

六阶收敛的非线性方程组求根方法



课程: 数值分析

任课教师: 黎卫兵/张雨浓

| | | | |
|----|----------|----|------------------|
| 年级 | 2022 | 专业 | 计算机科学与技术 |
| 学号 | 22214373 | 姓名 | 林泽佳 |
| 班级 | 学硕 2 班 | 日期 | 2022 年 12 月 24 日 |

摘要

本文以张老师上课所介绍的构造 8 阶收敛的牛顿法, 和 PPT 第 63 页简要介绍 Steffenson 和 Halley 方法为基础和启发, 构造了 6 阶收敛的非线性方程组求根方法, 并使用前向差商代替二阶导数, 从而实现只需求一阶导数。本文使用泰勒展开证明了其收敛阶, 充分的数值试验对比了本方法与牛顿法、Halley 法和若干现有的 6 阶方法。

1 引言

本节主要对张老师上课所介绍的构造 8 阶收敛的牛顿法, 和 PPT 第 63 页简要介绍 Steffensen 和 Halley 方法做一个简要的回顾, 作为本文的基础。

1.1 8 阶收敛的牛顿法

使用三次牛顿法反复代入, 如公式1所示, 可以得到序列 $\{y_n\}_{n=0}^{\infty}$ 是 2 阶收敛, $\{z_n\}_{n=0}^{\infty}$ 是 4 阶收敛, 因此最终 $\{x_n\}_{n=0}^{\infty}$ 是 8 阶收敛的, 这是一种简单的提高迭代公式阶数的方法。

$$\begin{aligned}
 y_n &= x_n - \frac{f(x_n)}{f'(x_n)} \\
 z_n &= y_n - \frac{f(y_n)}{f'(y_n)} \\
 x_{n+1} &= z_n - \frac{f(z_n)}{f'(z_n)}
 \end{aligned} \tag{1}$$

1.2 Steffenson 方法

由于在零点附近 $\lim_{n \rightarrow \infty} f(x_n) = 0$, 因此可以使用前向差商可以将零点附近的导数近似为:

$$f'(x) \approx \frac{f(x_n + f(x_n))}{f(x_n)} \quad (2)$$

将公式2代入牛顿迭代公式, 可以得到:

$$x_{n+1} = x_n - \frac{f(x_n)^2}{f(x + f(x_n)) - f(x_n)} \quad (3)$$

这就是 PPT 第 63 页所提到的 Steffensen 公式, 它是二阶收敛的 [1], 同时避免了求导。

1.3 Halley 方法

PPT 第 63 页同样提到了三阶收敛的 Halley 方法和 Chebyshev 方法, 查阅文献 [2] 后发现, 它们属于同一族方法, 具有如下形式:

$$x_{n+1} = x_n - \left(1 + \frac{1}{2} \frac{L_f(x_n)}{1 - \beta L_f(x_n)}\right) \frac{f(x_n)}{f'(x_n)}, \quad (4)$$

其中

$$L_f(x_n) = \frac{1}{2} \frac{f''(x_n)f(x_n)}{[f'(x_n)]^2} \quad (5)$$

特别的, 当 $\beta = 0$ 时为 Chebyshev 方法, $\beta = \frac{1}{2}$ 时为 Halley 方法, $\beta = 1$ 时为超 Halley 方法。不少研究对这族方法进行改进, 但是它们都具有较为复杂的迭代格式, 如 [3] 提出的一种无需二阶导的三阶收敛的方法:

$$\begin{cases} y_n &= x_n - \theta \frac{f(x_n)}{f'(x_n)}, \\ x_{n+1} &= x_n - \frac{\theta^2 f(x_n)}{(\theta^2 - \theta + 1)f(x_n) - f(y_n)} \frac{f(x_n)}{f'(x_n)}, \end{cases} \quad (\theta \in \mathbb{R}, \theta \neq 0), \quad (6)$$

又如 [4] 提出的另一种三阶收敛的方法:

$$\begin{cases} w_n &= x_n - \frac{f(x_n)}{f'(x_n)}, \\ K_\alpha(x_n) &= \frac{2f(x_n)f(w_n)(1 + \alpha f'^2(x_n))}{f^2(x_n) + \alpha f'^2(x_n)[f(w_n) - f(x_n)]^2}, \quad (\alpha \in \mathbb{R}), \\ x_{n+1} &= x_n - \left(1 + \frac{1}{2} \frac{K_\alpha(x_n)}{1 - \beta K_\alpha(x_n)}\right) \frac{f(x_n)}{f'(x_n)}, \quad (\beta \in \mathbb{R}), \end{cases} \quad (7)$$

这些方法与同为三阶收敛的其它方法相比, 或多或少能减少一些迭代次数, 但代价是较为复杂的迭代格式。

1.4 六阶收敛的方法

在三阶收敛方法的基础上, 再使用一次牛顿迭代或其它二阶迭代法, 即可实现六阶收敛的方法。如最早由 [5] 提出的六阶收敛方法:

$$\begin{cases} w_n &= x_n - \frac{f(x_n)}{f'(x_n)}, \\ z_n &= w_n - \frac{f(w_n)}{f'(x_n)} \cdot \frac{f(x_n) - \frac{1}{2}f(w_n)}{f(x_n) - \frac{5}{2}f(w_n)}, \\ x_{n+1} &= z_n - \frac{f(z_n)}{f'(x_n)} \cdot \frac{f(x_n) - f(w_n)}{f(x_n) - 3f(w_n)} \end{cases} \quad (8)$$

又如 [6] 中对 Ostrowski 方法 [7] 改进提出的六阶收敛的方法:

$$\begin{cases} y_n &= x_n - \frac{f(x_n)}{f'(x_n)}, \\ z_n &= y_n - \frac{y_n - x_n}{2f(y_n) - f(x_n)} f(y_n), \\ x_{n+1} &= z_n - \frac{y_n - x_n}{2f(y_n) - f(x_n)} f(z_n), \end{cases} \quad (9)$$

又如 [8] 中提出的用前向差商代替导数的六阶收敛的方法:

$$\begin{cases} w_n &= x_n + f(x_n), \\ y_n &= x_n - \frac{f(x_n)}{f[x_n, w_n]}, \\ z_n &= x_n - \frac{f(x_n)}{f[x_n, w_n]} \left(1 + \frac{f(y_n)}{f(x_n)} \left(1 + 2 \frac{f(y_n)}{f(x_n)} \right) \right), \\ x_{n+1} &= z_n - \frac{f(z_n)}{f[y_n, z_n]} \left(1 - \frac{1 + f[x_n, w_n]f(z_n)}{f[x_n, w_n]f(w_n)} \right) \end{cases} \quad (10)$$

