \mid .$$

Since

$$M \left(\mid \frac{\partial f_k(x)}{\partial x_k} \mid - \sum_{i=1, i\neq k}^{n} \mid \frac{\partial f_k(x)}{\partial x_j} \mid \right) > \sum_{i=1, i\neq k}^{n} \mid \frac{\partial f_i(x)}{\partial x_k} \mid - \sum_{j=1, j\neq k}^{n} \sum_{i=1, i\neq k}^{n} \mid \frac{\partial f_i(x)}{\partial x_j} \mid$$

and

$$\mid \frac{v(j)}{v(k)} \mid < 1.$$

we have,

$$\mid \lambda \mid < 2M \left(\mid \frac{\partial f_k(x)}{\partial x_k} \mid + \sum_{i=1, i\neq k}^{n} \mid \frac{\partial f_k(x)}{\partial x_j} \mid . \right) = \delta(x)$$

thus for all $X \in \mathbb{R}^n$ and $Y \in \mathbb{R}^n$, we have:

$$< \Lambda'(x)X, Y > < \delta(x)\|X\|\|Y\|. \tag{2.2}$$

On the other hand, it is evident that the matrix is positive definite because it is diagonally dominant (since $M > \dfrac{\sum_{i=1, i\neq k}^{n} \mid \frac{\partial f_i(x)}{\partial x_k} \mid - \sum_{j=1, j\neq k}^{n} \sum_{i=1, i\neq k}^{n} \mid \frac{\partial f_i(x)}{\partial x_j} \mid}{\mid \frac{\partial f_k(x)}{\partial x_k} \mid - \sum_{i=1, i\neq k}^{n} \mid \frac{\partial f_k(x)}{\partial x_j} \mid}$).

Thus,

$$\forall X \in \mathbb{R}^n, \ < \Lambda'(x)X, X > \geq 0. \tag{2.3}$$

Now, let $x_k \neq x^*$, x^* and consider the application $\chi : \mathbb{R}^n \to \mathbb{R}$ defined by:

$$\forall y \in \mathbb{R}^n; \ \ \chi(y) = < y, \frac{\Lambda(x_k)}{\|\Lambda(x_k)\|} > .$$

By the finite-increment formula, there exists $\theta \in]0, 1[$ such that

$$\chi(\Lambda(x_k)) - \chi(\Lambda(x^*)) = Diff(\chi \circ \Lambda)(x_k + \theta x^*).(x_k - x^*).$$

Thus

$$\|\Lambda(x_k)\|^2 =< \Lambda'(x_k+\theta x^*)(x_k-x^*), \Lambda(x_k) > .$$

Notice $\Lambda(x_k) = -d(x_k)(x_{k+1}-x_k)$, so we have:

$$\begin{aligned} d(x_k)\|x_{k+1}-x_k\|^2 &= < \Lambda'(x_k+\theta x^*)(x_k-x^*),\ x_{k+1}-x_k > \\ &= < \Lambda'(x_k+\theta x^*)(x_{k+1}-x^*-(x_{k+1}-x_k)), x_{k+1}-x_k > \\ &= < \Lambda'(x_k+\theta x^*)(x_{k+1}-x^*), x_{k+1}-x_k > \\ &\quad - < \Lambda'(x_k+\theta x^*)(x_{k+1}-x_k), x_{k+1}-x_k > . \end{aligned} \tag{2.4}$$

By Inequalities 2.2 and 2.3 we have

$$< \Lambda'(x_k+\theta x^*)(x_{k+1}-x_k), x_{k+1}-x_k >\geq 0$$

and

$$< \Lambda'(x_k+\theta x^*)(x_k-x^*), x_{k+1}-x^* >\leq \rho(x_k+\theta x^*)\|x_{k+1}-x^*\|\|x_{k+1}-x_k\|.$$

Substituting these into 2.4, we obtain:

$$d(x_k)\|x_{k+1}-x_k\|^2 \leq \rho(x_k+\theta x^*)\|x_{k+1}-x^*\|\|x_{k+1}-x_k\|.$$

