

High-Precision Solution of Nonlinear Equations by Halley's Method

Zhong Zhou^{1, a}

¹College of Mathematics and Information Science, Zhongyuan University of Technology,
Zhengzhou 450007, China

^asouthmonarch@qq.com

Abstract

Although Newton's method is widely used for solving nonlinear equations and systems of nonlinear equations due to its quadratic convergence, it may become inefficient in settings where a single iteration is computationally expensive or where solutions must be computed at precisions far beyond standard 64-bit double precision. In such cases, Halley's method, which achieves cubic convergence, can offer better overall efficiency. This work investigates Halley-type methods for both scalar nonlinear equations and multivariate nonlinear systems, establishes several lemmas and corollaries, and provides high-precision implementations based on the Python Mpmath library. Numerical experiments show that, despite the higher per-step cost, Halley's method can require fewer iterations and less total computation time than Newton's method when high-precision solutions are required.

Keywords

Halley's Method; Systems of Nonlinear Equations; High-Precision.

1. Introduction

In the late 17th century, Isaac Newton and Joseph Raphson independently developed similar iterative procedures for solving nonlinear equations; both schemes exhibit quadratic convergence and are now commonly known as Newton's method or the Newton-Raphson method [1-2]. For simplicity, this paper refers to it as Newton's method. In the 18th century, Edmond Halley improved this approach to achieve cubic convergence [3]. Later, some numerical computing scientists developed and improved Halley's method [4-5], but no major breakthroughs have been seen. Although the convergence rate is substantially higher, Halley's method requires not only the first derivative but also the second derivative. For systems of nonlinear equations, it further requires the Jacobian matrix and a third-order derivative structure (often represented as a Hessian tensor), so the computational cost and implementation complexity are considerably greater than those of Newton's method. This is likely the main reason why Newton's method remains more popular and mainstream in practice.

Another reason is that in many applications a single Newton iteration is cheap, and the required solution accuracy is modest; in such cases quadratic convergence is already sufficiently fast, and the advantage of cubic convergence is not fully exploited.

However, we observe that in some problems—such as the word-lifting technique embedded in trimodal maps of symbolic dynamics, where Newton's method is used to compute the three parameters corresponding to super-stable kneading sequences—the study of metric universality in n -tupling period sequences makes each Newton iteration very expensive [6], while the solution itself must be obtained at extremely high precision. In these settings,

replacing Newton's method with Halley's method can exploit cubic convergence and improve the overall efficiency of reaching the fixed-point solution.

This paper does not address the replacement of Newton's method inside the word-lifting technique of symbolic dynamics. Instead, we argue that whenever a nonlinear equation or system satisfies two conditions:

1. The solution requires ultra-high (arbitrary) precision;
2. A single iteration is computationally expensive;

then replacing Newton's method by a Halley-type method is likely to improve total efficiency.

With the maturity of arbitrary-precision arithmetic—e.g., the C-based MPFR library, the C++ NTL library, and the Python mpmath library (multiple-precision mathematics library)—the hardware limitation of fixed 64-bit floating point is no longer a bottleneck. Notably, combining arbitrary-precision arithmetic with Halley's method can also greatly reduce error accumulation from matrix inversion and improve numerical stability.

2. The Halley-Type Method

2.1. Halley's Formular for Scalar Nonlinear Equation

2.1.1. The Basic Idea

Assume $f: \mathbb{R} \rightarrow \mathbb{R}$, $f \in C^n$, $n \geq 3$, x^* is the zero point of f , $f(x^*) = 0$ and $f'(x^*) \neq 0$. The idea of Halley's method is to approximate using a rational function (hyperbolic type), rather than a linear function as in Newton's method. This paper does not adopt this approach; instead, we derive the method using Taylor expansion and Halley-type approximation.