除此之外, [9][10][11] 等均提出了不同形式的六阶收敛的方法, 虽然它们的计算效率较高, 但均具有较为复杂的迭代格式。本文希望能建立一种较为简单的迭代格式对 Halley 方法进行改造, 在避免求二阶导数的基础上实现六阶收敛的方法。诚信声明, 在我所了解到的文献里没有与本文方法相同的, 虽然推导和证明过程均有参考它们, 也有相应的引用, 但皆为原创和独立完成; 如有雷同, 是我查找资料不周全, 没有恶意抄袭的意思。

2 方法设计

2.1 方法推导

在三阶的 Halley 方法的基础上再进行一次牛顿法迭代得到的序列, 显然是六阶收敛的:

$$\begin{cases} z_n &= x_n - \frac{2f(x_n)f'(x_n)}{2[f'(x_n)]^2 - f(x_n)f''(x_n)}, \\ x_{n+1} &= x_n - \frac{f(z_n)}{f'(z_n)} \end{cases} \quad (11)$$

参考 [12] 中的方法, 希望消去 $\{z_n\}_{n=0}^{\infty}$ 中的二阶导数, 如果记 $y_n = x_n - \frac{f(x)}{f'(x)}$, 则有 $\lim_{n \rightarrow \infty} \frac{y_n}{x_n} = 1$, 因此二阶导数可以使用一阶导计算前向差商近似为:

$$\begin{aligned} f''(x_n) &= \lim_{n \rightarrow \infty} \frac{f'(y_n) - f'(x_n)}{y_n - x_n} \\ &\approx \frac{f'(y_n) - f'(x_n)}{y_n - x_n}, \end{aligned} \quad (12)$$

将公式12代入公式11, 使用 sympy¹ (代码见附录 1) 化简得:

$$\begin{cases} y_n &= x_n - \frac{f(x)}{f'(x)} \\ z_n &= x_n - \frac{2f(x_n)}{f'(x_n) + f'(y_n)} \\ x_{n+1} &= z_n - \frac{f(z_n)}{f'(z_n)} \end{cases} \quad (13)$$

公式13即为本文所提出的方法, 由于它使用了差商代替二阶导数, 因此需要对收敛阶进行证明。

2.2 收敛阶证明

我参考了 [3] 和 [12] 使用的技巧对公式13的收敛阶进行证明。由于涉及到 **6** 阶泰勒展开的计算较为复杂, 因此我使用了 sympy¹ 进行符号计算和化简, 代码在附录 2 和 `code/convergence.py` 中, 本部分所推导的结果皆由该代码产生, 其正确性保证了本文推导结果的正确。由于篇幅有限, 部分泰勒展开用省略号表示。

假设函数 $f(x)$ 的单根是 α 且 $f'(\alpha) \neq 0$, 记截断误差 $e_n = \alpha - x_n$, 对 $f(x_n)$ 在 α 处进行泰勒展开得:

$$\begin{aligned} f(x_n) &= f(\alpha + e_n) \\ &= f(\alpha) + f'(\alpha)e_n + \frac{1}{2}f''(\alpha)e_n^2 + \frac{1}{3!}f'''(\alpha)e_n^3 + \cdots + O(e_n^7) \\ &= f'(\alpha) \left[e_n + \frac{f''(\alpha)e_n^2}{2f'(\alpha)} + \frac{f'''(\alpha)e_n^3}{3!f'(\alpha)} + \cdots + O(e_n^7) \right], \end{aligned} \quad (14)$$

注意到 $f(\alpha) = 0$, 所以公式中将它消去了, 记 $c_k = \frac{f^{(k)}(\alpha)}{f'(\alpha)k!}$, 则有

$$\begin{aligned} f(x_n) &= f'(\alpha) [c_1e_n + c_2e_n^2 + c_3e_n^3 + c_4e_n^4 + c_5e_n^5 + c_6e_n^6 + O(e_n^7)] \\ &= f'(\alpha) \left[\sum_{k=1}^6 c_k e_n^k + O(e_n^7) \right] \end{aligned} \quad (15)$$

¹sympy 是 Python 的符号计算库, <https://www.sympy.org/en/index.html>

同理对 $f'(x_n)$ 在 α 处进行泰勒展开得:

$$\begin{aligned}
 f'(x_n) &= f'(\alpha + e_n) \\
 &= f'(\alpha) + f''(\alpha)e_n + \frac{1}{2}f'''(\alpha)e_n^2 + \cdots + O(e_n^6) \\
 &= f'(\alpha)[1 + 2c_2e_n + 3c_3e_n^2 + 4c_4e_n^3 + 5c_5e_n^4 + 6c_6e_n^5 + O(e_n^6)] \\
 &= f'(\alpha) \left[1 + \sum_{k=2}^6 kc_k e_n^{k-1} + O(e_n^6) \right]
 \end{aligned} \tag{16}$$

希望得到 y_n 的泰勒展开形式, 因此首先需要计算:

$$\frac{f(x_n)}{f'(x_n)} = \frac{\sum_{k=1}^6 c_k e_n^k + O(e_n^7)}{1 + \sum_{k=2}^6 kc_k e_n^{k-1} + O(e_n^6)}, \tag{17}$$

记 $\theta = 1 + \sum_{k=2}^6 kc_k e_n^{k-1} + O(e_n^6)$, 显然 $\lim_{n \rightarrow \infty} \theta = 0$, 因此在 $\theta = 0$ 处对分母进行泰勒展开有 (代码第 61 行):

$$\begin{aligned}
 \frac{1}{1+\theta} &= 1 - \theta - 2c_2e + \theta^2 - \theta^3 + \theta^4 - \theta^5 + O(\theta^6) \\
 &= 1 + e_n^2 \cdot (4c_2^2 - 3c_3) + e_n^3 (-8c_2^3 + 12c_2c_3 - 4c_4) \\
 &\quad + e_n^4 \cdot (16c_2^4 - 36c_2^2c_3 + 16c_2c_4 + 9c_3^2 - 5c_5) \\
 &\quad + e_n^5 (-32c_2^5 + 96c_2^3c_3 - 48c_2^2c_4 - 54c_2c_3^2 + 20c_2c_5 + 24c_3c_4 - 6c_6) + O(e_n^6),
 \end{aligned} \tag{18}$$