Thus, we have:

$$d(x_k)\|x_{k+1}-x_k\| \leq \rho(x_k+\theta x^*)\|x_{k+1}-x^*\|.$$

Since

$$\|x_{k+1}-x_k\| = \|x_{k+1}-x^*-(x_k-x^*)\| \geq \|x_{k+1}-x^*\| - \|x_k-x^*\|,$$

we obtain

$$\left(\frac{d(x_k)}{\rho(x_k+\theta x^*)}-1\right)\|x_{k+1}-x_k\| \leq \|x_{k+1}-x^*\| - \|x_k-x^*\|.$$

Summing both sides:

$$\sum_{k=0}^{n}\left(\frac{d(x_k)}{\rho(x_k+\theta x^*)}-1\right)\|x_{k+1}-x_k\| \leq \|x_{n+1}-x^*\| - \|x_0-x^*\|.$$

Since $(\|x_{n+1}-x^*\|)_n$ is bounded ($\Phi(U)\subset U$), the series

$$\sum_{k=0}^{\infty}\left(\frac{d(x_k)}{\rho(x_k+\theta x^*)}-1\right)\|x_{k+1}-x_k\|$$

converges. Therefore, $\left(\frac{d(x_k)}{\rho(x_k+\theta x^*)}-1\right)\|x_{k+1}-x_k\|$ converges to 0.

Now, since

$$\left(\frac{d(x_k)}{\rho(x_k+\theta x^*)}-1\right) > \frac{N_0 \min\limits_{x\in U}\max\limits_{i\in[\![1;n]\!]} |\ \dfrac{\partial f_i(x)}{\partial x_i}\ |}{4\max\limits_{x\in U}\max\limits_{i\in[\![1;n]\!]} |\ \dfrac{\partial f_i(x)}{\partial x_k}\ |} > 0,$$

it follows that $\|x_{k+1}-x_k\|$ converges to zero. Therefore, the sequence (x_n) converges to a vector, which must necessarily be zero of F.

□

Remark 2.1. Assumption 2 of Theorem 2.1 may seem like a non-trivial condition, but it is satisfied by almost all nonlinear equations arising from partial differential equations. To address this, we can modify Algorithm 2.1 in such a way that Assumption 2 of Theorem 2.1 holds. An example of such a modification is given next in Algorithm 2.

Remark 2.2. Using the proof of Theorems 2.1 and 2.2, we can choose M such that $\| \Phi'(x^*) \|$ is small enough to ensure the rapid convergence of our algorithm.

A simplified version of Algorithm 2 is to change the algorithm as follows: Consider $x_0 \in \mathbb{R}^n$ as an initial guess. We construct the sequence of our method as:

$$x_{k+1} = x_k - \frac{1}{d(x_k)} H(x_k) F(x_k), \tag{2.5}$$

where

$$d(x) = 1 + MN|\frac{\partial f_k}{\partial x_k}|$$

and

$$H(x) = \begin{pmatrix} Msign(\frac{\partial f_1(x)}{\partial x_1}) & 0 & 0 & \dots & 0 \\ 0 & Msign(\frac{\partial f_2(x)}{\partial x_2}) & 0 & & \vdots \\ 0 & \ddots & \ddots & \ddots & \vdots \\ \vdots & & & & \\ 0 & \dots & 0 & 0 & Msign(\frac{\partial f_n(x)}{\partial x_n}) \end{pmatrix}.$$

Here, N and M are non-zero positive numbers.

The diagonal fixed point method is then:

Algorithm 2: Diagonal fixed point method

1. **Require** x_0, ε: precision.
2. **Ensure** x^* solution of $F(x) = 0$.
3. **State** $d(x)$ and $H(x)$ where H is the diagonal matrix.
4. **State** $dd = 0$ and $x = x_0$.
5. **While** $\|F(x)\| \geq \varepsilon$ and $dd < 100$.
6. **State** $x = x - \frac{1}{d(x)} H(x) F(x)$.
7. **State** dd=dd+1.
8. **EndWhile**
9. **If** $dd < 100$ **State** $x^* = x$.
10. **EndIf**

The same proof used for the theorems of Algorithm 2 can be applied to derive the following theorems.

Theorem 2.3. *Let x^* be a zero of F and suppose that:*

1. *F is C^1-differentiable in a neighborhood of x^*.*
2. $\forall k \in [\![1;n]\!]\ |\frac{\partial f_k(x^*)}{\partial x^k}| > \sum_{i=1,i\neq k}^{n} |\frac{\partial f_k(x^*)}{\partial x^i}|$.
3. $N \geq 1$.

