理学院

高等数值分析实验报告

非线性方程组的迭代解法

——Newton 法及其衍生法和拟 Newton 法的对比

姓　　名：袁　子　豪

学　　号：104971210327

指导教师：朱 国 甫 教 授

2021 年 12 月 29 日

摘要

考虑到非线性代数方程 $\boldsymbol{f}(\boldsymbol{x})=\mathbf{0}$，一般很难去直接求解。只能采用逐次逼近的迭代方法求解，即当知道解 $\boldsymbol{x}^*$ 的某一初始近似值 $\boldsymbol{x}_0$ 后，从它开始逐步逼近解 $\boldsymbol{x}^*$，我们经常选用 Newton 迭代法解以上方程。Newton 迭代法是迭代法的一种，是求解函数方程 $f(x)=0$ 及函数方程组 $\boldsymbol{f}(\boldsymbol{x})=\mathbf{0}$ 的一种有效方法，其基本特征是计算格式简单且收敛较快。作为工程中常用的数据处理分析方法，Newton 迭代法客观上存在优缺点。优点：Newton 迭代法具有平方收敛的速度，所以在迭代过程中只要迭代几次就会得到很精确的解，这是 Newton 迭代法比简单迭代法优越的地方；缺点：选定的初值要接近方程的解，否则有可能得不到收敛的结果。再者，Newton 迭代法计算量比较大，因为每次迭代除计算函数值外还要计算微商值。因此，在 Newton 法的基础上又衍生出了一些改进的迭代算法，修正 Newton 法、割线法和拟 Newton 法。本试验报告将从理论重述作为出发点，对 Newton 法、修正 Newton 法、割线法和拟 Newton 法的原理进行重述及证明，其后通过在 MATLAB 模拟计算来说明如何用 Newton 法、修正 Newton 法、割线法和拟 Newton 法去逼近求解非线性方程组。最后，本文通过模拟实验的结果数据去验证支撑 Newton 迭代法、修正 Newton 法、割线法和拟 Newton 法的一些优劣性。

关键词: 非线性方程组，Newton 法，修正 Newton 法，割线法，拟 Newton 法

目　录

摘要 **I**

第 1 章　实验目标及准备工作 **1**

1.1　非线性方程组的数值解法 1

1.2　Newton 法简介 1

1.3　实验环境 2

1.4　编译环境 3

第 2 章　Newton 法及其衍生算法 **4**

2.1　Newton 法原理 4

2.2　Newton 法的局部收敛性 5

2.3　修正 Newton 法 9

2.4　割线法 10

2.5　拟 Newton 法 12

第 3 章　实验过程（附代码） **14**

3.1　非线性方程组迭代求解算法的伪代码 14

3.1.1　Newton 法的伪代码 14

3.1.2　修正 Newton 法的伪代码 15

3.1.3　割线法 15

3.1.4　拟 Newton 法 16

3.2　非线性方程组迭代算法代码的编写 17

第 4 章　实验结果 **24**

4.1　简例 24

4.2　各方法收敛速度的可视化展示 25

4.3　总结展望 28

第 1 章　实验目标及准备工作

本实验的目标为在 MATLAB 环境中使用 MATLAB 自带的基础函数对 Newton 法法进行编程以实现对任意非线性方程组的自适应求解问题，并且使用 MATLAB 自带的 plot 函数对迭代中的误差大小进行绘图，具象其收敛速度，对其性能进行分析并可视化。本实验的具体目标如下：

1. 掌握 Newton 法求解非线性方程组的基本思想和步骤。
2. 理解 Newton 法求解非线性方程组过程中的优缺点。
3. 掌握 Newton 法求解非线性方程组在 MATLAB 中的计算过程。

1.1　非线性方程组的数值解法

在研究非线性方程组的数值解法之前，首先要给非线性方程组于一个合理的定义：

定义 1 (非线性方程组)**.** n 个变量 n 个方程 $(n>1)$ 的方程组表示为 $f_i(x_1,x_2,\ldots,x_n)=0$（其中 $i=1,2,\ldots,n$），若 f_i 中至少有一个是非线性函数，则称上述表示的方程组 $f_i(x_1,x_2,\ldots,x_n)=0, i=1,2,\ldots,n$ 为非线性方程组，并简记为

$$\boldsymbol{f}(\boldsymbol{x})=\boldsymbol{0}$$

其中 $\boldsymbol{f}(\boldsymbol{x})=(f_1,f_2,\ldots,f_n)^{\mathrm{T}}\in\mathbb{F}^n$，$f_i(\boldsymbol{x})=f_i(x_1,x_2,\ldots,x_n)$ 且 $x\in\mathbb{R}^n$。

若存在 $\boldsymbol{x}^*\in\mathbb{R}^n$，使 $\boldsymbol{f}(\boldsymbol{x}^*)=\boldsymbol{0}$，则称 $\boldsymbol{x}^*$ 为非线性方程组的解。上述方程组可能有一个解或多个解，也可能有无穷多解或无解。对非线性方程组解的存在性的研究远不如线性方程组那样成熟，现有的解法也不及线性方程组那样有效。除极特殊的方程外，一般不能用直接方法求得解析解。目前，主要采用迭代算法求数值解以逼近真值。根据不同思想构造收敛于解 $\boldsymbol{x}^*$ 的迭代序列 $\{\hat{\boldsymbol{x}}_k\}(k=0,1,\ldots)$，即可得到求解非线性方程组的各种迭代法。本文则是以 Newton 法为主，去研究非线性方程组的 Newton 法。

1.2　Newton 法简介

Newton 迭代法（Newton's method）又称为 Newton-拉夫逊方法（Newton-Raphson method），它是 Newton 在 17 世纪提出的一种在实数域和复数域上近似求解方程的方法。多数方程不存在求根公式，因此求精确根非常困难，甚至不可能，从而寻找方程的近似根就显得特别重要。

Newton 迭代法是求方程根的重要方法之一，其最大优点是在方程 $f(x)=0$ 的单根附近具有平方收敛，而且该法还可以用来求方程的重根、复根，此时线性收敛，但是可通过一些方法变成超线性收敛。

简单迭代法是用直接的方法从原方程中隐含地解出 x，从而确定出求解迭代函数 $\varphi(x)$。而 Newton 迭代法是用一种间接而特殊的方法来确定 $\varphi(x)$。先以一元方程为例，给出 Newton 迭代求解思想。

假设 x_k 是非线性方程为 $f(x)=0$ 的一个近似根，把 $f(x)$ 在 x_k 处泰勒展开：

$$f(x)=f\left(x_k\right)+f'\left(x_k\right)\left(x-x_k\right)+\frac{f''\left(x_k\right)}{2!}\left(x-x_k\right)^2+\cdots\cdots$$

若取前两项来近似代 $f(x)$（称为对 $f(x)$ 局部线性光滑处理），则得到近似的线性方程

$$f(x)\approx f\left(x_k\right)+f'\left(x_k\right)\left(x-x_k\right)=0$$

设 $f'\left(x_k\right)\neq 0$，令其解为 x_{k+1}，则得

$$x_{k+1}=x_k-\frac{f\left(x_k\right)}{f'\left(x_k\right)}$$

这称为方程 $f(x)=0$ 的 Newton 迭代公式。它对应的迭代方程为

$$x=x-\frac{f(x)}{f'(x)}$$

显然与 $f(x)=0$ 同解，故其迭代函数为

$$\varphi(x)=x-\frac{f(x)}{f'(x)}\quad\left(f'(x)\neq 0\right)$$

在 $f(x)=0$ 的根 x^* 的某个邻域 $R(|x-x^*|\leq\delta)$ 内，$f(x)\approx 0$

$$\left|\varphi'(x)\right|=\frac{\left|f''(x)\right|\left|f(x)\right|}{\left|f'(x)\right|^2}\leq L<1$$

对于多元非线性方程组，Newton 法的思想依旧如此，具体的形式引入与性质及证明将在第二章进行说明。

1.3 实验环境

本实验运行在 matlab(R2021b) 的 macOS 版本下，界面如图1.1所示。

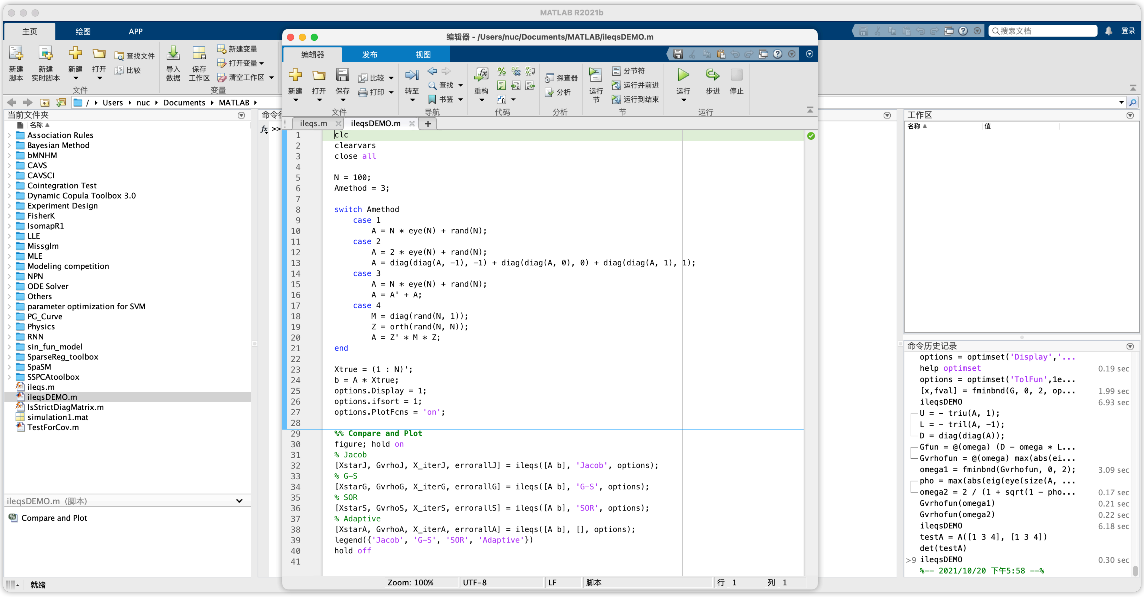

图 1.1: matlab 运行环境示意图

1.4 编译环境

本报告编译环境为 macOS 下的 Texpad，界面如图1.2所示。

图 1.2: Texpad 编译排版环境示意图

第 2 章　Newton 法及其衍生算法

在第 1 章第 2 节中，我们已经说明 Newton 法是解方程 $f(x)=0$ 的基本方法之一。目前使用得较多的方法均以 Newton 法为基础, 是 Newton 法的修改与变形。本章则是对 Newton 法求解一般非线性方程组的迭代思想，迭代形式，迭代收敛性进行详细叙述。