将公式18代入公式17并化简有 (代码第 65 行):

$$\begin{aligned}
 \frac{f(x_n)}{f'(x_n)} &= e - c_2e_n^2 + e_n^3 \cdot (2c_2^2 - 2c_3) + e_n^4 \cdot (-4c_2^3 + 7c_2c_3 - 3c_4) \\
 &\quad + e_n^5 \cdot (8c_2^4 - 20c_2^2c_3 + 10c_2c_4 + 6c_3^2 - 4c_5) \\
 &\quad + e_n^6 \cdot (-16c_2^5 + 52c_2^3c_3 - 28c_2^2c_4 - 33c_2c_3^2 + 13c_2c_5 + 17c_3c_4 - 5c_6) + O(e_n^7),
 \end{aligned} \tag{19}$$

因此求得 y_n 的泰勒展开 (代码第 70 行):

$$\begin{aligned}
 y_n &= x_n - \frac{f(x_n)}{f'(x_n)} \\
 &= \alpha + c_2e_n^2 + e_n^3 \cdot (-2c_2^2 + 2c_3) + e_n^4 \cdot (4c_2^3 - 7c_2c_3 + 3c_4) \\
 &\quad + e_n^5 \cdot (-8c_2^4 + 20c_2^2c_3 - 10c_2c_4 - 6c_3^2 + 4c_5) \\
 &\quad + e_n^6 \cdot (16c_2^5 - 52c_2^3c_3 + 28c_2^2c_4 + 33c_2c_3^2 - 13c_2c_5 - 17c_3c_4 + 5c_6) + O(e_n^7),
 \end{aligned} \tag{20}$$

公式20同时也证明了牛顿法是二阶收敛的, 截断误差为 $\frac{f''(\alpha)}{2f'(\alpha)}e_n^2 + O(e_n^3)$ 。继续对 $f'(y_n)$ 在 α

处做泰勒展开有（代码第 77 行）：

$$\begin{aligned} f'(y_n) &= f'(\alpha) + [c_2 e_n^2 + (2c_2^2 - 2c_3)e_n^3 + O(e_n^4)]f''(\alpha) + \cdots + O(e_n^6) \\ &= 1 + 2c_2^2 e_n^2 + e_n^3 (-4c_2^3 + 4c_2 c_3) + e_n^4 (8c_2^4 - 11c_2^2 c_3 + 6c_2 c_4) \\ &\quad + e_n^5 (-16c_2^5 + 28c_2^3 c_3 - 20c_2^2 c_4 + 8c_2 c_5) + O(e_n^6), \end{aligned} \quad (21)$$

将公式16与公式21相加得到 z_n 的分母部分的泰勒展开：

$$f'(x_n) + f'(y_n) = 2f'(\alpha) \left[1 + c_2 e_n + \left(c_2^2 + \frac{3}{2} c_3 \right) e_n^2 + \cdots + O(e_n^6) \right], \quad (22)$$

因此将公式22与公式14相除得到 z_n 的泰勒展开：

$$x_n - \frac{2f(x_n)}{f'(x_n) + f'(y_n)} = \alpha + e_n - \frac{2 \left[\sum_{k=1}^6 c_k e_n^k + O(e_n^7) \right]}{1 + c_2 e_n + \left(c_2^2 + \frac{3}{2} c_3 \right) e_n^2 + \cdots + O(e_n^6)}, \quad (23)$$

使用与公式18相似的技巧，对分母部分进行泰勒展开得（代码第 82 行）：

$$\begin{aligned} x_n - \frac{2f(x_n)}{f'(x_n) + f'(y_n)} &= \alpha + e_n^3 \cdot \left(c_2^2 + \frac{c_3}{2} \right) + e_n^4 \cdot \left(-3c_2^3 + \frac{3c_2 c_3}{2} + c_4 \right) \\ &\quad + e_n^5 \cdot \left(6c_2^4 - 9c_2^2 c_3 + 2c_2 c_4 - \frac{3c_3^2}{4} + \frac{3c_5}{2} \right) \\ &\quad + e_n^6 \cdot \left(-9c_2^5 + 25c_2^3 c_3 - 15c_2^2 c_4 - \frac{5c_2 c_3^2}{2} + \frac{5c_2 c_5}{2} - \frac{5c_3 c_4}{2} + 2c_6 \right) \\ &\quad + O(e_n^7), \end{aligned} \quad (24)$$

公式24同时也证明了序列 $\{z_n\}_{n=0}^\infty$ 是三阶收敛的，即使用一阶导数的前向差商来计算二阶导数的 Halley 方法是仍是三阶收敛的。继续计算 $f(z_n)$ 和 $f'(z_n)$ 的泰勒展开，使用与公式14-20相同的方法，可以得到 x_{n+1} 的泰勒展开（代码第 88 行）：

$$x_{n+1} = \alpha + e^6 \cdot \left(c_2^5 + c_2^3 c_3 + \frac{c_2 c_3^2}{4} \right) + O(e^7), \quad (25)$$

因此证明了本方法是六阶收敛的，截断误差为 $e^6 \cdot \left(c_2^5 + c_2^3 c_3 + \frac{c_2 c_3^2}{4} \right) + O(e^7)$

2.3 一些有趣的发现

由于使用 sympy 可以很方便地进行泰勒展开来对收敛阶进行推导，我又尝试了一下修改迭代格式，使用与 $\{z_n\}_{n=0}^\infty$ 相似的格式来代替最后一步的牛顿法：

$$\begin{cases} y_n &= x_n - \frac{f(x)}{f'(x)} \\ z_n &= x_n - \frac{2f(x_n)}{f'(x_n) + f'(y_n)}, \\ x_{n+1} &= z_n - \frac{2f(z_n)}{f'(y_n) + f'(z_n)} \end{cases} \quad (26)$$

附录 2 的代码第 94 行展示了收敛性证明的计算过程, 使用与公式21-25类似的方法, 可以得到 x_{n+1} 的泰勒展开:

$$x_{n+1} = e^5 \cdot \left(c_2^4 + \frac{c_2^2 c_3}{2} \right) + e^6 \cdot \left(-5c_2^5 + \frac{5c_2^3 c_3}{2} + c_2^2 c_4 + c_2 c_3^2 \right) + \alpha + O(e^7), \quad (27)$$

因此公式26只有 5 阶收敛, 但它的计算量比牛顿法还多 (同样需要计算 $f(z_n)$ 和 $f'(z_n)$, 且需要额外计算一次加法和乘法), 进一步让我感受到了牛顿法无可比拟的优势。

2.4 计算效率讨论

在 [13] 中定义了迭代法的效率指数 (Efficiency index) 为:

$$EI = q^{1/\omega}, \quad (28)$$

其中 q 是收敛阶, ω 是每次迭代需要计算的函数的次数 (包括其导数)。例如牛顿法二阶收敛, 需要计算一次 $f(x_n)$ 和一次 $f'(x_n)$, 因此效率指数为 $EI = 2^{1/2}$ 。