There exists M_0 such that the sequence of Algorithm 2 converges to x^* for some initial guess x_0 in a neighborhood of x^*.

Theorem 2.4. *Let $F : U \to U$ where U is a compact set of $\mathbb{R}^n$, and suppose that:*

1. *$\Phi(U) \subset U$.*
2. *F is C^1-differentiable on U.*
3. *F has at least one zero x^*.*
4. $\forall x \in U, \forall k \in [\![1;n]\!];\ |\frac{\partial f_k(x)}{\partial x^k}| > \sum_{i=1,i\neq k}^{n} |\frac{\partial f_k(x)}{\partial x^i}|$.

Then, there exist M_0 and N_0 such that the sequence of Algorithm 2 converges to a zero of F.

3. Acceleration of Algorithm 2.5

To accelerate the convergence of Algorithm 2, one can apply Anderson acceleration or a line search method. Next, we propose a modified fixed-point method to accelerate Algorithm 2. This acceleration can help resolve the limitation caused by assumption 2 of Theorem 2.3. A version of the modified fixed-point method can be found in [21–23].

The new method consists of modifying Algorithm 2 as follows:

We decompose the Jacobian of F as:

$$F'(x) = -M(x) + N(x).$$

As a fixed-point method, Algorithm 2 can be written as:

$$\begin{aligned} x =& x - \frac{1}{d(x)} H(x)F(x), \qquad (3.1)\\ \Leftrightarrow & x = x - \frac{1}{d(x)} H(x)(F(x) + M(x) * x) + \frac{1}{d(x)} H(x) * M(x) * x\\ \Leftrightarrow & x - \frac{1}{d(x)} H(x) * M(x) * x = x - \frac{1}{d(x)} H(x)(F(x) + M(x) * x)\\ \Leftrightarrow & x = (I - \frac{1}{d(x)} H(x) * M(x))^{-1} * (x - \frac{1}{d(x)} H(x)(F(x) + M(x) * x)). \qquad (3.2) \end{aligned}$$

So the modified fixed point method is:

$$x_{k+1} = (I - \frac{1}{d(x_k)} H(x_k) * M(x_k))^{-1} * (x_k - \frac{1}{d(x_k)} H(x_k)(F(x_k) + M(x_k) * x_k)). \quad (3.3)$$

Here, $M(x)$ is generally a tridiagonal part of $F'(x)$ with 0 in the diagonal. The accelerated algorithm is described next:

Algorithm 3: Acceleration of the fixed point method

1. **Require** x_0, ε: precision.
2. **Ensure** x^* solution of $F(x) = 0$.
3. **State** $d(x)$ and $H(x)$ where H is the diagonal matrix.
4. **State** $dd = 0$ and $x = x_0$.
5. **While** $\|F(x)\| \geq \varepsilon$ and $dd < 100$.
6. **State** $M(x)$.
7. **State** $x(I - \frac{1}{d(x)}H(x) * M(x))^{-1} * (x - \frac{1}{d(x)}H(x)(F(x) + M(x) * x))$.
8. **State** dd=dd+1.
9. **EndWhile**
10. **If** $dd < 100$ **State** $x^* = x$.
11. **EndIf**

The Algorithm 3 is faster because:

$$\begin{aligned}\|\Phi'(x^*)\| &= \left\| \left(I - \frac{1}{d(x^*)}H(x^*)M(x^*) \right)^{-1} \left(x^* - \frac{1}{d(x^*)}H(x^*)(F'(x^*) + M(x^*)) \right) \right\| \\ &\leq \left\| \left(I - \frac{1}{d(x^*)}H(x^*)M(x^*) \right)^{-1} \right\| \cdot \left\| x^* - \frac{1}{d(x^*)}H(x^*)(F'(x^*) + M(x^*)) \right\| \\ &\ll 1.\end{aligned}$$

Because as in the proof of 2.1 we have

$$\|x^* - \frac{1}{d(x^*)}H(x^*)(F'(x^*) + M(x^*)\| << 1$$

and we can prove easily:

$$\|I - \frac{1}{d(x^*)}H(x^*) * M(x^*))^{(-1)}\| << 1$$

For illustration, consider a nonlinear reaction-diffusion equation:

$$\begin{cases} \Delta u(x) + R(u) = 0. \text{ in } \Omega = [0,1] \times [0,1]. \\ u = 0 \text{ on } \partial\Omega. \end{cases}$$

The discretization of this equation using a finite element or finite difference method leads to a nonlinear algebraic system $F(x) = 0$. We take $M(x) = F'(x) - diag(F'(x))$, where $diag(F'(x))$ is the diagonal part of $F'(x)$. This results in Algorithm 3.