2.1　Newton 法原理

这里先假定映射 $\boldsymbol{f}:\mathbf{D}\subset\mathbb{R}^n\to\mathbb{R}^n$ 在开凸集 $\mathbf{D}$ 中二次 **G**-可导，且 $\boldsymbol{f}''(\boldsymbol{x})$ 于 $\mathbf{D}$ 连续。设 $\boldsymbol{x}^*$ 是非线性方程组 $\boldsymbol{f}(\boldsymbol{x})=\mathbf{0}$ 的解。$\boldsymbol{x}_0\in\mathbf{D}$ 为 $\boldsymbol{x}^*$ 的初始近似。利用 Taylor 展式，得

$$\boldsymbol{f}(\boldsymbol{x})=\boldsymbol{f}\left(\boldsymbol{x}_0\right)+\boldsymbol{f}'\left(\boldsymbol{x}_0\right)\left(\boldsymbol{x}-\boldsymbol{x}_0\right)+\int_0^1\left(\boldsymbol{x}-\boldsymbol{x}_0\right)^{\top}\boldsymbol{f}''\left(\boldsymbol{x}_0+t\left(\boldsymbol{x}-\boldsymbol{x}_0\right)\right)\left(\boldsymbol{x}-\boldsymbol{x}_0\right)(1-t)\mathrm{d}t \tag{2.1}$$

若是 $\boldsymbol{x}_0\approx\boldsymbol{x}^*$，略去上式的高阶小量，用线性方程组去局部光滑非线性方程组：

$$\boldsymbol{f}(\boldsymbol{x}_0)+\boldsymbol{f}'(\boldsymbol{x}_0)(\boldsymbol{x}-\boldsymbol{x}_0)=\mathbf{0}$$

则推导出式（2.1）的解 $\boldsymbol{x}_1$ 为

$$\boldsymbol{x}_1=\boldsymbol{x}_0-\left[\boldsymbol{f}'\left(\boldsymbol{x}_0\right)\right]^{-1}\boldsymbol{f}\left(\boldsymbol{x}_0\right)$$

一般 $\boldsymbol{x}_1$ 较 $\boldsymbol{x}_0$ 更接近 $\boldsymbol{x}^*$。可用 $\boldsymbol{x}_1$ 作为 $\boldsymbol{x}^*$ 新的初始近似, 导出:

$$\boldsymbol{f}\left(\boldsymbol{x}_1\right)+\boldsymbol{f}'\left(\boldsymbol{x}_1\right)\left(\boldsymbol{x}-\boldsymbol{x}_1\right)=\mathbf{0}$$

设其解为 $\boldsymbol{x}_2$

$$\boldsymbol{x}_2=\boldsymbol{x}_1-\left[\boldsymbol{f}'\left(\boldsymbol{x}_1\right)\right]^{-1}\boldsymbol{f}\left(\boldsymbol{x}_1\right)$$

依此类推，得 Newton 法迭代公式为

$$\boldsymbol{x}_{k+1}=\boldsymbol{x}_k-\left[\boldsymbol{f}'\left(\boldsymbol{x}_k\right)\right]^{-1}\boldsymbol{f}\left(\boldsymbol{x}_k\right)\quad k=0,1,2,\ldots \tag{2.2}$$

如第 1 章第 2 节中所述类似，但考虑到该公式为多元非线性方程组的迭代公式，需要注意的是：公式中 $\boldsymbol{f}'\left(\boldsymbol{x}_k\right)$ 是 $\boldsymbol{f}$ 在 $\boldsymbol{x}_k$ 处的 Jacobi 矩阵。这意味着需要用 n 个超切平面在 $\boldsymbol{x}_k$ 处来局部线性光滑 $\boldsymbol{f}(\boldsymbol{x})=\mathbf{0}$，并以此来确定方程组的解。在每步计算 Jacobi 矩阵 $\boldsymbol{f}'\left(\boldsymbol{x}_k\right)$ 的逆时，若 n 比较大，则计算十分麻烦。在实际计算时，一般采用如下形式

$$\left\{\begin{array}{l}\boldsymbol{f}'\left(\boldsymbol{x}_k\right)\Delta\boldsymbol{x}_k=-\boldsymbol{f}\left(\boldsymbol{x}_k\right)\\ \boldsymbol{x}_{k+1}=\boldsymbol{x}_k+\Delta\boldsymbol{x}_k\end{array}\right.\quad k=0,1,2,\ldots \tag{2.3}$$

即每一步迭代都需要解一个 n 阶线性方程组。

2.2 Newton 法的局部收敛性

在给出了 Newton 法的迭代思想和迭代公式之后，我们需要对 Newton 法的收敛性进行探究。首先有下面的局部收敛性定理

定理 1. *设 $\boldsymbol{x}^*$ 是方程组 $\boldsymbol{f}(\boldsymbol{x}) = \mathbf{0}$ 的一个解, $\boldsymbol{f}: \mathbb{R}^n \to \mathbb{R}^n$ 在包含 $\boldsymbol{x}^*$ 的邻域 $\mathbf{D}$ 中 $\mathbf{F}$-可微, $\boldsymbol{f}'(\boldsymbol{x})$ 在 $\boldsymbol{x}^*$ 连续, 且 $\boldsymbol{f}'(\boldsymbol{x}^*)$ 非奇异, 那么存在闭球 $\bar{\mathbf{S}}(\boldsymbol{x}^*, r) = \{\boldsymbol{x} \mid \|\boldsymbol{x} - \boldsymbol{x}^*\| \le r\} \subset \mathbf{D}$, 使得对任意的 $\boldsymbol{x}_0 \in \bar{\mathbf{S}}(\boldsymbol{x}^*, r)$，由 Newton 法的迭代公式 (2.1) 产生的迭代序列是适定的。并且 $\boldsymbol{x}_k \to \boldsymbol{x}^* (a.e.\ k \to \infty)$。*

在给出证明前，先给出一条重要的引理：线性方程组的迭代法时，得出单步定常迭代法收敛的充分必要条件为迭代矩阵的谱半径小于 1；对非线性方程组的单步定常迭代法，我们也有下面的类似定理.

引理 1 (Ostrowski)**.** *设映射 $\varphi: \mathbf{D} \subset \mathbb{R}^n \to \mathbb{R}^n$ 有一不动点 $\boldsymbol{x}^*$ 是 $\mathbf{D}$ 的内点，且在 $\boldsymbol{x}^*$ 处 $\mathbf{F}$-可导，$\varphi'(\boldsymbol{x}^*)$ 的谱半径*

$$\rho\left(\varphi'\left(\boldsymbol{x}^*\right)\right) = \zeta < \mathbf{1},$$

则存在开球 $\mathbf{S} = \mathbf{S}(\boldsymbol{x}^*, \delta) \subset \mathbf{D}$，对任意的初值 $\boldsymbol{x}_0 \in \mathbf{S}$，由 $\boldsymbol{x}_{k+1} = \varphi(\boldsymbol{x}_k)$ 产生的迭代序列 $\{\boldsymbol{x}_k\} \subset S$ 并且收敛于 $\boldsymbol{x}^*$。

引理 1 的证明:

由于 $\zeta < 1$，故存在 $\varepsilon > 0$，使 $\alpha = \zeta + 2\varepsilon < 1$。由线性代数知识，存在一种范数使

$$\left\|\varphi'\left(\boldsymbol{x}^*\right)\right\|_\varepsilon \le \zeta + \varepsilon$$

另一方面，由 φ 在 $\boldsymbol{x}^*$ 处 $\mathbf{F}$-可导的定义得，对上述 $\varepsilon > 0, \exists \delta > 0$，使 $\mathbf{S} = \mathbf{S}(\boldsymbol{x}^*, \delta) \subset \mathbf{D}$，且对任意 $\boldsymbol{x} \in \mathbf{S}$，有

$$\left\|\varphi(\boldsymbol{x}) - \varphi\left(\boldsymbol{x}^*\right) - \varphi'\left(\boldsymbol{x}^*\right)\left(\boldsymbol{x} - \boldsymbol{x}^*\right)\right\| \le \varepsilon \|\boldsymbol{x} - \boldsymbol{x}^*\|$$