表1列举了本方法和部分方法的效率指数, 和方法中额外引入的乘除法次数, 这些方法也将会在实验中进行验证。可以看到, 本方法具有相对简单的迭代格式, 尽管效率指数略低于现有方法, 但引入额外的乘除法次数较少, 且减少了一次计算 $f(x)$, 仍然具有一定的实用价值。

表 1: 一些迭代法的计算效率

| 方法 | 收敛阶 | 计算次数 | | | 效率指数 | 额外乘除法次数 |
|----------------|-----|--------|---------|----------|-------------------------|---------|
| | | $f(x)$ | $f'(x)$ | $f''(x)$ | | |
| 牛顿法 | 2 | 1 | 1 | 0 | $2^{1/2} \approx 1.414$ | 1 |
| Halley 法 (公式4) | 3 | 1 | 1 | 1 | $3^{1/3} \approx 1.442$ | 4 |
| NM[5] (公式8) | 6 | 3 | 1 | 0 | $6^{1/4} \approx 1.567$ | 7 |
| GM[6] (公式9) | 6 | 3 | 1 | 0 | $6^{1/4} \approx 1.567$ | 5 |
| 本方法 (公式13) | 6 | 2 | 3 | 0 | $6^{1/5} \approx 1.431$ | 3 |

3 实验

3.1 测试方法和指标

测试代码在附录 3 和 `code/compute.py` 中, 本文复现了表1中的 5 种方法, 表2列举了本文用于测试的函数和零点 (大部分来源于 [3] 和 [9]), 图1展示了这些函数在零点附近的图像。使用 IEEE 64 位浮点数 (具有约 15 位十进制的规格化有效数字) 进行计算, 迭代终止条件是 $|x_n - x_{n-1}| + |f(x_n)| < 10^{-12}$, 评价指标为迭代次数。

表 2: 本文用于测试的函数和零点

| 记号 | 函数 | 零点 |
|-------|---------------------------------------|----------------------------------|
| f_1 | $x^3 + 4x^2 - 15$ | $x^* \approx 1.6319808055660636$ |
| f_2 | $x^2 - e^x - 3x + 2$ | $x^* \approx 0.2575302854398608$ |
| f_3 | $xe^{x^2} - \sin^2(x) + 3\cos(x) + 5$ | $x^* \approx -1.207647827130919$ |
| f_4 | $\sin^2(x) - x^2 + 1$ | $x^* \approx 1.4044916482153411$ |
| f_5 | $\ln(x^2 + 7x + 14) - x - 2$ | $x^* \approx 1.1525907367571583$ |
| f_6 | $\frac{(x-4)(x+1)^4}{e^x}$ | $x^* = -1$ (四重根) |
| f_7 | $e^{x^2+11x-12} - 1$ | $x^* = 1$ |
| f_8 | $(x-1)^2 \arctan(e^{x+3} - 1)$ | $x^* = 1$ (二重根) |

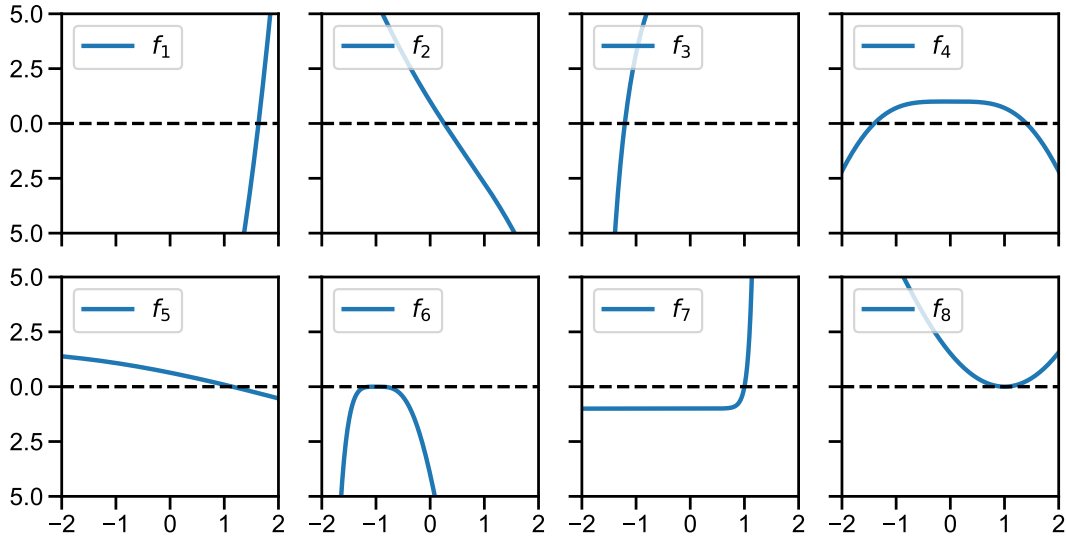


图 1: 测试函数在零点附近的图像

3.2 实验结果

表3展示表1的方法对表2的函数进行测试的实验结果, 对于每个函数都使用了 2 个分布在根两端的初值进行实验。迭代次数中的 (U) 表示下溢 (接近 0 而超过了精度范围, 导致变成 0 了), (F) 表示上溢 (绝对值太大), (N) 表示迭代发散。注意 IEEE 64 位浮点数具有大约 15 位的规格化有效数字, 16 位的非规格化有效数字。