The method converges quickly, but computing $(I - \frac{1}{d(x_k)}H(x_k) * M(x_k))^{-1}$ involves costly operations. A solution to this issue is to consider $M(x)$ as the tridiagonal part of $F'(x)$ with zeros on the diagonal. Another solution is to set $M(x) = M(x_0)$. The algorithm becomes:

Algorithm 5: Modified fixed point method

1. **Require** x_0, ε: precision.

2. **Ensure** x^* solution of $F(x) = 0$.
3. **State** $RR = F'(x_0) - diag(F'(x_0))$.
4. **State** $K = (I + \frac{1}{d(x_0)} H(x_0) * RR)^{-1}$.
5. **State** $dd = 0$ and $x = x_0$.
6. **While** $\|F(x)\| \geq \varepsilon$ and $dd < 100$.
7. **State** $x = K * (x - \frac{1}{d(x)} H(x)F(x) + RR)$.
8. **State** dd=dd+1.
9. **EndWhile**
10. **If** $dd < 100$ **State** $x^* = x$.
11. **EndIf**

4. Numerical simulation

In this section, we present test cases and compare the performance of our proposed algorithm with the Scilab command fsolve and the python command fsolve in solving nonlinear systems.

4.1. Some nonlinear system equations

The first example to consider is the function $F : \mathbb{R}^6 \to \mathbb{R}^6$ defined as:

$$F(x) = \begin{pmatrix} x_1^2 + x_3^2 - 1 \\ x_2^2 + x_4^2 - 1 \\ x_5 x_3^3 + x_6 x_4^3 \\ x_5 x_1^3 + x_6 x_2^3 \\ x_5 x_1 x_3^2 + x_6 x_2 x_4^2 \\ x_5 x_3 x_1^2 + x_6 x_4 x_2^2 \end{pmatrix}.$$

This jacobian of this function has a lot of singularities. Depending on the initial guess, a quai-Newton method will generally converge to a singular point instead of a zero of the function. A simple code in scilab of Algorithm 2 is:

```
function a=F(x)
...//Function F
endfunction
function aa=diff(x)
...// The jacobian of F
endfunction
//Define the matrix diff2(x)=(1/d(x))*H(x) of algorithm (2)
function bb=diff2(x)
     c=diff(x);
```

```
    h=sign(c');x=(M-2)*diag(h);b=h+diag(x);
    f=abs(c);y=diag(f);
     hh=sum(f,2);
    [ll,mm]=max(hh)
    dd=(M-1)*y(mm)+ll;
    bb=(1/dd)*b;
    endfunction
    eps=10^(-6)
dd=0
while norm(F(ss))>eps & dd <500
    dd=dd+1;
    if (det(diff(ss))>0.1 & norm(F(ss))>0.1 then
    ss=ss-diff2(ss)*F(ss);
else
    //ss=ss-diff(ss)^(-1)*F(ss); Newton acceleration
    yy=gmres(diff(ss),F(ss),300,10^(-8));
    ss=ss-yy;
    end;
    end
```

We take M=2 and the initial guess $x_0 =^t (2, 3, -1, 3, 4, 5)$. The number of iteration to reach the 10^{-8} precision is $n = 152$. The execution time is $t = 0.038493U$. The Newton methods implemented in the scipy packages of python like Krylov, Broyden and Levenberg-Marquardt's quasi-newton method will diverge.

We examine six different test cases, all involving nonlinear system equations. For each test case, we provide the function $F(x)$ and its Jacobian $F'(x)$, along with the corresponding Scilab code to compute the function and its Jacobian. The test cases are summarized as follows:

Case1: $f_i(x) = exp(-\sum_{j=1}^{n} x_j^2) + 20x_i - n$ pour $i = 1, 2, .., n$.