于是

$$\begin{aligned}
\|\varphi(\boldsymbol{x}) - \boldsymbol{x}^*\|_\varepsilon &= \left\|\varphi(\boldsymbol{x}) - \varphi\left(\boldsymbol{x}^*\right)\right\|_\varepsilon \\
&\le \left\|\varphi(\boldsymbol{x}) - \varphi\left(\boldsymbol{x}^*\right) - \varphi'\left(\boldsymbol{x}^*\right)\left(\boldsymbol{x} - \boldsymbol{x}^*\right)\right\|_\varepsilon + \left\|\varphi'\left(\boldsymbol{x}^*\right)\left(\boldsymbol{x} - \boldsymbol{x}^*\right)\right\|_\varepsilon \\
&\le (\zeta + 2\varepsilon)\|\boldsymbol{x} - \boldsymbol{x}^*\|_\varepsilon = \alpha\|\boldsymbol{x} - \boldsymbol{x}^*\|_\varepsilon
\end{aligned}$$

对任意 $\boldsymbol{x}_0 \in \mathbf{S}$，$\boldsymbol{x}_1 = \varphi(\boldsymbol{x}_0)$，有

$$\|\boldsymbol{x}_1 - \boldsymbol{x}^*\|_\varepsilon = \|\varphi(\boldsymbol{x}_0) - \boldsymbol{x}^*\|_\varepsilon \le \alpha\|\boldsymbol{x}_0 - \boldsymbol{x}^*\|_\varepsilon < \delta$$

因此，$\boldsymbol{x}_1 \in \mathbf{S}$。若已知 $\boldsymbol{x}_{k-1} \in \mathbf{S}$，则由

$$\|\boldsymbol{x}_k - \boldsymbol{x}^*\|_\varepsilon = \|\varphi(\boldsymbol{x}_{k-1}) - \boldsymbol{x}^*\|_\varepsilon \le \alpha\|\boldsymbol{x}_{k-1} - \boldsymbol{x}^*\|_\varepsilon \le \alpha^k\|\boldsymbol{x}_0 - \boldsymbol{x}^*\|_\varepsilon < \delta$$

则可知 $\boldsymbol{x}_k \in \mathbf{S}$，这说明迭代序列 $\{\boldsymbol{x}_k\} \subset \mathbf{S}$ 是适定的。又因为 $0 < \alpha < 1$，并且可得 $\boldsymbol{x}_k \to \boldsymbol{x}^*$。

引理 1 证毕

定理 1 的证明:

由于 $\boldsymbol{f}'(\boldsymbol{x}^*)$ 非奇异，所以 $\det \boldsymbol{f}'(\boldsymbol{x}^*) \neq \mathbf{0}$。又 $\boldsymbol{f}'(\boldsymbol{x})$ 在 $\boldsymbol{x}^*$ 连续，因此 $\det \boldsymbol{f}'(\boldsymbol{x})$ 在 $\boldsymbol{x}^*$ 处亦连续，从而存在 $\delta_1 > 0$，当 $\|\boldsymbol{x}-\boldsymbol{x}^*\| \leq \delta_1$ 时，恒有 $\det \boldsymbol{f}'(\boldsymbol{x}) \neq \mathbf{0}$，即 $[\boldsymbol{f}'(\boldsymbol{x})]^{-1}$ 在闭球 $\bar{\mathbf{S}}(\boldsymbol{x}^*, \delta_1) = \{\boldsymbol{x} \mid \|\boldsymbol{x}-\boldsymbol{x}^*\| \leq \delta_1\} \subset \mathbf{D}$ 上存在令

$$\varphi(\boldsymbol{x}) = \boldsymbol{x} - [\boldsymbol{f}'(\boldsymbol{x})]^{-1} \boldsymbol{f}(\boldsymbol{x})$$

我们将证明 $\varphi'(\boldsymbol{x}^*) = \mathbf{0}$。根据 $\boldsymbol{f}'(\boldsymbol{x})$ 在 $\boldsymbol{x}^*$ 处的连续性，对任给的 $\varepsilon > \mathbf{0}$，存在 $\delta > 0$，使得当 $\boldsymbol{x} \in \bar{\mathbf{S}}(\boldsymbol{x}^*, \delta) = \{\boldsymbol{x} \mid \|\boldsymbol{x}-\boldsymbol{x}^*\| \leq \delta < \delta_1\}$ 时有 $\|\boldsymbol{f}'(\boldsymbol{x}) - \boldsymbol{f}'(\boldsymbol{x}^*)\| \leq \varepsilon$。令 $\beta = \|[\boldsymbol{f}'(\boldsymbol{x}^*)]^{-1}\|$，则有

$$\begin{aligned}
\|[\boldsymbol{f}'(\boldsymbol{x})]^{-1}\| &= \|[\boldsymbol{f}'(\boldsymbol{x})]^{-1} - [\boldsymbol{f}'(\boldsymbol{x}^*)]^{-1} + [\boldsymbol{f}'(\boldsymbol{x}^*)]^{-1}\| \\
&\leq \|[\boldsymbol{f}'(\boldsymbol{x})]^{-1} - [\boldsymbol{f}'(\boldsymbol{x}^*)]^{-1}\| + \|[\boldsymbol{f}'(\boldsymbol{x}^*)]^{-1}\| \\
&\leq \|[\boldsymbol{f}'(\boldsymbol{x}^*)]^{-1}\| \times \|\boldsymbol{f}'(\boldsymbol{x}) - \boldsymbol{f}'(\boldsymbol{x}^*)\| \times \|[\boldsymbol{f}'(\boldsymbol{x})]^{-1}\| + \beta \\
&\leq \beta\varepsilon \|[\boldsymbol{f}'(\boldsymbol{x})]^{-1}\| + \beta
\end{aligned}$$

选取 ε, 使 $0 < \varepsilon < (2\beta)^{-1}$, 则

$$\|[\boldsymbol{f}'(\boldsymbol{x})]^{-1}\| \leq \frac{\beta}{1-\varepsilon\beta} < 2\beta$$

由于 $\boldsymbol{f}$ 在 $\boldsymbol{x}^*$ 处 $\mathbf{F}$-可微，我们可以选取足够小的正数 $\boldsymbol{\delta}$，使得当 $\boldsymbol{x} \in \bar{\mathbf{S}}(\boldsymbol{x}^*, \delta) = \{\boldsymbol{x} \mid \|\boldsymbol{x}-\boldsymbol{x}^*\| \leq \delta < \delta_1\}$ 时有

$$\|\boldsymbol{f}(\boldsymbol{x}) - \boldsymbol{f}(\boldsymbol{x}^*) - \boldsymbol{f}'(\boldsymbol{x}^*)(\boldsymbol{x}-\boldsymbol{x}^*)\| \leq \varepsilon \|\boldsymbol{x}-\boldsymbol{x}^*\|$$

从而有

$$\begin{aligned}
&\|\varphi(\boldsymbol{x}) - \varphi(\boldsymbol{x}^*) - \mathbf{0}(\boldsymbol{x}-\boldsymbol{x}^*)\| = \|\boldsymbol{x}-\boldsymbol{x}^* - [\boldsymbol{f}'(\boldsymbol{x})]^{-1} \boldsymbol{f}(\boldsymbol{x})\| \\
&= \|[\boldsymbol{f}'(\boldsymbol{x})]^{-1} \boldsymbol{f}'(\boldsymbol{x})(\boldsymbol{x}-\boldsymbol{x}^*) - [\boldsymbol{f}'(\boldsymbol{x})]^{-1} \boldsymbol{f}'(\boldsymbol{x}^*)(\boldsymbol{x}-\boldsymbol{x}^*) \\
&\quad + [\boldsymbol{f}'(\boldsymbol{x})]^{-1} \boldsymbol{f}'(\boldsymbol{x}^*)(\boldsymbol{x}-\boldsymbol{x}^*) + [\boldsymbol{f}'(\boldsymbol{x})]^{-1} \boldsymbol{f}(\boldsymbol{x}^*) - [\boldsymbol{f}'(\boldsymbol{x})]^{-1} \boldsymbol{f}(\boldsymbol{x})\| \\
&\leq \|[\boldsymbol{f}'(\boldsymbol{x})]^{-1}\| \times \|[\boldsymbol{f}'(\boldsymbol{x}) - \boldsymbol{f}'(\boldsymbol{x}^*)](\boldsymbol{x}-\boldsymbol{x}^*)\| \\
&\quad + \|[\boldsymbol{f}'(\boldsymbol{x})]^{-1}\| \times \|\boldsymbol{f}(\boldsymbol{x}) - \boldsymbol{f}(\boldsymbol{x}^*) - \boldsymbol{f}'(\boldsymbol{x}^*)(\boldsymbol{x}-\boldsymbol{x}^*)\| \\
&\leq (2\beta\varepsilon + 2\beta\varepsilon)\|\boldsymbol{x}-\boldsymbol{x}^*\| = 4\beta\varepsilon\|\boldsymbol{x}-\boldsymbol{x}^*\|
\end{aligned}$$