表 3: 迭代次数和误差

| 函数 | 初值 | 方法 | 迭代 次数 | 与 x^* 误差 | 函数 | 初值 | 方法 | 迭代 次数 | 与 x^* 误差 |
|-------|----|--------|----------|------------------------|-------|------|--------|----------|------------------------|
| f_1 | 1 | 牛顿 | 6 | $-2.22 \cdot 10^{-16}$ | f_5 | 1 | 牛顿 | 4 | $2.22 \cdot 10^{-16}$ |
| | | Halley | 4 | $-2.22 \cdot 10^{-16}$ | | | Halley | 3 | $-2.22 \cdot 10^{-16}$ |
| | | NM | 6 | $-4.44 \cdot 10^{-16}$ | | | NM | 4 | $< 10^{-16}$ |
| | | GM | 3 | $-2.22 \cdot 10^{-16}$ | | | GM | 3(U) | - |
| | | 本文 | 3 | $< 10^{-16}$ | | | 本文 | 2 | $-2.22 \cdot 10^{-16}$ |
| | 2 | 牛顿 | 5 | $< 10^{-16}$ | | 2 | 牛顿 | 5 | $-2.22 \cdot 10^{-16}$ |
| | | Halley | 4 | $< 10^{-16}$ | | | Halley | 4 | $< 10^{-16}$ |
| | | NM | 5 | $-4.44 \cdot 10^{-16}$ | | | NM | 5(U) | - |
| | | GM | 3 | $-2.22 \cdot 10^{-16}$ | | | GM | 3(U) | - |
| | | 本文 | 3 | $-2.22 \cdot 10^{-16}$ | | | 本文 | 3 | $-4.44 \cdot 10^{-16}$ |
| f_2 | 0 | 牛顿 | 5 | $< 10^{-16}$ | f_6 | -1.5 | 牛顿 | 91 | $-2.6 \cdot 10^{-12}$ |
| | | Halley | 4 | $< 10^{-16}$ | | | Halley | 53 | $-1.01 \cdot 10^{-12}$ |
| | | NM | 4 | $< 10^{-16}$ | | | NM | 22 | $-1.71 \cdot 10^{-13}$ |
| | | GM | 3(U) | - | | | GM | 37 | $-8.79 \cdot 10^{-13}$ |
| | | 本文 | 3 | $< 10^{-16}$ | | | 本文 | 38 | $-7.74 \cdot 10^{-13}$ |
| | 1 | 牛顿 | 5 | $< 10^{-16}$ | | -0.5 | 牛顿 | 90 | $2.32 \cdot 10^{-12}$ |
| | | Halley | 4 | $< 10^{-16}$ | | | Halley | 52 | $1.25 \cdot 10^{-12}$ |
| | | NM | 5 | $-2.22 \cdot 10^{-16}$ | | | NM | 22 | $1.78 \cdot 10^{-13}$ |
| | | GM | 3(U) | - | | | GM | 37 | $6.25 \cdot 10^{-13}$ |
| | | 本文 | 3 | $< 10^{-16}$ | | | 本文 | 38 | $5.08 \cdot 10^{-13}$ |
| f_3 | -2 | 牛顿 | 9 | $2.22 \cdot 10^{-16}$ | f_7 | 0.5 | 牛顿 | 2(F) | - |
| | | Halley | 5 | $2.22 \cdot 10^{-16}$ | | | Halley | 7 | $< 10^{-16}$ |
| | | NM | 2(F) | - | | | NM | 2(F) | - |
| | | GM | 4 | $< 10^{-16}$ | | | GM | 2(F) | - |
| | | 本文 | 4 | $2.22 \cdot 10^{-16}$ | | | 本文 | 2(F) | - |
| | -1 | 牛顿 | 6 | $2.22 \cdot 10^{-16}$ | | 1.5 | 牛顿 | 12 | $-1.11 \cdot 10^{-16}$ |
| | | Halley | 4 | $2.22 \cdot 10^{-16}$ | | | Halley | 7 | $-1.11 \cdot 10^{-16}$ |
| | | NM | 7 | $-1.02 \cdot 10^{-14}$ | | | NM | 2(F) | - |

续下页

表 3 迭代次数和误差 (接上页)

| 函数 | 初值 | 方法 | 迭代 次数 | 与 x^* 误差 | 函数 | 初值 | 方法 | 迭代 次数 | 与 x^* 误差 |
|-------|----|--------|----------|-----------------------|-------|-----|--------|----------|------------------------|
| | | GM | 3 | $< 10^{-16}$ | | | GM | 5 | $< 10^{-16}$ |
| | | 本文 | 3 | $2.22 \cdot 10^{-16}$ | | | 本文 | 6 | $< 10^{-16}$ |
| f_4 | 1 | 牛顿 | 6 | $< 10^{-16}$ | f_8 | 0.5 | 牛顿 | 39 | $-9.01 \cdot 10^{-13}$ |
| | | Halley | 4 | $2.22 \cdot 10^{-16}$ | | | Halley | 26 | $-1.96 \cdot 10^{-13}$ |
| | | NM | 7 | $2.22 \cdot 10^{-16}$ | | | NM | 27(N) | -4.0 |
| | | GM | 3 | $< 10^{-16}$ | | | GM | 17 | $-2.18 \cdot 10^{-13}$ |
| | | 本文 | 3 | $< 10^{-16}$ | | | 本文 | 16 | $-1.75 \cdot 10^{-13}$ |
| | 2 | 牛顿 | 6 | $< 10^{-16}$ | | 1.5 | 牛顿 | 39 | $9.14 \cdot 10^{-13}$ |
| | | Halley | 4 | $2.22 \cdot 10^{-16}$ | | | Halley | 26 | $1.97 \cdot 10^{-13}$ |
| | | NM | 7 | $2.22 \cdot 10^{-16}$ | | | NM | 14(N) | -4.0 |
| | | GM | 3 | $< 10^{-16}$ | | | GM | 17 | $2.19 \cdot 10^{-13}$ |
| | | 本文 | 3 | $< 10^{-16}$ | | | 本文 | 16 | $1.78 \cdot 10^{-13}$ |

3.3 结果分析

表4总结了这 5 种方法的成功次数和失败原因。Halley 法在所有测试中均成功得到结果, 这可能是因为使用了二阶导数, 数值稳定性非常好。牛顿法和本文方法除了有一次上溢以外也都成功了。GM 法出现了 4 次下溢错误, 这是因为它的公式中包含“小数减小数”、“小数除小数”等计算, 因此非常容易产生下溢, 数值稳定性不好。NM 法中出现了 3 次上溢和 2 次发散, 这可能是因为每代额外使用 7 次乘除法, 加速了误差积累。本文方法作为六阶收敛的方法, 数值稳定性与牛顿法相当, 高于另外两种方法, 且计算效率 (公式28) 高于牛顿法, 具有良好的实用价值。

对前 5 个函数 $f_1 \sim f_5$, 各方法基本有比较好的效果, 并且迭代次数趋势也与收敛阶相匹配。对于第 6 个函数 f_6 , 其零点是 4 重根, 因此牛顿法收敛速度非常慢, NM 法最快, GM 法和本文方法相当。对于第 7 个函数 f_7 , 选取初值 $x_0 = 0.5$ 时, 多种方法均产生了上溢, 观察函数图像 (图1) 发现它在 $(-2, 1]$ 上非常平缓, 因此导数很小, 基于牛顿法的一系列方法在计算 $\frac{f(x_n)}{f'(x_n)}$ 时容易上溢。但是当选取另一个初值 $x_0 = 1.5$ 时, 由于导数很大, 因此没有这个问题。对于第 8 个函数 f_8 , NM 很意外的发散了, 因此及时终止了迭代, 由于时间有限我没有仔细探究其原因。由于零点是 2 重根的原因, 牛顿法同样收敛的很慢, 但本文方法拥有最快的收敛速度, GM 法其次。