The scilab code to compute the function $F(x)$ and its jacobian $F'(x)$:

```
function a=F(x)
    s=sum(x^2);
    a=exp(-s)+20*x-M
endfunction
function b=diff(x)
s=sum(x^2);
b=-2*exp(-s)*repmat(x,1,M)'+20*diag(ones(1,M))
endfunction
```

Case2: $f_i(x) = \sum_{j=1}^{n} x_j^2 + nix_i - 2n$ pour $i = 1, 2, .., n$.

The scilab code to compute the function $F(x)$ and its jacobian $F'(x)$:

```
M=1000;
function a=F(x)
    s=sum(x^2);
    a=s+M*[1:M]'.*x-2*M
endfunction
```

```
function b=diff(x)
b=(2*repmat(x,1,M))'+M*diag([1:M])
endfunction
```

Case3: $f_i(x) = n - \sum_{j=1}^{n} \cos(x_j) + i(1 - \cos(x_i)) - \sin(x_i)$ pour $i = 1, 2, .., n.$

The scilab code to compute the function $F(x)$ and its jacobian $F'(x)$:

```
M=1000;
function a=F(x)
s=sum(cos(x));
a=M-s-sin(x)+[1:M]'.*(1-cos(x));
endfunction
function b=diff(x)
b=repmat(sin(x),1,M)'+diag([1:M]'.*sin(x)-cos(x));
endfunction
```

Case4: $f_i(x) = log(1 + \sum_{j=1}^{n} x_j^2) + x_i - 1)$ pour $i = 1, 2, .., n.$

The scilab code to compute the function $F(x)$ and its jacobian $F'(x)$:

```
M=1000;
function a=F(x)
    s=sum(x^2);
    a=log(1+s)+x-1;
endfunction
function b=diff(x)
s=sum(x^2);
b=(2/(1+s))*(repmat(x,1,M))'+diag(ones(1,M));
endfunction
```

Case5: $f_i(x) = x_i^4 - 1 - \frac{i}{n} \sum_{j=1}^{n} j(x_j - 1) - \frac{2i}{n^2} \left(\sum_{j=1}^{n} j(x_j - 1) \right)^3$ pour $i = 1, 2, .., n.$

The scilab code to compute the function $F(x)$ and its jacobian $F'(x)$:

```
M=1000;
function a=F(x)
s1=sum([1:M]'.*(x-1)); s2=sum([1:M]'.*(x-1));
a=x^4-1-(1/M)*s1*[1:M]'-(2/M^2)*s2^(3)*[1:M]';
endfunction
function b=diff(x)
s2=sum([1:M]'.*(x-1));
b=4*diag(x^3)-(1/M)*[1:M]'*[1:M]-(6/M^2)*s2^(2)*([1:M]')*[1:M]
endfunction
```

Case6: $f_i(x) = x_i - 1 + i \sum_{j=1}^{n} j(x_j - 1) + 2i \left(\sum_{j=1}^{n} j(x_j - 1) \right)^3$ pour $i = 1, 2, .., n.$

We apply Algorithm 3 because the assumption 2 of Theorem 2.1 is not realised.
The scilab code to compute the function $F(x)$ and its jacobian $F'(x)$:

```
M=1000;
function a=F(x)
    s1=sum([1:M]'.*(x-1));
   a=x-1+s1*[1:M]'+2*s1^(3)*[1:M]';
endfunction
function b=diff(x)
s2=sum([1:M]'.*(x-1));
b=diag(ones(M,1))+[1:M]'*[1:M]+6*s2^(2)*([1:M]')*[1:M];
endfunction
```

For each case, we solve the nonlinear system using both our proposed method and the fsolve command from Scilab and Python.

Test Execution:

The numerical tests were performed for n=1000, with the initial guess $x_0 = 20 \times^t (1, 1, ..., 1)$. The precision was set to $\varepsilon = 10^{-6}$. The number of iterations and the time taken for execution in the CPU were measured for both methods.