于是证得 $\varphi'(x^*) = 0$。从而 $\rho(\varphi'(x^*)) < 1$。根据引理 1, 存在闭球:$\bar{\mathbf{S}}(\boldsymbol{x}^*, r) = \{\boldsymbol{x} \mid \|\boldsymbol{x}-\boldsymbol{x}^*\| \leq r\} \subset \mathbf{D}$ 使得对任意的 $\boldsymbol{x}_0 \in \bar{\mathbf{S}}(\boldsymbol{x}^*, r)$，由 Newton 法 (2.1) 产生的迭代序列是适定的。并且 $\boldsymbol{x}_k \to \boldsymbol{x}^*$。

定理 1 证毕

定理 2 (Kantorovich). 假设给定了 $\mathbb{R}^n$ 中的一个开集 $\mathbf{D}$，$\mathbf{D}_0$ 为一凸集，且 $\bar{\mathbf{D}}_0 \subset \mathbf{D}$。设对于给定的 $\boldsymbol{x}_0 \in \mathbf{D}_0$，存在正常数 $r, \alpha, \beta, \gamma, h$，它们具有下列性质

$$\mathbf{S}\left(\boldsymbol{x}_0, r\right) \subseteq \mathbf{D}_0, \quad h = \frac{\alpha\beta\gamma}{2} < 1, \quad r = \frac{\alpha}{1-h}$$

若 $\mathbf{f}: \mathbb{R}^n \to \mathbf{R}^n$ 在 $\mathbf{D}$ 中连续，在 $\mathbf{D}_0$ 上 $\mathbf{F}$-可微，且具有下列性质：

（1） $\|\boldsymbol{f}'(\boldsymbol{x}) - \boldsymbol{f}'(\boldsymbol{y})\| \leq \gamma \|\boldsymbol{x} - \boldsymbol{y}\|, \forall \boldsymbol{x}, \boldsymbol{y} \in \mathbf{D}_0$

（2） $[\boldsymbol{f}'(\boldsymbol{x})]^{-1}$ 存在，且：$\|\, [\boldsymbol{f}'(\boldsymbol{x})]^{-1} \| \leq \beta, \forall \boldsymbol{x} \in \mathbf{D}_0$

（3） $\|\, [\boldsymbol{f}'(\boldsymbol{x}_0)]^{-1} \boldsymbol{f}(\boldsymbol{x}_0)\| \leq \alpha$

则

（1） 从 $\boldsymbol{x}_0$ 出发，$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - [\boldsymbol{f}'(\boldsymbol{x}_k)]^{-1} \boldsymbol{f}(\boldsymbol{x}_k)$ 是适定的，且 $\boldsymbol{x}_k \in \mathbf{S}(\boldsymbol{x}_0, r)$

（2） $\lim\limits_{k\to\infty} \boldsymbol{x}_k = \boldsymbol{x}^*$ 极限存在，且 $\boldsymbol{x}^* \in \bar{\mathbf{S}}(\boldsymbol{x}_0, r), \boldsymbol{f}(\boldsymbol{x}^*) = \mathbf{0}$

（3） *Newton* 法至少为二阶收敛

（4） 对 $k = 0, 1, 2, \ldots$，有 $\|\boldsymbol{x}_k - \boldsymbol{x}^*\| \leq \alpha \frac{h^{2^k - 1}}{1 - h^{2^k}}$

在给出定理 2 的证明前，先给出引理 2 及其证明：

引理 2. 设 $\boldsymbol{f}: \mathbb{R}^n \to \mathbb{R}^m$ 在凸集 $\mathbf{S} \subset \mathbb{R}^n$ 上连续可导，且 $\mathbf{F}$-导数 $\boldsymbol{f}'$ 满足

$$\|\boldsymbol{f}'(\boldsymbol{u}) - \boldsymbol{f}'(\boldsymbol{v})\| \leq \alpha \|\boldsymbol{u} - \boldsymbol{v}\|^p, \forall \boldsymbol{u}, \boldsymbol{v} \in \mathbf{S}$$

其中 $\alpha \geq 0, p \geq 0$ 为常数，则对任意的 $\boldsymbol{x}, \boldsymbol{y} \in \mathbf{S}$ 有

$$\|\boldsymbol{f}(\boldsymbol{y}) - \boldsymbol{f}(\boldsymbol{x}) - \boldsymbol{f}'(\boldsymbol{x})(\boldsymbol{y} - \boldsymbol{x})\| \leq \frac{\alpha}{1+p} \|\boldsymbol{y} - \boldsymbol{x}\|^{1+p}$$

引理 2 的证明

由引理 2 中形式可得

$$\begin{aligned}
\|\boldsymbol{f}(\boldsymbol{y}) - \boldsymbol{f}(\boldsymbol{x}) - \boldsymbol{f}'(\boldsymbol{x})(\boldsymbol{y} - \boldsymbol{x})\| &= \left\| \int_0^1 [\boldsymbol{f}'(\boldsymbol{x} + t(\boldsymbol{y} - \boldsymbol{x})) - \boldsymbol{f}'(\boldsymbol{x})](\boldsymbol{y} - \boldsymbol{x}) \mathrm{d}t \right\| \\
&\leq \int_0^1 \|\boldsymbol{f}'(\boldsymbol{x} + t(\boldsymbol{y} - \boldsymbol{x})) - \boldsymbol{f}'(\boldsymbol{x})\| \times \|(\boldsymbol{y} - \boldsymbol{x})\| \mathrm{d}t \\
&\leq \int_0^1 \alpha \|t(\boldsymbol{y} - \boldsymbol{x})\|^p \times \|(\boldsymbol{y} - \boldsymbol{x})\| \mathrm{d}t \\
&\leq \alpha \|(\boldsymbol{y} - \boldsymbol{x})\|^{1+p} \int_0^1 t^p \mathrm{d}t = \frac{\alpha}{1+p} \|(\boldsymbol{y} - \boldsymbol{x})\|^{1+p}
\end{aligned}$$

引理 2 证毕

定理 2 的证明

(1) 对 $\boldsymbol{x}_0 \in \mathbf{D}_0$，$[\boldsymbol{f}'(\boldsymbol{x})]^{-1}$ 存在，根据条件（3），有

$$\|\boldsymbol{x}_1 - \boldsymbol{x}_0\| = \| - [\boldsymbol{f}'(\boldsymbol{x}_0)]^{-1}\boldsymbol{f}(\boldsymbol{x}_0)\| \leq \alpha = \alpha h^{2^0-1} < r$$

所以 $\boldsymbol{x}_1 \in \mathbf{S}(\boldsymbol{x}_0, r)$，先设 $\boldsymbol{x}_j \in \mathbf{S}(\boldsymbol{x}_0, r), j = 0, 1, 2, \ldots, k$。则根据条件 (2)$\boldsymbol{x}_{k+1}$ 有定义且

$$\begin{aligned}\|\boldsymbol{x}_{k+1} - \boldsymbol{x}_k\| &= \| - [\boldsymbol{f}'(\boldsymbol{x}_k)]^{-1}\boldsymbol{f}(\boldsymbol{x}_k)\| \leq \beta\|\boldsymbol{f}(\boldsymbol{x}_k)\| \\ &= \beta\|\boldsymbol{f}(\boldsymbol{x}_k) - \boldsymbol{f}(\boldsymbol{x}_{k-1}) + \boldsymbol{f}(\boldsymbol{x}_{k-1})\| \\ &= \beta\|\boldsymbol{f}(\boldsymbol{x}_k) - \boldsymbol{f}(\boldsymbol{x}_{k-1}) - \boldsymbol{f}'(\boldsymbol{x}_{k-1})(\boldsymbol{x}_k - \boldsymbol{x}_{k-1})\|\end{aligned}$$

由引理 2($p = 1$ 的情况) 可知

$$\|\boldsymbol{x}_{k+1} - \boldsymbol{x}_k\| \leq \frac{\beta\gamma}{2}\|\boldsymbol{x}_k - \boldsymbol{x}_{k-1}\|^2$$

于是有

$$\|\boldsymbol{x}_{k+1} - \boldsymbol{x}_k\| \leq \alpha h^{2^k-1} \tag{2.4}$$

事实上，当 $k = 0$ 时，上面已经验证 (2.4) 成立。当 (2.4) 对 $k - 1$ 成立时

$$\|\boldsymbol{x}_{k+1} - \boldsymbol{x}_k\| \leq \frac{\beta\gamma}{2}\|\boldsymbol{x}_k - \boldsymbol{x}_{k-1}\|^2 \leq \frac{\beta\gamma}{2}(\alpha h^{2^{k-1}-1})^2 = \alpha h^{2^k-1}$$

因此，(2.4) 对 k 也成立，则根据数学归纳法可知，(2.4) 对任意的 $k = 0, 1, 2, \ldots$ 皆成立，则

$$\begin{aligned}\|\boldsymbol{x}_{k+1} - \boldsymbol{x}_0\| &= \|\sum_{i=0}^{k}(\boldsymbol{x}_{i+1} - \boldsymbol{x}_i)\| \leq \sum_{i=0}^{k}\|\boldsymbol{x}_{i+1} - \boldsymbol{x}_i\| \\ &\leq \alpha(1 + h + h^3 + \cdots + h^{2^k-1}) \\ &\leq \alpha/(1-h) = r\end{aligned}$$

则有 $\boldsymbol{x}_{k+1} \in \mathbf{S}(\boldsymbol{x}_0, r)$，即 $\{\boldsymbol{x}_k\}$ 是适定的，且有 $\boldsymbol{x}_k \in \mathbf{S}(\boldsymbol{x}_0, r)$。