2.1.2. Derivation of Halley's Iteration Formula

Assume x^* is the single root of equation $f(x) = 0$, that is $f(x^*) = 0$, in addition, $f'(x^*) = a \neq 0$. We note $f_k = f(x_k)$, $f'_k = f'(x_k)$, $f''_k = f''(x_k)$ for conciseness. Perform a second-order Taylor expansion at x_k , let $h = x - x_k$,

$$0 = f(x_k + h) \approx f_k + f'_k h + \frac{f''_k}{2} h^2 \quad (1)$$

Newton's method only retains the linear term, taking $h_N = f_k / f'_k$. If the quadratic term is retained, the above equation can be rewritten in fixed-point form:

$$h = -\frac{f_k}{f'_k + \frac{f''_k}{2} h} \quad (2)$$

The key trick of Halley-type approximation is to substitute the Newton correction term $h_N = f_k / f'_k$ for h on the right-hand side as an approximation in (2):

$$h \approx -\frac{f_k}{f'_k + \frac{f''_k f_k}{2 f'_k}} = -\frac{2 f_k f'_k}{2 (f'_k)^2 + f_k f''_k} \quad (3)$$

Thus, (3) is regarded as Halley iteration step, we obtain the Halley iteration formular (4):

$$x_{k+1} = x_k - \frac{2f_k f'_k}{2(f'_k)^2 + f_k f''_k} \quad (4)$$

Theorem 1 (Cubic Convergence of Halley's Method). Assume $f: \mathbb{R} \rightarrow \mathbb{R}$ is three times continuously differentiable in a neighborhood of a simple root x^* , with $f(x^*) = 0$ and $f'(x^*) \neq 0$. Then the Halley iteration (4) is locally convergent with cubic order. Specifically,

$$\lim_{k \rightarrow \infty} \frac{|e_{k+1}|}{|e_k|^3} = C = \left| \frac{f''(x^*)^2}{4f'(x^*)^2} - \frac{f'''(x^*)}{6f'(x^*)} \right| \quad (5)$$

Proof: Assume x^* is the single root of f , that is $f(x^*) = 0$, in addition, $f'(x^*) \neq 0$.

According to the Halley formular (4), the iteration form is simplified as:

$$x_{k+1} = \varphi(x_k) \quad (6)$$

Simultaneously, another equivalent form of (4) is written as (7):

$$x_{k+1} = x_k - \frac{f_k}{f'_k} \cdot \frac{1}{1 - \frac{f_k f''_k}{2(f'_k)^2}} \quad (7)$$

Here, iterative map φ is defined as:

$$\begin{cases} \varphi(x) = x - \frac{u(x)}{1 - w(x)}, \\ u(x) = \frac{f(x)}{f'(x)}, \\ w(x) = \frac{f(x)f''(x)}{2(f'(x))^2}. \end{cases} \quad (8)$$

Let $f'(x^*) = a \neq 0$, $f''(x^*) = p$, $f'''(x^*) = q$. Consider the values of $u(x)$ and $w(x)$ at x^* .

$\because u(x) = \frac{f}{f'}$, $\therefore u'(x) = \frac{(f')^2 - f \cdot f''}{(f')^2}$. Substituting x^* for x in $f(x)$, $f'(x)$, $f''(x)$, and $u(x)$, the

following results are obtained: $u(x^*) = 0$ and $u'(x^*) = \frac{a^2}{a^2} = 1$.

$w(x) = \frac{f(x)f''(x)}{2(f'(x))^2} \Rightarrow w'(x) = \frac{f'(x)f''(x) + f(x)f'''(x)}{2(f'(x))^2}$, Substituting x^* for x in $w(x)$ and $w'(x)$,

so $w(x^*) = 0$ and $w'(x^*) = \frac{f'(x^*)f''(x^*)}{2(f'(x^*))^2} = \frac{p}{2a}$.

First, $\varphi(x^*) = x^*$ is easily obtained by (8) and $u(x^*) = 0$, so x^* is the fixed-point of iterative equation (6).

Second, let $\varphi(x) = x - v(x)$, $v(x) = \frac{u(x)}{1-w(x)}$, then $\varphi'(x) = 1 - v'(x)$. $v'(x) = \frac{u'(1-w) + uw'}{(1-w)^2}$,

substituting x^* for x in $\varphi'(x)$ and $v'(x)$ by $u(x^*) = 0$ and $w(x^*) = 0$, so $v'(x^*) = \frac{u'(1-0) + 0}{(1-0)^2}$

$= u'(x^*) = 1$, then $\varphi'(x^*) = 1 - v'(x^*) = 1 - 1 = 0$

Third, $v''(x) = \frac{2u'w'(1-w) + u''(1-w)^2 + uw''(1-w) + 2u(w')^2}{(1-w)^3}$, substituting x^* for x in $v''(x)$ by

$u(x^*) = 0$, $w(x^*) = 0$, $u'(x^*) = 1$ and $w'(x^*) = \frac{p}{2a}$, so $v''(x^*) = 2u'(x^*)w'(x^*) + u''(x^*) = \frac{p}{a} + u''(x^*)$

holds.