Results:

The results of the comparison between our method and fsolve command of scilab and Python are summarized in Table 1 below:

Table 1. Results between the proposed method and the command fsolve of scilab

Cases:	Case1	Case2	Case3	Case4	Case5	Case6
Number of iteration to $\epsilon = 10^{-6}$	1	6	15	5	32	26
Time Execution in CPU	0.219	0.267	1.426	0.944	5.90	2 .431
Time execution (cpu)for scilab fsolve	0.67	0.697	6.16	0.696	5.901	diverge
Time execution (cpu)for Scipy fsolve	1.036	1.049	diverge	diverge	diverge	diverge

From Table 1, we observe the following:

- For small value of n=1000, both our method and fsolve perform similarly in terms of time execution, with minor differences in the number of iterations required.
- The time for execution in the CPU for our method is generally lower than that of fsolve in most cases.
- For large value of n=10000, fsolve experiences significant delays, taking more than 10 minutes to complete, while our method remains efficient, taking only between 100 and 200 seconds.

Conclusion:

The numerical simulation results demonstrate the efficiency of our proposed algorithm in solving nonlinear systems compared to the Scilab fsolve method. Our method not only achieves faster execution times but also handles larger problem sizes more efficiently. This makes it a suitable choice for large-scale problems, where traditional methods like fsolve may become computationally expensive.

4.1.1. Resolution of equations that come from partial differential equations

In this section, we consider examples of nonlinear partial differential equations (PDEs) that are discretized using a finite difference scheme. The resulting algebraic equations form nonlinear systems, which we solve using the proposed method (Algorithm 3) and the Scilab and Python commands fsolve. We then compare the execution times of both methods.

The examples are listed below:

Equation1: The problem is a one-dimensional compressible-flow problem describing transonic flow in a duct. The PDE is given by:

$$\begin{cases} [A(x)\rho(u)u_{x]_x} = 0. \text{ in } [0,2]. \\ \qquad u(0) = 0 \qquad u(2) = u_R. \end{cases}$$

where $\rho(u) = [1+\frac{\gamma-1}{2}(1-u^2)]^1/(\gamma-1)$ is the density and $A(x) = 0.4+0.6(x-1)^2$ prescribes the cross-section of the duct. The solution u is the potential and u_x is the velocity of the flow. We take $u_R = 1.15$ and $\gamma = 1.4$ and apply the finite-difference discretization. The following code describes the obtained nonlinear equation $F(x) = 0$ and its differential.

We take for M, N and n the values $M = n = 1000$, $N = 1$ and the initial condition $x_0 =^t [0, 0, ..., 0]$.

Equation2:

$$\begin{cases} \Delta u(x) - 6e^u = 0. \text{ in } \Omega = [0,1] \times [0,1]. \\ \qquad\qquad u = 0 \text{ on } \partial\Omega. \end{cases}$$

We take for M, N and n the values $M = n = 126 \times 126$, $N = 1$ and we take for the initial condition $x_0 =^t [0, 0, ..., 0]$.

The following Scilab code illustrates the finite difference discretization for solving these equations using the proposed algorithm:

```
function a=F(x)
    y=[0;x(1:M^2-1)];
    yy=[zeros(M,1);x(1:M^2-M)];
    rr=[x(M+1:M^2);zeros(M,1)];
    r=[x(2:M^2);0];
    a=h*(yy+rr+y+r-4*x)+2*exp(x)//-6*exp(x);//-sin(x)-1;
    //a=h*(yy+rr+y+r-4*x)-1.5*x^2;
    //a(1)=a(1)+h*1;a(M^2)=a(M^2)+h*1;
            endfunction
function bb=diff2(x)
       b=-M^2*diag(ones(1,M^2));
       dd=(M^2)*(4*h+2*max(exp(x)))+1;
       bb=(1/dd)*b;
       endfunction
a0=0*ones(M^2,1);
ss=a0;
eps=10^(-6)
dd=0
```

```
cc1=ones(M^2-1,1);cc2=ones(M^2-M,1);
MM=diag(cc1,1)+diag(cc2,M)+diag(cc1,-1)+diag(cc2,-M);
RR=h*diff2(rr)*MM;
K= (eye(M^2,M^2)+RR)^(-1)
tic()
while norm(F(ss))>eps & dd <1000
    dd=dd+1;
    d(dd)=ss(1);
    ss=K*(ss-diff2(ss)*F(ss)+RR*ss)
end
t2=toc()
```

Equation3: This equation is the 2D problem:

$$\begin{cases} \Delta u - \dfrac{3}{2}u^2 = 0. \text{ in } [0,1]\times[0,1]. \\ \qquad\qquad u = 1 \text{ on } \partial\Omega. \end{cases}$$