表 4: 各方法成功次数和失败原因

| 类型 | 牛顿 | Halley (公式4) | NM[5] (公式8) | GM[6] (公式9) | 本文 (公式13) |
|----|----|-----------------|----------------|----------------|--------------|
| 成功 | 15 | 16 | 10 | 12 | 15 |
| 下溢 | 0 | 0 | 1 | 3 | 0 |
| 上溢 | 1 | 0 | 3 | 1 | 1 |
| 发散 | 0 | 0 | 2 | 0 | 0 |

3.4 对重根的讨论

计算收敛阶 (Computational order of convergence) [14] 是数值方法在实际计算过程中的收敛阶, 其定义如下:

$$COC = \frac{\ln \left(\left| \frac{x_{n+1} - x_n}{x_n - x_{n-1}} \right| \right)}{\ln \left(\left| \frac{x_n - x_{n-1}}{x_{n-1} - x_{n-2}} \right| \right)} \quad (29)$$

表5所示本文列举了 $f_6 \sim f_8$ 的计算收敛阶。对于单根的函数 f_7 , 各方法的计算收敛阶基本符合其理论的收敛阶; 对于重根函数 f_6 和 f_8 , 所有方法只有线性收敛。

表 5: 计算重根时的计算收敛阶

| 函数 | 重复度 | 初值 | 牛顿 | Halley | NM | GM | 本文 |
|-------|------|------|-------|--------|-------|-------|-------|
| f_7 | 单根 | 1.5 | 2.002 | 3.066 | 发散 | 5.990 | 6.053 |
| f_6 | 4 重根 | -1.5 | 0.991 | 0.998 | 1.000 | 0.985 | 0.993 |
| | | -0.5 | 0.996 | 0.996 | 0.999 | 1.001 | 1.000 |
| f_8 | 2 重根 | 0.5 | 1.000 | 0.998 | 发散 | 1.000 | 1.000 |
| | | 1.5 | 0.977 | 1.021 | 发散 | 1.000 | 0.999 |

3.5 迭代过程展示

作为补充材料, 表6和表7对迭代过程进行了展示。

4 总结

表 6: f_4 选取初值为 2 的迭代过程

| n | 牛顿 | Halley | NM | GM | 本文 |
|-----|-------------------|-------------------|-------------------|-------------------|-------------------|
| 0 | 2 | 2 | 2 | 2 | 2 |
| 1 | 1.543143068960336 | 1.456885216221384 | 1.120415325387814 | 1.407237330215151 | 1.405535212978439 |
| 2 | 1.417094222312942 | 1.404562548049610 | 1.309731172778497 | 1.404491648215341 | 1.404491648215341 |
| 3 | 1.404614018363034 | 1.404491648215529 | 1.397104012247177 | 1.404491648215341 | 1.404491648215341 |
| 4 | 1.404491659946959 | 1.404491648215341 | 1.404448806412075 | - | - |
| 5 | 1.404491648215341 | - | 1.404491646777146 | - | - |
| 6 | 1.404491648215341 | - | 1.404491648215341 | - | - |

表 7: f_7 选取初值为 1.5 的迭代过程

| | 牛顿 | Halley | NM | GM | 本文 |
|----|--------------------|-------------------|------------------------------|-------------------|-------------------|
| 0 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 |
| 1 | 1.428655062830056 | 1.356011165775886 | 0.666554094953474 | 1.302765996348761 | 1.323425736359648 |
| 2 | 1.356719234358469 | 1.211129011680508 | $-1.275596982 \cdot 10^{52}$ | 1.109913322973212 | 1.147701833153800 |
| 3 | 1.284419811223503 | 1.078073976922075 | - | 1.002996956434495 | 1.017028589466088 |
| 4 | 1.212406308451123 | 1.006179477275287 | - | 1.000000000003765 | 1.000000403894250 |
| 5 | 1.142418159478025 | 1.000003327216270 | - | 1.000000000000000 | 1.000000000000000 |
| 6 | 1.078725914448773 | 1.000000000000000 | - | - | 1.000000000000000 |
| 7 | 1.029866713280862 | 1.000000000000000 | - | - | - |
| 8 | 1.005182160439837 | - | - | - | - |
| 9 | 1.000172764038992 | - | - | - | - |
| 10 | 1.000000196158916 | - | - | - | - |
| 11 | 1.0000000000000253 | - | - | - | - |
| 12 | 1.000000000000000 | - | - | - | - |

参考文献

- [1] Pankaj Jain. Steffensen type methods for solving non-linear equations. *Applied Mathematics and Computation*, 194(2):527–533, 2007.
- [2] Jose Manuel Gutierrez and Miguel A Hernández. A family of chebyshev-halley type methods in banach spaces. *Bulletin of the Australian Mathematical Society*, 55(1):113–130, 1997.
- [3] Jisheng Kou, Yitian Li, and Xiuhua Wang. Modified halley’s method free from second derivative. *Applied Mathematics and Computation*, 183(1):704–708, 2006.
- [4] Changbum Chun. Some second-derivative-free variants of chebyshev–halley methods. *Applied Mathematics and Computation*, 191(2):410–414, 2007.
- [5] Beny Neta. A sixth-order family of methods for nonlinear equations. *Int. J. Comput. Math*, 7(1997):157–161, 1979.
- [6] Miquel Grau and José Luis Díaz-Barrero. An improvement to ostrowski root-finding method. *Applied Mathematics and Computation*, 173(1):450–456, 2006.
- [7] AS Householder. Solution of equations and systems of equations (am ostrowski). *SIAM Review*, 9(3):608, 1967.
- [8] F Soleymani and V Hosseinabadi. New third-and sixth-order derivative-free techniques for nonlinear equations. *Journal of Mathematics Research*, 3(2):107, 2011.
- [9] Changbum Chun and Beny Neta. A new sixth-order scheme for nonlinear equations. *Applied mathematics letters*, 25(2):185–189, 2012.
- [10] Obadah Said Solaiman and Ishak Hashim. Two new efficient sixth order iterative methods for solving nonlinear equations. *Journal of King Saud University-Science*, 31(4):701–705, 2019.
- [11] Mona Narang, Saurabh Bhatia, and Vinay Kanwar. New two-parameter chebyshev–halley-like family of fourth and sixth-order methods for systems of nonlinear equations. *Applied Mathematics and Computation*, 275:394–403, 2016.
- [12] Tahereh Eftekhari. A new sixth-order steffensen-type iterative method for solving nonlinear equations. *International Journal of Analysis*, 2014, 2014.
- [13] A.M. Ostrowski. *Solutions of equations and system of equations*. Academic Press, 1960.
- [14] Alicia Cordero and Juan R Torregrosa. Variants of newton’s method using fifth-order quadrature formulas. *Applied Mathematics and Computation*, 190(1):686–698, 2007.