From (8), $u''(x) = \frac{2f \cdot (f'')^2 - f \cdot f' \cdot f''' - f'' \cdot (f')^2}{(f')^3}$, then substituting x^* for x in $u''(x)$ by

$f(x^*) = 0$, $f'(x^*) = a \neq 0$ and $f''(x^*) = p$, so $u''(x^*) = -\frac{f''(x^*)}{f'(x^*)} = -\frac{p}{a}$, then $v''(x^*) = \frac{p}{a} - \frac{p}{a} = 0$.

From $\varphi'(x) = 1 - v'(x)$, $\varphi''(x) = -v''(x)$, then $\varphi''(x^*) = -v''(x^*) = 0$ holds. By the way, $\varphi'''(x^*)$ should be calculated.

$$\begin{cases} \varphi'''(x) = -v'''(x) = -\frac{3w'' \cdot u'}{(1-w)^2} - \frac{6(w')^2 \cdot u'}{(1-w)^3} - \frac{u''}{1-w} - \frac{3u'' \cdot w'}{(1-w)^3} - \frac{6u \cdot (w')^3}{(1-w)^4} - \frac{u \cdot w''}{(1-w)^2} - \frac{6u \cdot w' \cdot w''}{(1-w)^3} \\ \varphi'''(x^*) = -3w''(x^*) - 6(w'(x^*))^2 - u''(x^*) - 3u''(x^*) \cdot w'(x^*) \end{cases} \quad (9)$$

$$\begin{cases} u'''(x) = \frac{3(f'')^2}{(f')^2} - \frac{2f'''}{f'} - \frac{f \cdot f^{(4)}}{(f')^2} - \frac{6f \cdot (f'')^3}{(f')^4} + \frac{6f \cdot f''' \cdot f''}{(f')^3} \\ u'''(x^*) = \frac{3(f''(x^*))^2}{(f'(x^*))^2} - \frac{2f'''(x^*)}{f'(x^*)} = \frac{3p^2}{a^2} - \frac{2q}{a} \end{cases} \quad (10)$$

$$\begin{cases} w''(x) = \frac{f'''}{f'} - \frac{3(f'')^2}{2(f')^2} + \frac{f \cdot f^{(4)}}{2(f')^2} + \frac{3f \cdot (f'')^3}{(f')^4} - \frac{3f \cdot f''' \cdot f''}{(f')^3} \\ w''(x^*) = \frac{f'''(x^*)}{f'(x^*)} - \frac{3(f''(x^*))^2}{2(f'(x^*))^2} = \frac{q}{a} - \frac{3p^2}{2a^2} \end{cases} \quad (11)$$

Considering the result of (10) and (11),

$\varphi'''(x^*) = -\left(\frac{3q}{a} - \frac{9p^2}{2a^2} + \frac{3p^2}{2a^2} + \frac{6p^2}{2a^2} - \frac{2q}{a} - \frac{3p^2}{2a^2}\right) = \frac{3p^2}{2a^2} - \frac{q}{a}$, so $\varphi'''(x^*) = \frac{3f''(x^*)^2}{2f'(x^*)^2} - \frac{f'''(x^*)}{f'(x^*)}$ holds.

Final, near the fixed-point x^* , the error $e_k = x_k - x^*$, noting that $\varphi(x^*) = x^*$, $\varphi'(x^*) = 0$ and $\varphi''(x^*) = 0$:

$$\begin{aligned}
 x_{k+1} &= \varphi(x_k) = \varphi(x^* + e_k) = \varphi(x^*) + \varphi'(x^*)e_k + \frac{\varphi''(x^*)}{2!}e_k^2 + \frac{\varphi'''(x^*)}{3!}e_k^3 + O(e_k^4) \\
 &= x^* + \frac{\varphi'''(x^*)}{3!}e_k^3 + O(e_k^4)
 \end{aligned} \tag{12}$$

It is easily to get the following result:

$$\lim_{k \rightarrow \infty} \frac{|e_{k+1}|}{|e_k|^3} = \lim_{k \rightarrow \infty} \frac{|x_{k+1} - x^*|}{|e_k|^3} = \frac{1}{6} |\varphi'''(x^*)| = \left| \frac{f''(x^*)^2}{4f'(x^*)^2} - \frac{f'''(x^*)}{6f'(x^*)} \right| = C \tag{13}$$

Thus, the Halley iteration (4) is locally convergent with cubic order.