(2) 由 (2.4) 式可得：

$$\begin{aligned}\|\boldsymbol{x}_{k+m} - \boldsymbol{x}_k\| &= \|\sum_{i=1}^{m}(\boldsymbol{x}_{k+i} - \boldsymbol{x}_{k+i-1})\| \leq \sum_{i=0}^{k}\|\boldsymbol{x}_{k+i} - \boldsymbol{x}_{k+i-1}\| \\ &\leq \alpha h^{2^k-1}(1 + h^{2^k} + h^{2^k3} + \cdots + h^{2^k2^m-1}) \\ &\leq \alpha h^{2^k-1}/(1-h^{2^k}) \to 0\end{aligned} \tag{2.5}$$

所以 $\{\boldsymbol{x}_k\}$ 是一个 Cauchy 序列，从而极限存在且有

$$\lim_{k\to\infty}\boldsymbol{x}_k = \boldsymbol{x}^*,\ \boldsymbol{x}^* \in \bar{\mathbf{S}}(\boldsymbol{x}_0, r)$$

对 $k \geq 0$ 时，

$$\begin{aligned}\|\boldsymbol{f}'(\boldsymbol{x}_k)\| &= \|\boldsymbol{f}'(\boldsymbol{x}_k) - \boldsymbol{f}'(\boldsymbol{x}_0) + \boldsymbol{f}'(\boldsymbol{x}_0)\| \leq \|\boldsymbol{f}'(\boldsymbol{x}_k) - \boldsymbol{f}'(\boldsymbol{x}_0)\| + \|\boldsymbol{f}'(\boldsymbol{x}_0)\| \\ &\leq \gamma\|\boldsymbol{x}_k - \boldsymbol{x}_0\| + \|\boldsymbol{f}'(\boldsymbol{x}_0)\| \leq \gamma r + \|\boldsymbol{f}'(\boldsymbol{x}_0)\|\end{aligned}$$

且又因为 $\boldsymbol{f}(\boldsymbol{x}_k) = -\boldsymbol{f}'(\boldsymbol{x}_k)(\boldsymbol{x}_{k+1} - \boldsymbol{x}_k)$，则

$$\|\boldsymbol{f}(\boldsymbol{x}_k)\| \leq \|\boldsymbol{f}'(\boldsymbol{x}_k)\| \times \|\boldsymbol{x}_{k+1} - \boldsymbol{x}_k\| \leq (\gamma r + \|\boldsymbol{f}'(\boldsymbol{x}_0)\|)\|\boldsymbol{x}_{k+1} - \boldsymbol{x}_k\|$$

得到

$$\lim_{k\to\infty} \|\boldsymbol{f}(\boldsymbol{x}_k)\| = \|\boldsymbol{f}(\boldsymbol{x}^*)\| = 0$$

(3) 由引理 2（$p = 1$ 的情况）可得：

$$\begin{aligned}\|\boldsymbol{x}_{k+1} - \boldsymbol{x}^*\| &= \|\boldsymbol{x}_k - [\boldsymbol{f}'(\boldsymbol{x}_k)]^{-1}\boldsymbol{f}(\boldsymbol{x}_k) - \boldsymbol{x}^* + [\boldsymbol{f}'(\boldsymbol{x}_k)]^{-1}\boldsymbol{f}(\boldsymbol{x}^*)\| \\ &= \|[\boldsymbol{f}'(\boldsymbol{x}_k)]^{-1}[\boldsymbol{f}(\boldsymbol{x}^*) - \boldsymbol{f}(\boldsymbol{x}_k) - \boldsymbol{f}'(\boldsymbol{x}_k)(\boldsymbol{x}^* - \boldsymbol{x}_k)]\| \\ &\leq \|[\boldsymbol{f}'(\boldsymbol{x}_k)]^{-1}\| \times \|\boldsymbol{f}(\boldsymbol{x}^*) - \boldsymbol{f}(\boldsymbol{x}_k) - \boldsymbol{f}'(\boldsymbol{x}_k)(\boldsymbol{x}^* - \boldsymbol{x}_k)\| \\ &\leq \frac{\beta\gamma}{2}\|\boldsymbol{x}^* - \boldsymbol{x}_k\|^2 = \frac{h}{\alpha}\|\boldsymbol{x}^* - \boldsymbol{x}_k\|^2\end{aligned}$$

则得到结论：Newton 法至少二阶收敛。

(3) 根据式 (2.5)，我们有

$$\|\boldsymbol{x}^* - \boldsymbol{x}_k\| = \lim_{m\to\infty} \|\boldsymbol{x}_{m+k} - \boldsymbol{x}_k\| \leq \alpha h^{2^k-1}/(1 - h^{2^k})$$

定理 2 证毕

2.3 修正 Newton 法

在 Newton 法的实际计算中，我们发现 Newton 法收敛速度高，计算量很大：

（1）在每一步迭代中，要计算 n 个函数值；

（2）形成 Jacobi 矩阵 $\boldsymbol{f}'(\boldsymbol{x}_k)$ 要计算 n^2 个偏导数值；

（3）解一个 n 阶线性方程组。

为了减少 Newton 法的计算量，加快迭代的计算速度，于是对 Newton 法 (2.2) 进行简单的改进，得到修正 Newton 法迭代公式为

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - [\boldsymbol{f}'(\boldsymbol{x}_0)]^{-1}\boldsymbol{f}(\boldsymbol{x}_k) \quad k = 0, 1, 2, \ldots \tag{2.6}$$

可以看出，在 (2.6) 中只是对 (2.2) 的将 $[\boldsymbol{f}'(\boldsymbol{x}_k)]^{-1}$ 固定为了 $[\boldsymbol{f}'(\boldsymbol{x}_0)]^{-1}$，从而在每步迭代中减少了 Jacobi 矩阵 $\boldsymbol{f}'(\boldsymbol{x}_k)$ 的 n^2 个偏导数值的计算。这样做使得计算量大为减少，但降低了收敛速度。可以在用 (2.6) 计算若干步后, 重新形成 Jacobi 矩阵，以提高收敛速度。

2.4 割线法

应用 Newton 法时，在每一步迭代中都要形成 Jacobi 矩阵 $\boldsymbol{f}'(\boldsymbol{x})$。这需要计算 n^2 个偏导数值。当 $\boldsymbol{f}(\boldsymbol{x}_k)$ 的分量 $f_i(\boldsymbol{x}_k)$ 的偏导数无法计算或计算过程很复杂时，应用 Newton 法将会有很大困难。为了克服这种困难，这一节，我们来介绍割线法，它避免了求导过程。Newton 法是用以下线性函数

$$\boldsymbol{L}_k(\boldsymbol{x})=\boldsymbol{f}'(\boldsymbol{x}_k)(\boldsymbol{x}-\boldsymbol{x}_k)+\boldsymbol{f}(\boldsymbol{x}_k)\approx\boldsymbol{f}(\boldsymbol{x})$$

一般地，可用线性函数

$$\boldsymbol{L}_k(\boldsymbol{x})=\boldsymbol{A}_k\boldsymbol{x}+\boldsymbol{b}_k\approx\boldsymbol{f}(\boldsymbol{x}) \tag{2.7}$$

去逼近 $\boldsymbol{f}(\boldsymbol{x})$，这里的 $\boldsymbol{A}_k$ 为 n 阶矩阵，$\boldsymbol{b}_k$ 为 n 阶向量。若是用插值的方法去确定 $\boldsymbol{A}_k$ 和 $\boldsymbol{b}_k$，则得到其对应的迭代公式。这就是割线法的基本思想。

设已求得 $\boldsymbol{f}(\boldsymbol{x})=\boldsymbol{0}$ 的 k 次近似为 $\boldsymbol{x}_k$，记 $\boldsymbol{x}_k=\boldsymbol{x}_k^{(0)}$。取 n 个辅助点 $\boldsymbol{x}_k^{(1)},\boldsymbol{x}_k^{(2)},\ldots,\boldsymbol{x}_k^{(n)}$ 及其对应的函数方程组的值向量 $\boldsymbol{f}(\boldsymbol{x}_k^{(1)}),\boldsymbol{f}(\boldsymbol{x}_k^{(2)}),\ldots,\boldsymbol{f}(\boldsymbol{x}_k^{(n)})$，代入到式 (2.7) 得

$$\begin{cases}\boldsymbol{L}_k(\boldsymbol{x}_k)=\boldsymbol{A}_k\boldsymbol{x}_k+\boldsymbol{b}_k=\boldsymbol{f}(\boldsymbol{x}_k)\\ \boldsymbol{L}_k(\boldsymbol{x}_k^{(j)})=\boldsymbol{A}_k\boldsymbol{x}_k^{(j)}+\boldsymbol{b}_k=\boldsymbol{f}(\boldsymbol{x}_k^{(j)}),\ j=1,2,\ldots,n\end{cases}$$

两式相减，得

$$\begin{aligned}&\boldsymbol{A}_k(\boldsymbol{x}_k^{(j)}-\boldsymbol{x}_k)=\boldsymbol{f}(\boldsymbol{x}_k^{(j)})-\boldsymbol{f}(\boldsymbol{x}_k),\ j=1,2,\ldots,n\\ &\boldsymbol{b}_k=\boldsymbol{f}(\boldsymbol{x}_k)-\boldsymbol{A}_k\boldsymbol{x}_k\end{aligned} \tag{2.8}$$