We take $M = n = 126 \times 126$, N=1 and the initial condition $x_0 =^t [0, 0, ..., 0]$.
Equation4: This equation is called the 2D Bratu equation. The problem is:

$$\begin{cases} \Delta u + \lambda e^u = 0. \text{ in } \Omega = [0,1]\times[0,1]. \\ \qquad\qquad u = 0 \text{ on } \partial\Omega. \end{cases}$$

We take $\lambda = 3$, $M = n = 126 \times 126$, $N = 1$ and the initial condition $x_0 =^t [0, 0, ..., 0]$.
Equation5: This equation is the 3D Bratu equation. The problem is:

$$\begin{cases} \Delta u + e^u = 0. \text{ in } \Omega = [0,1]\times[0,1]\times[0,1]. \\ \qquad\qquad u = 0 \text{ on } \partial\Omega. \end{cases}$$

We take $\lambda = 3$, $M = n = 30 \times 30 \times 30$, $N = 1$ and the initial condition $x_0 =^t [0, 0, ..., 0]$.
Equation6: We consider the 3D problem:

$$\begin{cases} \Delta u + \sin(u) - 1 = 0. \text{ in } \Omega = [0,1]\times[0,1]\times[0,1]. \\ \qquad\qquad\qquad u = 0 \text{ on } \partial\Omega. \end{cases}$$

We take $\lambda = 3$, $M = n = 30 \times 30 \times 30$, N=1 and the initial condition $x_0 =^t [0, 0, ..., 0]$.

Table 2 presents a comparison of our algorithm and the fsolve command in Scilab for solving the equation $F(x) = 0$ for Equations 1 through 6. We compare the CPU execution time for each method.

From Table 2, we observe the following:

- For small n=10, both our method and fsolve perform similarly in terms of execution time, with minor differences in the number of iterations required.

Table 2. Results between the proposed method and the command fsolve of scilab

Cases:	Equation1	Equation2	Equation3	Equation4	Equation5	Equation6
Number of iteration to $\epsilon = 10^{-6}$	5	12	5	41	21	9
Time Execution in CPU	0.456	12.290	4.850	40.522	63.919	27.353
Time execution for scilab's fsolve	1.283	More than 10 min	$>$ 10 min	$>$ 10 min	$>$ 10 min	$>$ 10 min
Time execution for Scipy's fsolve	9.06	More than 10 min	$>$ 10 min	$>$ 10 min	$>$ 10 min	$>$ 10 min

- The time for execution in the CPU for our method is generally lower than that of fsolve in most cases.
- For large $n = 126 \times 126$, fsolve experiences significant delays, taking more than 10 minutes to complete, while our method remains efficient, taking only between 100 and 200 seconds.

Conclusion:

The numerical simulation results demonstrate the efficiency of our proposed algorithm in solving nonlinear systems compared to the Scilab or Scipy's fsolve method.

5. Conclusion

In this study, we have presented a novel modified fixed-point method for solving nonlinear systems of equations. We rigorously established the global and local convergence of the proposed method and demonstrated its efficiency through extensive testing. To further improve performance, we accelerated the method by modifying the underlying equations. The results indicate that the method is highly effective for large-scale systems, offering a scalable solution with favorable computational complexity compared to existing methods. This makes it a cost-efficient alternative for solving complex nonlinear problems.

Looking ahead, we plan to extend this approach to tackle global optimization challenges and apply it to solve sophisticated models in mechanics, such as thermo-elasto-plastic systems. The versatility and scalability of the proposed method open up exciting possibilities for its application in diverse fields requiring high-performance computational solutions.

References

[1] Peter W. Christensen. *A nonsmooth Newton method for elastoplastic problems*, Computer Methods in Applied Mechanics and Engineering. Volume 191, Issues 11-12, 4 January 2002, Pages 1189–1219.

[2] Zerah, G. *An Efficient Newton's Method for the Numerical Solution of Fluid Integral Equations. Novembre 1985*, Journal of Computational Physics, Volume 61, Issue 2, p. 280–285.

[3] Editor Alwyn C. Scott. *Nonlinear Biology. The nonlinear universe*, Chaos, Emergence, Life. The frontier collection, Springer, Berlin (2007), pp. 181–276.