The proof is over.

Theorem 2. (Degeneracy of Halley's method at multiple roots and its modification)

Let f be three times continuously differentiable in a neighborhood of x^* , and let x_k be generated from a starting point x_0 sufficiently close to x^* .

(i) (Degeneracy at multiple roots) For the classical Halley iteration (4), supposing x^* is a root of f of multiplicity m , i.e., $f(x) = (x - x^*)^m g(x)$, $g(x^*) \neq 0$, then the method degenerates to first-

order (linear) convergence: $\lim_{k \rightarrow \infty} \frac{x_{k+1} - x^*}{x_k - x^*} = \frac{m-1}{m+1} \in (0,1)$.

(ii) (Modified Halley method for multiple roots.) Assume the multiplicity m of the root is known. Then the modified iteration (14) is as following:

$$x_{k+1} = x_k - \frac{2m \cdot f(x_k) f'(x_k)}{(m+1) \cdot f'(x_k)^2 - m \cdot f(x_k) f''(x_k)} \tag{14}$$

(14) holds the cubic order convergence.

2.2. Halley's Formular for Multivariate Systems of Nonlinear Equations

Halley's method for scalar nonlinear equations is relatively straightforward to understand. In contrast, the extension to systems of nonlinear equations is considerably more complex and challenging, yet it is also the case with the broadest range of applications. For this reason, after a theoretical investigation of the multivariate setting, we focus our implementation of the high-precision Halley algorithm on systems of nonlinear equations.

2.2.1. Notation and Mathematic Definition

Consider the system of nonlinear equations:

$F(x) = 0$, where $F: \mathbb{R}^n \rightarrow \mathbb{R}^n$, $x = (x_1, x_2, \dots, x_n)^T \in \mathbb{R}^n$, $F(x) = (f_1(x), f_2(x), \dots, f_n(x))^T$, the objective is to find x^* such that $F(x^*) = 0$.

For a vector-valued function, Taylor expansion of F near x_k has the following vector form:

$$F(x_k + h) = F(x_k) + J(x_k)h + \frac{1}{2}H(x_k)[h, h] + O(\|h\|^3) \tag{15}$$

Where $J(x_k) \in \mathbb{R}^{n \times n}$ is the Jacobian matrix which is defined as: $[J(x)]_{ij} = \frac{\partial f_i}{\partial x_j}(x)$, the corresponding matrix form is:

$$J(x) = \frac{\partial F}{\partial x} = \frac{\partial(f_1, f_2, \dots, f_n)}{\partial(x_1, x_2, \dots, x_n)} = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \dots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \dots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \dots & \frac{\partial f_n}{\partial x_n} \end{pmatrix} \quad (16)$$

The Hessian tensor $H(x) \in \mathbb{R}^{n \times n \times n}$ is a third-order tensor whose entries are given by:

$[H(x)]_{ijl} = \frac{\partial^2 f_i}{\partial x_j \partial x_l}(x)$, the slice representation of Hessian tensor can be viewed as a stack of n

Hessian matrix. For each i , defined $H_i(x) \in \mathbb{R}^{n \times n}$ by $[H_i(x)]_{jl} = \frac{\partial^2 f_i}{\partial x_j \partial x_l}(x)$. The tensor-vector contraction $H(x_k)[h, h]$ produces a n -dimensional vector whose i -th component is:

$$[H(x_k)[h, h]]_i = \sum_{j=1}^n \sum_{l=1}^n \frac{\partial^2 f_i}{\partial x_j \partial x_l} \bigg|_{x_k} h_j h_l = h^T H_i(x_k) h, \text{ equivalently, in vector form:}$$

$$H(x_k)[h, h] = \begin{pmatrix} h^T H_1(x_k) h \\ h^T H_2(x_k) h \\ \vdots \\ h^T H_n(x_k) h \end{pmatrix} \quad (17)$$