记

$$\begin{cases}\boldsymbol{H}_k=[\boldsymbol{x}_k^{(1)}-\boldsymbol{x}_k,\ldots,\boldsymbol{x}_k^{(n)}-\boldsymbol{x}_k]\\ \boldsymbol{\Gamma}_k=[\boldsymbol{f}(\boldsymbol{x}_k^{(1)})-\boldsymbol{f}(\boldsymbol{x}_k),\ldots,\boldsymbol{f}(\boldsymbol{x}_k^{(n)})-\boldsymbol{f}(\boldsymbol{x}_k)]\end{cases}$$

则可将 (2.8) 式简记为

$$\boldsymbol{A}_k\boldsymbol{H}_k=\boldsymbol{\Gamma}_k$$

若 $\boldsymbol{H}_k$ 非奇异，即 $\boldsymbol{x}_k^{(1)}-\boldsymbol{x}_k,\ldots,\boldsymbol{x}_k^{(n)}-\boldsymbol{x}_k$ 线性无关，则 $\boldsymbol{A}_k=\boldsymbol{\Gamma}_k\boldsymbol{H}_k^{-1}$ 若 $\boldsymbol{\Gamma}_k$ 也非奇异，即 $\boldsymbol{f}(\boldsymbol{x}_k^{(1)})-\boldsymbol{f}(\boldsymbol{x}_k),\ldots,\boldsymbol{f}(\boldsymbol{x}_k^{(n)})-\boldsymbol{f}(\boldsymbol{x}_k)$ 也是线性无关，则 $\boldsymbol{A}_k^{-1}=\boldsymbol{H}_k\boldsymbol{\Gamma}_k^{-1}$ 因此 (2.7) 式的系数可唯一确定。令 $\boldsymbol{L}_k(\boldsymbol{x})=\boldsymbol{0}$, 得

$$\begin{cases}\boldsymbol{A}_k(\boldsymbol{x}-\boldsymbol{x}_k)+\boldsymbol{f}(\boldsymbol{x}_k)=\boldsymbol{0}\\ \boldsymbol{A}_k=\boldsymbol{\Gamma}_k\boldsymbol{H}_k^{-1}\end{cases}$$

解出 $\boldsymbol{x}$，令 $\boldsymbol{x}_{k+1}=\boldsymbol{x}$，便得到迭代公式：

$$\begin{cases}\boldsymbol{x}_{k+1}=\boldsymbol{x}_k-\boldsymbol{A}_k^{-1}\boldsymbol{f}(\boldsymbol{x}_k)\\ \boldsymbol{A}_k^{-1}=\boldsymbol{H}_k\boldsymbol{\Gamma}_k^{-1}\end{cases} \tag{2.9}$$

这就是解非线性方程组 $\boldsymbol{f}(\boldsymbol{x})=\boldsymbol{0}$ 的割线法。

割线法完全避免了计算偏导数，但要计算辅助点的函数值。适当地选择辅助点是减少计算量、保证收敛速度的关键。下面讨论两种具体的辅助点选择方法：

- $n+1$ **点割线法**

取

$$\boldsymbol{H}_k = [\boldsymbol{x}_{k-1}-\boldsymbol{x}_k, \dots, \boldsymbol{x}_{k-n}-\boldsymbol{x}_k]$$
$$\boldsymbol{\Gamma}_k = [\boldsymbol{f}(\boldsymbol{x}_{k-1})-\boldsymbol{f}(\boldsymbol{x}_k), \dots, \boldsymbol{f}(\boldsymbol{x}_{k-n})-\boldsymbol{f}(\boldsymbol{x}_k)]$$

引入记号

$$\bar{\boldsymbol{H}}_k = [\boldsymbol{x}_{k-1}-\boldsymbol{x}_k, \boldsymbol{x}_{k-2}-\boldsymbol{x}_{k-1}, \dots, \boldsymbol{x}_{k-n}-\boldsymbol{x}_{k-n+1}]$$
$$\bar{\boldsymbol{\Gamma}}_k = [\boldsymbol{f}(\boldsymbol{x}_{k-1})-\boldsymbol{f}(\boldsymbol{x}_k), \boldsymbol{f}(\boldsymbol{x}_{k-2})-\boldsymbol{f}(\boldsymbol{x}_{k-1}), \dots, \boldsymbol{f}(\boldsymbol{x}_{k-n})-\boldsymbol{f}(\boldsymbol{x}_{k-n+1})]$$

并记

$$\boldsymbol{P} = \begin{bmatrix} 1 & -1 & & & \\ & 1 & -1 & & \\ & & \ddots & \ddots & \\ & & & 1 & -1 \\ & & & & 1 \end{bmatrix}$$

则

$$\bar{\boldsymbol{H}}_k = \boldsymbol{H}_k\boldsymbol{P}, \quad \bar{\boldsymbol{\Gamma}}_k = \boldsymbol{\Gamma}_k\boldsymbol{P}, \quad \boldsymbol{A}_k = \boldsymbol{\Gamma}_k\boldsymbol{H}_k^{-1} = \bar{\boldsymbol{\Gamma}}_k\bar{\boldsymbol{H}}_k^{-1}$$

因此 $n+1$ 点割线法的计算公式可写成为

$$\begin{cases} \boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \bar{\boldsymbol{H}}_k\boldsymbol{z}_k \\ \bar{\boldsymbol{\Gamma}}_k\boldsymbol{z}_k = \boldsymbol{f}(\boldsymbol{x}_k) \end{cases} \tag{2.10}$$

$n+1$ 点割线法的特点是计算量少，但是不稳定。

- **两点割线法**

当 $\boldsymbol{H}_k$ 只与 $\boldsymbol{x}_k, \boldsymbol{x}_{k-1}$ 有关时，称之为两点割线法。记

$$\boldsymbol{x}_k = (x_{k,1}, \dots, x_{k,n})$$
$$h_j^{(k)} = x_{k-1,j} - x_{k,j},\ j = 1, 2, \dots, n$$
$$\boldsymbol{e}_j\text{为第}j\text{维单位向量}$$

两点割线法最简单的形式是取

$$\boldsymbol{H}_k = \mathrm{Diag}[h_1^{(k)}, \dots, h_n^{(k)}] \tag{2.11}$$

此时

$$\boldsymbol{\Gamma}_k = [\boldsymbol{f}(\boldsymbol{x}_k + h_1^{(k)}\boldsymbol{e}_1) - \boldsymbol{f}(\boldsymbol{x}_k), \dots, \boldsymbol{f}(\boldsymbol{x}_k + h_n^{(k)}\boldsymbol{e}_n) - \boldsymbol{f}(\boldsymbol{x}_k)]$$
$$\boldsymbol{A}_k = \boldsymbol{\Gamma}_k\boldsymbol{H}_k^{-1} = \left[\frac{\boldsymbol{f}(\boldsymbol{x}_k + h_1^{(k)}\boldsymbol{e}_1) - \boldsymbol{f}(\boldsymbol{x}_k)}{h_1^{(k)}}, \dots, \frac{\boldsymbol{f}(\boldsymbol{x}_k + h_n^{(k)}\boldsymbol{e}_n) - \boldsymbol{f}(\boldsymbol{x}_k)}{h_n^{(k)}}\right]$$

则迭代公式变为

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \boldsymbol{A}_k^{-1}\boldsymbol{f}(\boldsymbol{x}_k) \tag{2.12}$$

2.5 拟 Newton 法

割线法的实质是在 $\boldsymbol{x}_{k+1}$ 点构造 Jacobi 矩阵 $\boldsymbol{f}'(\boldsymbol{x}_k)$ 的近似矩阵 $\boldsymbol{A}_k$，从而避免导数的计算。但割线法用于方程组的求解时, 数值稳定性不好。在构造 $n+1$ 点割线法时，当 $n \leq k$ 时，有以下关系：

$$
\begin{aligned}
&\boldsymbol{A}_k(\boldsymbol{x}_j - \boldsymbol{x}_k) = \boldsymbol{f}(\boldsymbol{x}_j) - \boldsymbol{f}(\boldsymbol{x}_k) \quad j = k-1, k-2, \ldots, k-n \\
&\boldsymbol{A}_k(\boldsymbol{x}_{k+1} - \boldsymbol{x}_k) = -\boldsymbol{f}(\boldsymbol{x}_k) \\
&\boldsymbol{A}_{k+1}(\boldsymbol{x}_j - \boldsymbol{x}_{k+1}) = \boldsymbol{f}(\boldsymbol{x}_j) - \boldsymbol{f}(\boldsymbol{x}_{k+1}) \quad j = k, k-1, \ldots, k-n+1
\end{aligned}
$$

因此

$$
\begin{aligned}
&\boldsymbol{A}_k\left(\boldsymbol{x}_{j+1} - \boldsymbol{x}_j\right) = \boldsymbol{f}\left(\boldsymbol{x}_{j+1}\right) - \boldsymbol{f}\left(\boldsymbol{x}_j\right) \quad j = k-1, k-2, \ldots, k-n \\
&\boldsymbol{A}_{k+1}\left(\boldsymbol{x}_{j+1} - \boldsymbol{x}_j\right) = \boldsymbol{f}\left(\boldsymbol{x}_{j+1}\right) - \boldsymbol{f}\left(\boldsymbol{x}_j\right) \quad j = k, k-1, \ldots, k-n+1
\end{aligned}
$$