附录

附录 1

对化简 Halley 方法二阶导数的代码

```
1  import sympy as sp
2
3  fx = sp.symbols('fx')      #  $f(x)$ 
4  dx = sp.symbols('dx')      #  $f'(x)$ 
5  x = sp.symbols('x')        #  $x$ 
6  dy = sp.symbols('dy')      #  $f'(y)$ 
7
8  y = x - fx / dx             #  $y$ 
9  ddx = (dy - dx) / (y - x)  #  $f''(x)$ 
10 halley = x - (2 * fx * dx) / (2 * dx * dx - fx * ddx)
11
12 print(sp.simplify(halley - x))
```

附录 2

对收敛性证明的代码

```
1  # %%
2  import sympy as sp
3
4  # %%
5  c2, c3, c4, c5, c6 = sp.symbols('c2 c3 c4 c5 c6')
6  c1 = 1
7  alpha = sp.symbols('alpha')
8  e = sp.symbols('e')
9  x = alpha + e
10 e1 = e
11 e2 = e ** 2
12 e3 = e ** 3
13 e4 = e ** 4
14 e5 = e ** 5
15 e6 = e ** 6
16
17 # %%
18 def taylor_1plus_theta(the, order):
```

```
19     inv = 1 / (1 + the)
20     return inv.series(e, 0, order)
21
22 def get0(the, order):
23     return taylor_1plus_theta(the, order) - taylor_1plus_theta(the, order).remove0()
24
25 o3 = get0(e, 3)
26 o4 = get0(e, 4)
27 o6 = get0(e, 6)
28 o7 = get0(e, 7)
29 o6, o7
30
31 # %%
32 def df(err, expand=False):
33     eee = lambda x: x
34     if expand:
35         eee = sp.expand
36     return eee(1 + 2*c2*err + 3*c3*err**2 + 4*c4*err**3 + 5*c5*err**4 + 6*c6*err**5 + o6)
37
38
39 def f(err):
40     return c1*err + c2*err**2 + c3*err**3 + c4*err**4 + c5*err**5 + c6*err**6 + o7
41
42
43 def taylor_inv(fenmu, order):
44     inv = 1 / fenmu
45     return 1 / inv.series(e, 0, order)
46
47 def texify(algo):
48     sp.print_latex(sp.collect(sp.expand(algo), e))
49     print()
50
51 # %%
52 fx = f(e)
53 fx
54
55 # %%
56 dx = df(e)
```

```
57 dx
58
59 # %%
60 print(" 公式 18")
61 texify(1 / taylor_inv(df(e), 6))
62
63 # %%
64 fx_div_dx = sp.collect(sp.expand(fx / taylor_inv(df(e), 6)), e)
65 print(" 公式 19")
66 texify(fx_div_dx)
67
68 # %%
69 y = x - fx_div_dx
70 print(" 公式 20")
71 texify(y)
72
73 # %%
74 dy = df(y - alpha, True)
75
76 # %%
77 print(" 公式 21")
78 texify(sp.collect(dy, e))
79
80 # %%
81 z = x - 2 * fx / taylor_inv(dx + dy, 6)
82 print(" 公式 24")
83 texify(sp.collect(sp.expand(z), e))
84
85 # %%
86 fz = f(z-alpha)
87 dz = df(z-alpha)
88 print(" 公式 25")
89 texify(sp.expand(z - fz / taylor_inv(dz, 6)))
90
91 # %%
92 fy = f(y - alpha)
93 newx = z - 2 * fz / taylor_inv(dz + dy, 6)
94 print(" 公式 27")
```



```
95 texify(sp.collect(sp.expand(newx), e))
```

```
96
```

```
97
```

附录 3

实验代码

```
1  # %%
2  import numpy as np
3  import matplotlib.pyplot as plt
4  import seaborn as sns
5  import sympy as sym
6  import decimal
7  from decimal import Decimal
8
9  plt.rcParams['text.usetex'] = False
10 plt.rcParams["font.family"] = "Arial Unicode MS"
11
12 # %%
13 mdl = np
14 roots = [
15     1.6319808055660636,
16     0.2575302854398608,
17     -1.207647827130919,
18     1.4044916482153411,
19     1.1525907367571583,
20     -1,
21     1,
22     1
23 ]
24 funcs = [
25     lambda x: x**3 + 4*x**2 - 15,
26     lambda x: x**2 - mdl.exp(x) - 3*x + 2,
27     lambda x: x*mdl.exp(x**2) - mdl.sin(x)**2 + 3*mdl.cos(x) + 5,
28     lambda x: mdl.sin(x)**2 - x**2 + 1,
29     lambda x: mdl.log(x**2 + 7*x + 14) - x - 2,
30     lambda x: (x - 4) * (x + 1)**4 / (mdl.exp(x)),
31     lambda x: mdl.exp(x**2 + 11*x - 12) - 1,
32     lambda x: mdl.arctan(mdl.exp(x + 3) - 1) * (x - 1)**2
```

```
33 ]
34
35 # %%
36 fig, axes = plt.subplots(2, 4, figsize=(10, 5),
37                          sharex=True, sharey=True)
38 sns.set_context('talk')
39 xmin, xmax = -2, 2
40 mdl = np
41 for i in range(8):
42     ax = axes.flatten()[i]
43     x = np.linspace(xmin, xmax, 1000)
44     y = funcs[i](x)
45     sns.lineplot(x=x, y=y, ax=ax, linewidth=3, label=rf'$f_{i+1}$')
46     ax.legend(loc='upper left')
47     ax.hlines(0, xmin, xmax, color='k', linestyle='--')
48     ax.set_xlim(xmin, xmax)
49     ax.set_ylim(-5, 5)
50     # ax.xaxis.set_minor_locator(plt.MultipleLocator(0.5))
51     ax.xaxis.set_major_locator(plt.MultipleLocator(1))
52
53 plt.subplots_adjust(left=0.05, right=0.99, bottom=0.07, top=0.98)
54 plt.savefig('../figure/eval-func.pdf')
55
56 # %%
57 mdl = sym
58 setattr(sym, 'arctan', sym.atan)
59 func_d1 = []
60 func_d2 = []
61 for f in funcs:
62     x = sym.symbols('x')
63     symf = f(x)
64     func_d1.append(sym.diff(symf, x, 1))
65     func_d2.append(sym.diff(symf, x, 2))
66
67 # for f in func_d1:
68 #     print(f"lambda x: {f},")
69 # print()
70 # for f in func_d2:
```

```

71 #     print(f"lambda x: {f},")
72
73 # %%
74 mdl = np
75 func_d1 = [
76     lambda x: 3*x**2 + 8*x,
77     lambda x: 2*x - np.exp(x) - 3,
78     lambda x: 2*x**2*np.exp(x**2) + np.exp(x**2) - 2*np.sin(x)*np.cos(x) - 3*np.sin(x),
79     lambda x: -2*x + 2*np.sin(x)*np.cos(x),
80     lambda x: (2*x + 7)/(x**2 + 7*x + 14) - 1,
81     lambda x: -(x - 4)*(x + 1)**4*np.exp(-x) + 4*(x - 4)*(x + 1)**3*np.exp(-x) + (x + 1)**4*np.exp(-x),
82     lambda x: (2*x + 11)*np.exp(x**2 + 11*x - 12),
83     lambda x: (x - 1)**2*np.exp(x + 3)/((np.exp(x + 3) - 1)**2 + 1) + (2*x - 2)*np.atan(np.exp(x + 3) - 1)
84 ]
85
86 func_d2 = [
87     lambda x: 2*(3*x + 4),
88     lambda x: 2 - np.exp(x),
89     lambda x: 4*x**3*np.exp(x**2) + 6*x*np.exp(x**2) + 2*np.sin(x)**2 - 2*np.cos(x)**2 - 3*np.cos(x),
90     lambda x: 2*(-np.sin(x)**2 + np.cos(x)**2 - 1),
91     lambda x: (-(2*x + 7)**2/(x**2 + 7*x + 14) + 2)/(x**2 + 7*x + 14),
92     lambda x: (x + 1)**2*(20*x + (x - 4)*(x + 1)**2 - 8*(x - 4)*(x + 1) - 2*(x + 1)**2 - 40)*np.exp(-x),
93     lambda x: ((2*x + 11)**2 + 2)*np.exp(x**2 + 11*x - 12),
94     lambda x: (x - 1)**2*(np.exp(x + 3) - 2*(np.exp(x + 3) - 1)*np.exp(2*x + 6)/((np.exp(x + 3) - 1)**2 + 1))/((
95 ]
96
97 # %%
98 np.seterr(all='raise')
99
100 def newton(f, fd, _, x):
101     return x - f(x) / fd(x)
102
103
104 def halley(f, fd1, fd2, x):
105     return x - (2 * f(x) * fd1(x)) / (2 * fd1(x)**2 - f(x)*fd2(x))
106
107
108 def neta(f, fd, _, x):