2.2.2. Derivation of Halley's Method for Multivariate Systems of Nonlinear Equations

Truncating the Taylor expansion to second order yields:

$$F(x_k + h) \approx F_k + J_k h + \frac{1}{2} H_k[h, h], \text{ Setting this to zero gives:}$$

$$F_k + J_k h + \frac{1}{2} H_k[h, h] = 0 \quad (18)$$

Let $s_k = -J_k^{-1} F_k$ denote the famous Newton step, then $F_k = -J_k s_k$. Substituting into the quadratic approximation (18):

$$J_k(h - s_k) + \frac{1}{2} H_k[h, h] = 0 \quad (19)$$

Write $h = s_k + \delta$, where δ is a correction term which is called the Halley step, once the δ is calculated out, the total Halley iteration step h is obtained. Substituting into (19):

$$J_k \delta + \frac{1}{2} H_k[s_k + \delta, s_k + \delta] = 0 \text{ and expanded as } J_k \delta + \frac{1}{2} H_k[s_k, s_k] + H_k[s_k, \delta] + \frac{1}{2} H_k[\delta, \delta] = 0,$$

Since δ is one order smaller than s_k , the higher-order terms $H_k[s_k, \delta]$ and $\frac{1}{2} H_k[\delta, \delta]$ may be neglected as:

$$J_k \delta + \frac{1}{2} H_k[s_k, s_k] = 0 \quad (20)$$

So, $\delta \approx -\frac{1}{2} J_k^{-1} H_k[s_k, s_k]$ is the so-called Halley correction step, s_k is the Newton step, $h = s_k + \delta$ is the full Halley step.

Theorem 3 (Cubic Convergence of the Multivariate Halley Method)

Let $F: \mathbb{R}^n \rightarrow \mathbb{R}^n$ be a three-times continuously differentiable function in a neighborhood of a simple root $x^* \in \mathbb{R}^n$, i.e, $F(x^*) = 0$ and the Jacobian matrix $J(x^*)$ is nonsingular. Then there exists a neighborhood $U \in \mathbb{R}^n$ of x^* such that for any initial guess $x_0 \in U$, the sequence $\{x_k\}_{k=0}^{\infty}$ generated by the multivariate Halley iteration

$$x_{k+1} = x_k + s_k + t_k \quad (21)$$

Where, $s_k = -J_k^{-1} F_k$ is the Newton step and $t_k = -\frac{1}{2} J_k^{-1} H_k[s_k, s_k]$ is the Halley correction step, converges to x^* with cubic order. Other notations may see Equations (15)–(20) and the definitions in the surrounding text.

Considering the length of the paper, the proof of Theorem 3 is omitted here. We enhance the application of the cubic convergence of the multivariate Halley method.

3. Algorithm Flowchart and High-Precision Python Implementation of the Multivariate Halley Method

3.1. Algorithm Flowchart of the Multivariate Halley Method

Before diving into the implementation details, it is worth emphasizing the importance of a well-designed flowchart. The multivariate Halley's method involves a sequence of non-trivial computational steps — evaluating the function, computing the Jacobian and Hessian, solving linear systems, and updating the iterate. Translating the abstract mathematical formulation directly into executable code can be error-prone and mentally taxing, especially when dealing with tensor operations and convergence checks.

A flowchart serves as a visual bridge between theory and practice. It decomposes the complex iterative procedure into a clear, step-by-step logical structure.

3.2. Mpmath Library and Python Implementation

The multivariate Halley's method described in Section 3.1 is implemented in Python using the Mpmath library for arbitrary-precision arithmetic. Mpmath provides essential tools for high-precision numerical computation, including automatic differentiation via the `mp.diff` function and matrix operations via `mp.matrix` and `mp.lu_solve`.