[4] Householder, A. S. *Principles of Numerical Analysis*, McGraw-Hill, New York, 1953, pp. 132–137.

[5] Peter Berzi. *Convergence and Stability Improvement of Quasi-Newton Methods by Full-Rank Update of the Jacobian Approximates*, AppliedMath 2024, 4(1), 143–181, https://doi.org/10.3390/appliedmath4010008.

[6] Kokurin, M.M., Kokurin, M.Y., Semenova, A.V. *Iteratively regularized Gauss-Newton type methods for approximating quasi-solutions of irregular nonlinear operator equations in Hilbert space with an application to COVID-19 epidemic dynamics*, Appl. Math. Comput. 2022, 431, 127312.

[7] Indrapriyadarsini, S., Mahboubi, S., Ninomiya, H., Kamio, T., Asai, H. *Accelerating Symmetric Rank-1 Quasi-Newton Method with Nesterov's Gradient for Training Neural Networks*, Algorithms 2022, 15, 6.

[8] Dennis, J.E., More, J.J. *A Characterization of Superlinear Convergence and its Application to Quasi-Newton Methods*, Math Comput. 28(126), 549–560 (1974).

[9] Schubert, L. *Modification of quasi-Newton method for nonlinear equations with a sparse Jacobian*, Math Comp. 24(109), 27–30 (1970).

[10] Rosen, E. M. *A review of quasi-Newton methods in nonlinear equation solving and unconstrained optimization*, Proc. 21st Nat. Conference of the ACM. Thompson Book Co., Washington, D. C., 1966, pp. 37–41.

[11] Weijun Zhou. *A globally convergent BFGS method for symmetric nonlinear equations*, Journal of Industrial and Management Optimization, 2022, 18(2): 1295–1303. doi: 10.3934/jimo.2021020.

[12] H. H. Dwail, M. A. Shiker. *Using Trust Region Method with BFGS Technique for Solving Nonlinear Systems of Equations*, Journal of Physics: Conference Series, vol. 1818, 2021.

[13] Deuflhard, P. *Global Inexact Newton Methods for Very Large Scale Nonlinear Problems*, Impact of Computing Science and Engineering. 3(4), 366–393 (1991).

[14] Barnafi, N.A., Pavarino, L.F., Scacchi, S. *Parallel inexact Newton-Krylov and Quasi-Newton Solvers for Nonlinear Elasticity*, arXiv. Available online: https://arxiv.org/abs/2203.05610 (accessed on 15 December 2023).

[15] Walker, Homer F., Ni, Peng. *Anderson Acceleration for Fixed-Point Iterations*, SIAM Journal on Numerical Analysis. 49 (4): 1715–1735 (2011).

[16] Anderson, Donald G. *Iterative Procedures for Nonlinear Integral Equations*, Journal of the ACM. 12 (4): 547–560 (1965).

[17] K. Amini, M. A. Shiker and M. Kimiaei. *A line search trust-region algorithm with nonmonotone adaptive radius for a system of nonlinear equations*, 4OR-Journal of Operation Research, vol. 14, no. 2, pp. 133–152, 2016.

[18] H. Jiang and D. Ralph. *Global and local superlinear convergence analysis of Newton-type methods for semismooth equations with smooth least squares*, Reformulation: nonsmooth, piecewise smooth, semismooth and smoothing methods. Springer, Boston, MA, pp. 181–209, 1998.

[19] J. Schauder. *Der Fixpunktsatz in Funktionalraumen*, Studia Math., vol. 2, 1930, p. 171–180.

[20] F. Boyer. *Théorèmes de point fixe et applications*, CMI Université Paul Cézanne (2008-2009).

[21] M. R. Amattouch, Belhadj Hassan. *A modified fixed point method for biochemical transport*, Mathematics Boletim da Sociedade Paranaense de Matemática, (3s.) v. 2022 (40): 1–5. 2 February 2022.

[22] M.R. Amattouch, H. Belhadj. *Combined Optimized Domain Decomposition Method and a Modified Fixed Point Method for Non Linear Diffusion Equation*, Applied Mathematics and Information Sciences, 11, No. 1, 201–207 (2017).

[23] M.R. Amattouch, N. Nagid, H. Belhadj. *A modified fixed point method for The Perona Malik equation*, Journal of Mathematics and System Science 7, 175–185, September 2017.