```

```
109     w = x - f(x) / fd(x)
110     z = w - f(x) / fd(x) * (f(x) - f(w)/2) / (f(x) - 5*f(w)/2)
111     xn = z - f(z) / fd(x) * (f(x) - f(w)) / (f(x) - 3*f(w))
112     return xn
113
114
115 def grau(f, fd, _, x):
116     y = x - f(x) / fd(x)
117     z = y - (y - x) / (2*f(y) - f(x)) * f(y)
118     xn = z - (y - x) / (2*f(y) - f(x)) * f(z)
119     return xn
120
121
122 def linz(f, fd, _, x):
123     y = x - f(x) / fd(x)
124     z = x - 2 * f(x) / (fd(x) + fd(y))
125     xn = z - f(z) / fd(z)
126     return xn
127
128
129 name_map = {
130     newton.__name__: '牛顿',
131     halley.__name__: 'Halley',
132     neta.__name__: 'NM',
133     grau.__name__: 'GM',
134     linz.__name__: '\\textbf{本文}',
135 }
136
137
138 def coc(x0, x1, x2, x3):
139     x0 = np.float128(x0)
140     x1 = np.float128(x1)
141     x2 = np.float128(x2)
142     x3 = np.float128(x3)
143     return np.log((x3 - x2) / (x2 - x1)) / np.log((x2 - x1) / (x1 - x0))
144
145
146 def compute(f, fd1, fd2, x, callback, root, tol, maxiter=100):
```

```

147     x0, x1, x2, x3 = x, x, x, x
148     for i in range(1, maxiter):
149         try:
150             tmp = x3
151             x3 = callback(f, fd1, fd2, x3)
152             x0 = x1
153             x1 = x2
154             x2 = tmp
155             if abs(x3 - x2) + abs(f(x3)) < tol:
156                 return i, (x0, x1, x2, x3)
157         except FloatingPointError as e:
158             print(x3, x2, e, '\n')
159             return -i, (x0, x1, x2, x3)
160     return -1, (x0, x1, x2, x3)
161
162 # %%
163 mdl = np
164 np.seterr('ignore')
165 init_vals = [
166     [1, 2],
167     [0, 1],
168     [-2, -1],
169     [1, 2],
170     [1, 2],
171     [-1.5, -0.5],
172     [0.5, 1.5],
173     [0.5, 1.5]
174 ]
175 # for i in [0]:# range(len(funcs)):
176 # print('\multirow{10}{*}{f_ ' + str(i) + '$}', end='')
177 for initx_idx in range(2):
178     for method in [newton, halley, neta, grau, linz]:
179         for i in [3, 7]:
180             inix = init_vals[i][initx_idx]
181             iters, xs = compute(funcs[i], func_d1[i], func_d2[i], inix, method, roots[i], 1e-12, 1000)
182             err = float(Decimal("{:.3g}".format(xs[3] - roots[i])))
183             if abs(err) < 1e-16:
184                 err = '$< 10^{-16}$'

```

```

185         else:
186             err = f'${sym.latex(err)}$'
187         if i > 3:
188             print('&', end='\t')
189         if method == newton:
190             print(' ', '\multirow{5}{*}{ ' + str(inix) + '}', name_map[method.__name__], iters, err, sep=' & '
191         else:
192             print(' ', ' ', name_map[method.__name__], iters, err, sep=' & \t', end='')
193     print('\\\\\\n')
194     if method == linz:
195         print('\\cline{2-5}\\cline{7-10}')
196     print('\\hline')
197
198     # %%
199     mdl = np
200     np.seterr(all='raise')
201     init_vals = [
202         [1, 2],
203         [0, 1],
204         [-2, -1],
205         [1, 2],
206         [1, 2],
207         [-1.5, -0.5],
208         [0.5, 1.5],
209         [0.5, 1.5]
210     ]
211     for i in [7]:
212         for inix in init_vals[i]:
213             for method in [newton, halley, neta, grau, linz]:
214                 # print(inix, method.__name__)
215                 iters, xs = compute(funcs[i], func_d1[i], func_d2[i], inix, method, roots[i], 1e-12, 1000)
216                 err = "{:.3g}".format(xs[3] - roots[i])
217                 print(i, inix, name_map[method.__name__], iters, err, sep='\t')
218
219
220

```