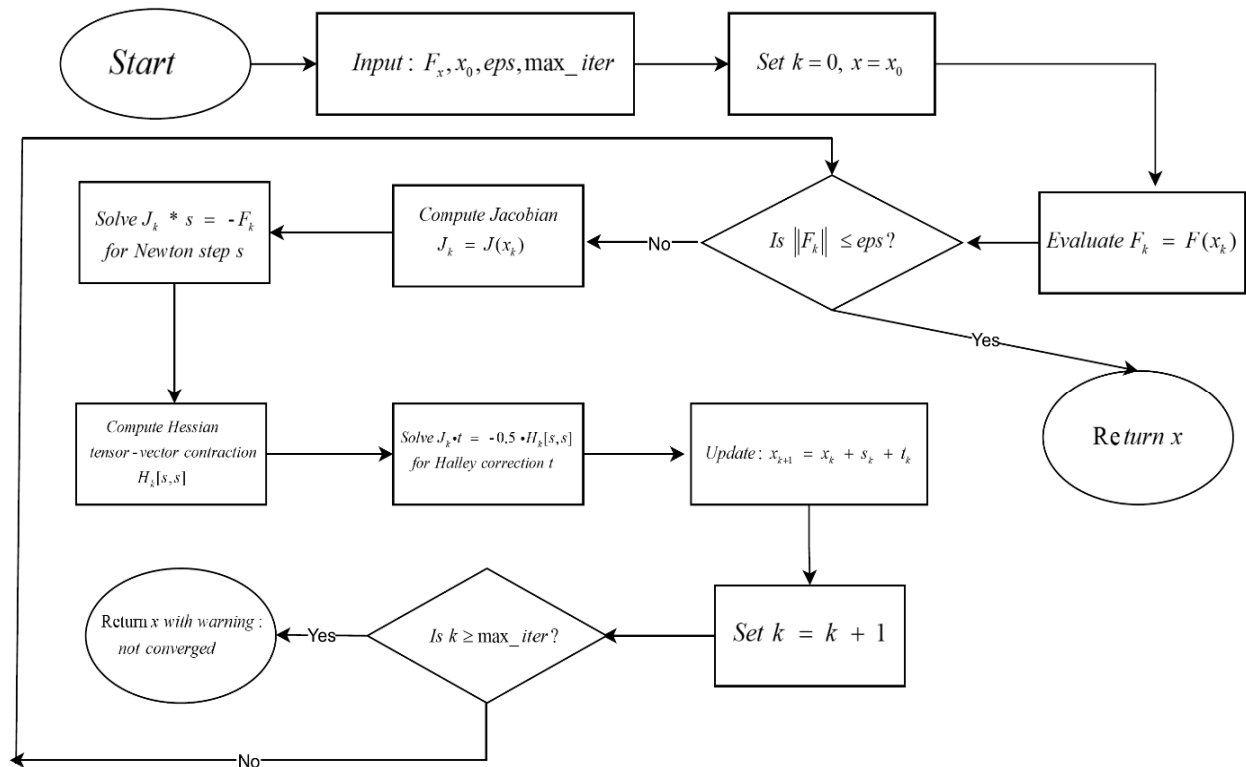


Figure 1. Flowchart of the multivariate Halley's method algorithm, illustrating the iterative procedure from initialization to convergence check.

3.2.1. Key Implementation Details

The implementation consists of three core components:

1. **Jacobian and Hessian Computation.** Mpmath's `mp.difffunction` computes partial derivatives to machine precision. For a system of n equations: $F: \mathbb{R}^n \rightarrow \mathbb{R}^n$, the Jacobian matrix $J(x) \in \mathbb{R}^{n \times n}$ and the Hessian tensor $H(x) \in \mathbb{R}^{n \times n \times n}$ are constructed element-wise:

from mpmath import mp, diff, matrix, lu_solve, norm

```
def jacobian(F, x, n):
```

```
    """Compute the n×n Jacobian matrix J(x)."""
```

```
    J = matrix(n, n)
```

```
    for i in range(n):
```

```
        for j in range(n):
```

```
            J[i, j] = diff(lambda h: F(x, idx=i, var=j, val=h), x[j])
```

```
    return J
```

```
def hessian_tensor(F, x, n):
```

```
    """Compute the n×n×n Hessian tensor H(x)."""
```

```
    H = [[[mp.mpf(0) for _ in range(n)] for _ in range(n)] for _ in range(n)]
```

```
    for i in range(n):
```

```
        for j in range(n):
```

```
            for k in range(n):
```

```
                H[i][j][k] = diff(lambda h: diff(
                    lambda g: F(x, idx=i, var=j, val=g), x[j] + h), x[k])
```

```
    return H
```

2. Halley Correction Step. Given the Newton step from solving $J(x) \cdot s = -F(x)$, the Halley correction t is obtained by solving: $J(x) \cdot t = -\frac{1}{2}H(x)[s, s]$, where $H(x)[s, s]$ denotes the vector

whose i -th component is $[H(x_k)[s, s]]_i = \sum_{j=1}^n \sum_{l=1}^n \frac{\partial^2 f_i}{\partial x_j \partial x_l} \bigg|_{x_k} s_j s_l = s^T H_i(x_k) s$:

The key python function is as follows:

```
def halley_correction(J, H, s, n):
```

```
    """Compute the Halley correction t from J·t = -1/2H[s,s]."""
```

```
    rhs = [mp.mpf(0) for _ in range(n)]
```

```
    for i in range(n):
```

```
        for j in range(n):
```

```
            for k in range(n):
```

```
                rhs[i] += H[i][j][k] * s[j] * s[k]
```

```
            rhs[i] *= -mp.mpf('0.5')
```

```
    return lu_solve(J, matrix(rhs))
```

3. Iteration Loop. The full algorithm proceeds as follows:

```
def multivariate_halley(F, x0, tol=mp.mpf('1e-30'), max_iter=50):
```

```
    """Multivariate Halley's method for root-finding."""
```

```
    x = [mp.mpf(xi) for xi in x0]
```

```
    n = len(x)
```

```
    for k in range(max_iter):
```

```
        Fk = F(x)
```

```
        if norm(Fk, 'inf') < tol:
```

```
            return x, k, True
```

```
        Jk = jacobian(F, x, n)
```

```
        Hk = hessian_tensor(F, x, n)
```

```
        s = lu_solve(Jk, matrix([-fi for fi in Fk]))
```

```
        t = halley_correction(Jk, Hk, s, n)
```

```
        x = [x[i] + s[i] + t[i] for i in range(n)]
```

```
    return x, max_iter, False
```

3.3. Numerical Results

Although we have implemented the multivariate Halley's method in high-precision Python for solving systems of nonlinear equations, it also supports the solution of single-variable equations.

3.3.1. Solving a Single -Variable Equation $x^x = 2026$ Using Halley's Method

The transcendental equation to be solved is: $f(x) = x^x - 2026 = 0$.

The solution parameters are as follows:

Initial guess $x_0 = 5.832$, convergence tolerance $\text{eps} = 1\text{e-}60$, $\text{calc_dps} = 150$,

Table 1. The complete iteration process

k	x_k	$ \Delta x_k _2$	Convergence Ratio $ F(x_k) / F(x_{k-1}) ^3$
0	5.832000000000000	—	—
1	4.83588835816835	9.9611×10^{-1}	—
2	4.83283877595638	3.0496×10^{-3}	$\approx 3.09 \times 10^{-3}$
3	4.83283877583218	1.2420×10^{-10}	≈ 4.38
4	4.83283877583218	8.3998×10^{-33}	≈ 4.40
5	4.83283877583218	2.5986×10^{-99}	≈ 4.39

For a third-order convergent method, the error propagation follows: $|e_{k+1}| \approx C \cdot |e_k|^3$.

Where $e_k = x_k - x^*$ is the error at iteration k , and C is the asymptotic error constant.

From the table, we observe the following:

From iteration 2 to 3: $\|e_3\| \approx 1.24 \times 10^{-10}$, $\|e_2\|^3 \approx (3.05 \times 10^{-3})^3 \approx 2.84 \times 10^{-8}$

$$\Rightarrow C \approx 1.24 \times 10^{-10} / 2.84 \times 10^{-8} \approx 4.38$$

From iteration 3 to 4: $\|e_4\| \approx 8.40 \times 10^{-33}$, $\|e_3\|^3 \approx (1.24 \times 10^{-10})^3 \approx 1.91 \times 10^{-30}$

$$\Rightarrow C \approx 8.40 \times 10^{-33} / 1.91 \times 10^{-30} \approx 4.40$$

From iteration 4 to 5: $\|e_5\| \approx 2.60 \times 10^{-99}$, $\|e_4\|^3 \approx (8.40 \times 10^{-33})^3 \approx 5.93 \times 10^{-97}$

$$\Rightarrow C \approx 2.60 \times 10^{-99} / 5.93 \times 10^{-97} \approx 4.39$$

The asymptotic error constant remains $C \approx 4.4$ remarkably consistent across three consecutive iterations, providing definitive evidence of third-order (cubic) convergence. This is a distinctive signature of Halley's method: in each iteration, the number of correct decimal digits approximately triples. Indeed, the error drops from 10^{-3} to 10^{-10} to 10^{-33} to 10^{-99} — a rate of convergence that Newton's method (quadratic, second-order) would require roughly twice as many iterations to achieve.