相减得：

$$
\begin{aligned}
&\left(\boldsymbol{A}_{k+1} - \boldsymbol{A}_k\right)\left(\boldsymbol{x}_{j+1} - \boldsymbol{x}_j\right) = \boldsymbol{0} \quad j = k-1, \quad k-2, \quad \cdots \quad k-n+1 \\
&\left(\boldsymbol{A}_{k+1} - \boldsymbol{A}_k\right)\left(\boldsymbol{x}_{k+1} - \boldsymbol{x}_k\right) = \boldsymbol{f}\left(\boldsymbol{x}_{k+1}\right)
\end{aligned}
$$

若

$$
\boldsymbol{x}_k - \boldsymbol{x}_{k-1}, \cdots, \boldsymbol{x}_{k-n+2} - \boldsymbol{x}_{k-n+1}
$$

线性无关，则 $\Delta \boldsymbol{A}_k = \boldsymbol{A}_{k+1} - \boldsymbol{A}_k$ 的秩不超过 1。

于是可利用递推公式，$\boldsymbol{A}_{k+1} = \boldsymbol{A}_k + \Delta \boldsymbol{A}_k$，并限制 $\Delta \boldsymbol{A}_k$ 的秩为 1 导出算法。为此次只需取

$$
\Delta \boldsymbol{A}_k = \boldsymbol{u}_k \boldsymbol{v}_k^{\mathsf{T}}
$$

其中 $\boldsymbol{v}_k$ 与 $\boldsymbol{x}_k - \boldsymbol{x}_{k-1}, \cdots, \boldsymbol{x}_{k-n+2} - \boldsymbol{x}_{k-n+1}$ 正交，$\boldsymbol{u}_k$ 为 $\boldsymbol{f}\left(\boldsymbol{x}_{k+1}\right)$ 的若干倍。

这种通过对 $\boldsymbol{A}_k$ 作修正而得到 $\boldsymbol{A}_{k+1}$ 的方法是拟 Newton 法的基本出发点。拟 Newton 法的算法如下:

$$
\begin{cases}
\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \boldsymbol{A}_k^{-1} \boldsymbol{f}\left(\boldsymbol{x}_k\right), \quad k = 0, 1, 2, \ldots \\
\boldsymbol{A}_{k+1}\left(\boldsymbol{x}_{k+1} - \boldsymbol{x}_k\right) = \boldsymbol{f}\left(\boldsymbol{x}_{k+1}\right) - \boldsymbol{f}\left(\boldsymbol{x}_k\right), \\
\boldsymbol{A}_{k+1} = \boldsymbol{A}_k + \Delta \boldsymbol{A}_k, \quad \operatorname{rank}\left(\Delta \boldsymbol{A}_k\right) = m \geq 1
\end{cases} \tag{2.13}
$$

其中 $\Delta \boldsymbol{A}_k$ 为 $\boldsymbol{A}_k$ 的修正矩阵，$m = 1$ 或 2。第二行为拟 Newton 方程，$\boldsymbol{A}_{k+1}$ 关于点 $\boldsymbol{x}_k$ 及 $\boldsymbol{x}_{k+1}$ 具有“差商”性质。为获得超收敛速度，$\boldsymbol{A}_k$ 应随 k 的增大逐渐逼近 $\boldsymbol{f}'(\boldsymbol{x}^*)$。

- **Broyden 方法**

在式 (2.13) 中，限制 $\Delta \boldsymbol{A}_k$ 的秩为 1。令

$$
\Delta \boldsymbol{A}_k = \boldsymbol{u}_k \boldsymbol{v}_k^{\mathsf{T}}
$$

$$\boldsymbol{v}_k = \boldsymbol{x}_{k+1} - \boldsymbol{x}_k$$

代入拟 Newton 方程得：

$$\left[\boldsymbol{A}_k + \boldsymbol{u}_k \left(\boldsymbol{x}_{k+1} - \boldsymbol{x}_k\right)^{\mathsf{T}}\right]\left(\boldsymbol{x}_{k+1} - \boldsymbol{x}_k\right) = \boldsymbol{f}\left(\boldsymbol{x}_{k+1}\right) - \boldsymbol{f}\left(\boldsymbol{x}_k\right)$$

所以

$$\boldsymbol{u}_k = \frac{\boldsymbol{f}\left(\boldsymbol{x}_{k+1}\right)}{\left(\boldsymbol{x}_{k+1} - \boldsymbol{x}_k\right)^{\mathsf{T}}\left(\boldsymbol{x}_{k+1} - \boldsymbol{x}_k\right)}$$

则得到

$$\begin{cases} \boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \boldsymbol{A}_k^{-1}\boldsymbol{f}\left(\boldsymbol{x}_k\right), \quad k = 0, 1, 2, \ldots \\ \boldsymbol{A}_{k+1} = \boldsymbol{A}_k + \dfrac{\boldsymbol{f}\left(\boldsymbol{x}_{k+1}\right)\left(\boldsymbol{x}_{k+1} - \boldsymbol{x}_k\right)^{\mathsf{T}}}{\left(\boldsymbol{x}_{k+1} - \boldsymbol{x}_k\right)^{\mathsf{T}}\left(\boldsymbol{x}_{k+1} - \boldsymbol{x}_k\right)} \end{cases} \tag{2.14}$$

其中 $\boldsymbol{A}_0 \approx \boldsymbol{f}'\left(\boldsymbol{x}_0\right)$。

可用下面的 Sherman-Morrison 矩阵逆修正公式

$$\left(\boldsymbol{A} + \boldsymbol{u}\boldsymbol{v}^{\mathsf{T}}\right)^{-1} = \boldsymbol{A}^{-1} - \frac{\boldsymbol{A}^{-1}\boldsymbol{u}\boldsymbol{v}^{\mathsf{T}}\boldsymbol{A}^{-1}}{1 + \boldsymbol{v}^{\mathsf{T}}\boldsymbol{A}^{-1}\boldsymbol{u}}$$

将 (2.14) 改为逆 Broyden 秩 1 公式:

$$\begin{cases} \boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \boldsymbol{B}_k\boldsymbol{f}\left(\boldsymbol{x}_k\right), \quad k = 0, 1, 2, \cdots, \\ \boldsymbol{B}_{k+1} = \boldsymbol{B}_k - \dfrac{\boldsymbol{B}_k\boldsymbol{f}\left(\boldsymbol{x}_{k+1}\right)\left(\boldsymbol{x}_{k+1} - \boldsymbol{x}_k\right)^{\mathsf{T}}\boldsymbol{B}_k}{\left(\boldsymbol{x}_{k+1} - \boldsymbol{x}_k\right)^{\mathsf{T}}\boldsymbol{B}_k\left[\boldsymbol{f}\left(\boldsymbol{x}_{k+1}\right) - \boldsymbol{f}\left(\boldsymbol{x}_k\right)\right]} \end{cases} \tag{2.15}$$

其中 $\boldsymbol{B}_0 \approx \boldsymbol{f}'\left(\boldsymbol{x}_0\right)^{-1}$。

第 3 章　实验过程（附代码）

与一般的仿真处理全流程类似，本实验的过程主要包括数据的生成、迭代方法的选择、计算结果分析和数据的可视化展示等几个主要步骤。实验的基本思路如图3.1所示。根据基本思路图，本实验对计算所有流程进行了编程，且均已上传到 GitHub[1]上。

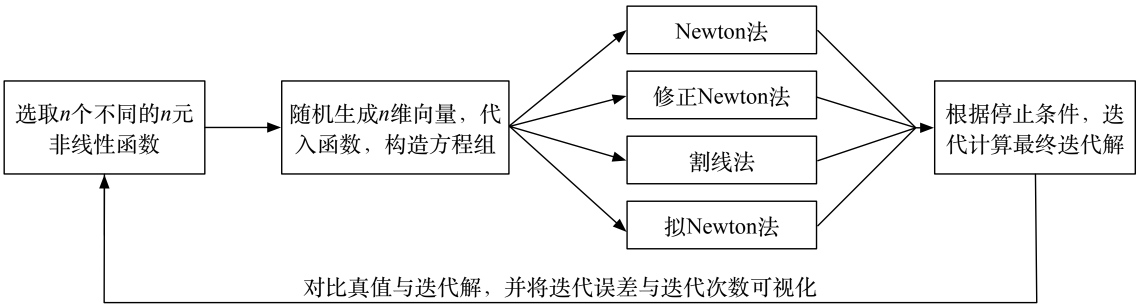

图 3.1: 本实验基本思路

3.1　非线性方程组迭代求解算法的伪代码

3.1.1　Newton 法的伪代码

为了对不同的线性方程组求解，本实验采用将迭代方法也作为输入变量进行编程。

Algorithm 1 求解非线性方程组的 Newton 法

Input: 非线性方程组 $\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{0}$ 及其对应的函数的一阶导 $\boldsymbol{f}'(\boldsymbol{x})$, 范数 p，初始值 $\boldsymbol{x}^{(0)}$，迭代残差的容忍误差 $\|\varepsilon^*\|$，迭代解的容忍误差 $\|\varepsilon_{\boldsymbol{x}}^*\|$，最大迭代次数 K；