The equation converges to the root:

$$x^* = 4.83283877583217891234661546680494268753299245560053922643155$$

with a final residual increment of $\|\Delta x\|_2 \approx 2.6 \times 10^{-99}$, confirming the exceptional precision delivered by the third-order Halley's method.

3.3.2. Solving the Equation Using Halley's Method

The second example demonstrates the application of the multivariate Halley's method to a system of three nonlinear equations. The system is given by:

$$F(x_1, x_2, x_3) = \begin{cases} f_1 = x_1^2 + x_2^2 + x_1 x_2 - 121 = 0 \\ f_2 = x_2^2 + x_3^2 + x_2 x_3 - 169 = 0 \\ f_3 = x_3^2 + x_1^2 + x_3 x_1 - 400 = 0 \end{cases} \quad (22)$$

The solution parameters are as follows:

Initial guess point $x_0 = (x_1^0, x_2^0, x_3^0) = (2.8, 7.4, 5.3)$, convergence tolerance $eps = 10^{-30}$ (with internal precision of 200 decimal digits),

Table 2. the complete iteration process, showing the evolution of each variable and the norm of the correction step $|\Delta x|_2$

k	x_1	x_2	x_3	$ \Delta x _2$
0	2.8000000000000000	7.4000000000000000	5.3000000000000000	—
1	7.88759888529409	2.10925687032655	10.5107412195855	9.0015×10^0
2	10.4839319103758	0.899849929282384	12.5124988082983	3.4944×10^0
3	10.5304397031284	0.885583062055465	12.5345659271666	5.3418×10^{-2}
4	10.5304398584283	0.8855830501551104	12.5345659484591	1.5720×10^{-7}
5	10.5304398584283	0.8855830501551104	12.5345659484591	6.1246×10^{-24}
6	10.5304398584283	0.8855830501551104	12.5345659484591	7.3940×10^{-73}

The asymptotic error constant C remains on the order of 10^{-3} , which is consistent across iterations. This confirms that the method achieves third-order (cubic) convergence— each iteration roughly cubes the error, leading to an extremely rapid reduction in the residual.

The system converges to the root:

$$x^* = (x_1^*, x_2^*, x_3^*)^T = \begin{pmatrix} 10.5304398584283334979263801998 \\ 0.885583050155103754629156025296 \\ 12.5345659484591344835465826257 \end{pmatrix}, \text{ with a final residual increment of}$$

$\|\Delta x\|_2 \approx 7.39 \times 10^{-73}$, confirming the ultra-high precision delivered by the third-order Halley's method.

4. Conclusion

We clarify why Newton's method remains dominant and identify the practical conditions under which Halley-type methods become advantageous; We formulate the replacement criterion: high-precision requirement + expensive per-iteration cost $\Rightarrow$ Halley-type methods likely improve total efficiency; We present Halley solvers for both scalar nonlinear equations and multivariate nonlinear systems, and prove several lemmas and corollaries on convergence behavior; We provide high-precision implementations using Python Mpmath library, showing

that arbitrary-precision arithmetic reduces inversion error accumulation and improves stability.

References

- [1] Newton, I. (1687). *Philosophiæ naturalis principia mathematica*. Royal Society.
- [2] Raphson, J. (1690). *Analysis aequationum universalis*. A. & J. Churchill.
- [3] Halley, E. (1694). A new, exact, and easy method of finding the roots of any equations generally, without any previous reduction. *Philosophical Transactions of the Royal Society of London*, 18, 136–148. <https://doi.org/10.1098/rstl.1694.0027>
- [4] Thota, S., Gemechu, T., & Ayoade, A. A. (2024). On new hybrid root-finding algorithms for solving transcendental equations using exponential and Halley's methods. *Ural Mathematical Journal*, 9, 176–186. <https://doi.org/10.15826/umj.2024.2.012>
- [5] Ali, K. M., & Fathi, B. G. (2025). Enhancing nonlinear equation solutions through the combination of variant Newton's and Halley's methods. *Science Journal of University of Zakho*, 13.
- [6] Zhou, Z., & Cao, K. F. (2003). An effective numerical method of the word-lifting technique in one-dimensional multimodal maps. *Physics Letters A*, 310(1), 52–59. [https://doi.org/10.1016/S0375-9601\(03\)00329-1](https://doi.org/10.1016/S0375-9601(03)00329-1)