Output: 最终迭代解 $\boldsymbol{x}^*$，每次迭代解构成的矩阵 $\boldsymbol{x}_{iter}$，每次迭代误差范数 $\|\boldsymbol{\varepsilon}\|_p$；

begin

设置 $k = 0$；$\boldsymbol{x}_{iter} = [\,]$；$\|\boldsymbol{\varepsilon}\|_p = [\,]$；
while $k < K$ **do**
　$k = k + 1$；
　根据式 (2.3) 和线性方程的迭代解法 ILEQS 计算 $\Delta\boldsymbol{x}_k$；
　计算迭代解 $\boldsymbol{x}_k = \boldsymbol{x}_{k-1} + \Delta\boldsymbol{x}_{k-1}$；$\boldsymbol{x}_{iter} = [\boldsymbol{x}_{iter}, \boldsymbol{x}_k]$；
　计算误差 $\|\varepsilon^{(k)}\|_p = \|\boldsymbol{f}(\boldsymbol{x}_k)\|_p$；$\|\boldsymbol{\varepsilon}\|_p = [\|\boldsymbol{\varepsilon}\|_p, \|\varepsilon^{(k)}\|_p]$；$\|\varepsilon_{\boldsymbol{x}}^{(k)}\|_p = \|\Delta\boldsymbol{x}_{k-1}\|_p$；
　if $(\|\varepsilon^{(k)}\|_p < \|\varepsilon^*\|)\ \ \|\ \ (\|\varepsilon_{\boldsymbol{x}}^{(k)}\|_p < \|\varepsilon_{\boldsymbol{x}}^*\|)$ **then**
　　break;
return $\boldsymbol{x}^* = \boldsymbol{x}_k$，$\boldsymbol{x}_{iter}$，$\|\boldsymbol{\varepsilon}\|_p$；

end

[1] https://github.com/yuanzihao945/IEQS

3.1.2 修正 Newton 法的伪代码

Algorithm 2 求解非线性方程组的修正 Newton 法

Input: 非线性方程组 $\boldsymbol{f}(\boldsymbol{x})=\boldsymbol{0}$ 及其对应的函数的一阶导 $\boldsymbol{f}'(\boldsymbol{x})$, 范数 p，初始值 $\boldsymbol{x}^{(0)}$，迭代残差的容忍误差 $\|\varepsilon^*\|$，迭代解的容忍误差 $\|\varepsilon^*_{\boldsymbol{x}}\|$，最大迭代次数 K；

Output: 最终迭代解 $\boldsymbol{x}^*$，每次迭代解构成的矩阵 $\boldsymbol{x}_{iter}$，每次迭代误差范数 $\|\boldsymbol{\varepsilon}\|_p$；

begin

设置 $k=0, m=0$ 和修正步长 s；$\boldsymbol{x}_{iter}=[\,]$；$\|\boldsymbol{\varepsilon}\|_p=[\,]$；
while $k<K$ **do**
    $k=k+1$；$m_k=\mathrm{ceil}(k/s)$；
    if $m_k>m$ **then**
        $m=m_k$; 根据式 (2.6) 和线性方程的迭代解法 ILEQS 计算 $\Delta\boldsymbol{x}_m$；
    计算迭代解 $\boldsymbol{x}_k=\boldsymbol{x}_{k-1}+\Delta\boldsymbol{x}_m$；$\boldsymbol{x}_{iter}=[\boldsymbol{x}_{iter},\boldsymbol{x}_k]$；
    计算误差 $\|\varepsilon^{(k)}\|_p=\|\boldsymbol{f}(\boldsymbol{x}_k)\|_p$；$\|\boldsymbol{\varepsilon}\|_p=[\|\boldsymbol{\varepsilon}\|_p,\|\varepsilon^{(k)}\|_p]$；$\|\varepsilon^{(k)}_{\boldsymbol{x}}\|_p=\|\Delta\boldsymbol{x}_m\|_p$；
    if $(\|\varepsilon^{(k)}\|_p<\|\varepsilon^*\|)\ \ \|\ \ (\|\varepsilon^{(k)}_{\boldsymbol{x}}\|_p<\|\varepsilon^*_{\boldsymbol{x}}\|)$ **then**
        break;
return $\boldsymbol{x}^*=\boldsymbol{x}_k$，$\boldsymbol{x}_{iter}$，$\|\boldsymbol{\varepsilon}\|_p$；

end

3.1.3 割线法

Algorithm 3 求解非线性方程组的 2 点割线法

Input: 非线性方程组 $\boldsymbol{f}(\boldsymbol{x})=\boldsymbol{0}$, 范数 p，初始值 $\boldsymbol{x}^{(0)}$，迭代残差的容忍误差 $\|\varepsilon^*\|$，迭代解的容忍误差 $\|\varepsilon^*_{\boldsymbol{x}}\|$，最大迭代次数 K；

Output: 最终迭代解 $\boldsymbol{x}^*$，每次迭代解构成的矩阵 $\boldsymbol{x}_{iter}$，每次迭代误差范数 $\|\boldsymbol{\varepsilon}\|_p$；

begin

设置 $k=0$；$\boldsymbol{x}_{iter}=[\,]$；$\|\boldsymbol{\varepsilon}\|_p=[\,]$；并根据式 (2.11) 随机生成 $\boldsymbol{H}_k$；
while $k<K$ **do**
    $k=k+1$；
    根据式 (2.11)、(2.12) 计算迭代解 $\boldsymbol{x}_k=\boldsymbol{x}_{k-1}-\boldsymbol{A}_k^{-1}\boldsymbol{f}(\boldsymbol{x}_k)$ 和 $\boldsymbol{H}_k$；
    计算误差 $\|\varepsilon^{(k)}\|_p=\|\boldsymbol{f}(\boldsymbol{x}_k)\|_p$；$\|\boldsymbol{\varepsilon}\|_p=[\|\boldsymbol{\varepsilon}\|_p,\|\varepsilon^{(k)}\|_p]$；$\|\varepsilon^{(k)}_{\boldsymbol{x}}\|_p=\|\Delta\boldsymbol{x}_m\|_p$；
    if $(\|\varepsilon^{(k)}\|_p<\|\varepsilon^*\|)\ \|\ (\|\varepsilon^{(k)}_{\boldsymbol{x}}\|_p<\|\varepsilon^*_{\boldsymbol{x}}\|)$ **then**
        break;
return $\boldsymbol{x}^*=\boldsymbol{x}_k$，$\boldsymbol{x}_{iter}$，$\|\boldsymbol{\varepsilon}\|_p$；

end

Algorithm 4 求解非线性方程组的 $n+1$ 点割线法

Input: 非线性方程组 $\boldsymbol{f}(\boldsymbol{x})=\boldsymbol{0}$, 范数 p，初始值 $\boldsymbol{x}^{(0)}$，迭代残差的容忍误差 $\|\varepsilon^*\|$，迭代解的容忍误差 $\|\varepsilon_{\boldsymbol{x}}^*\|$，最大迭代次数 K；

Output: 最终迭代解 $\boldsymbol{x}^*$，每次迭代解构成的矩阵 $\boldsymbol{x}_{iter}$，每次迭代误差范数 $\|\varepsilon\|_p$；

begin

1: 设置 $k=0$；$\boldsymbol{x}_{iter}=[\,]$；$\|\varepsilon\|_p=[\,]$；在 $\boldsymbol{x}^{(0)}$ 附近随机生成 k 个辅助点 $\boldsymbol{x}_{-1}$，$\boldsymbol{x}_{-2}$，$\ldots$，$\boldsymbol{x}_{-n}$;

2: **while** $k<K$ **do**

3: 计算 $\begin{array}{l}\bar{\boldsymbol{H}}_k=[\boldsymbol{x}_{k-1}-\boldsymbol{x}_k,\boldsymbol{x}_{k-2}-\boldsymbol{x}_{k-1},\ldots,\boldsymbol{x}_{k-n}-\boldsymbol{x}_{k-n+1}]\\ \bar{\boldsymbol{\Gamma}}_k=[\boldsymbol{f}(\boldsymbol{x}_{k-1})-\boldsymbol{f}(\boldsymbol{x}_k),\boldsymbol{f}(\boldsymbol{x}_{k-2})-\boldsymbol{f}(\boldsymbol{x}_{k-1}),\ldots,\boldsymbol{f}(\boldsymbol{x}_{k-n})-\boldsymbol{f}(\boldsymbol{x}_{k-n+1})]\end{array}$

4: 根据式 (2.10) 计算迭代解 $\boldsymbol{x}_{k+1}$；$k=k+1$；

5: 计算误差 $\|\varepsilon^{(k)}\|_p=\|\boldsymbol{f}(\boldsymbol{x}_k)\|_p;\|\varepsilon\|_p=[\|\varepsilon\|_p,\|\varepsilon^{(k)}\|_p];\|\varepsilon_{\boldsymbol{x}}^{(k)}\|_p=\|\boldsymbol{x}_{k+1}-\boldsymbol{x}_k\|_p$；

6: **if** $(\|\varepsilon^{(k)}\|_p<\|\varepsilon^*\|)\ \|\ (\|\varepsilon_{\boldsymbol{x}}^{(k)}\|_p<\|\varepsilon_{\boldsymbol{x}}^*\|)$ **then**

7: break;

8: **return** $\boldsymbol{x}^*=\boldsymbol{x}_k$，$\boldsymbol{x}_{iter}$，$\|\varepsilon\|_p$；

end

3.1.4 拟 Newton 法

Algorithm 5 求解非线性方程组的拟 Newton 法

Input: 非线性方程组 $\boldsymbol{f}(\boldsymbol{x})=\boldsymbol{0}$, 范数 p，初始值 $\boldsymbol{x}^{(0)}$，迭代残差的容忍误差 $\|\varepsilon^*\|$，迭代解的容忍误差 $\|\varepsilon_{\boldsymbol{x}}^*\|$，最大迭代次数 K；

Output: 最终迭代解 $\boldsymbol{x}^*$，每次迭代解构成的矩阵 $\boldsymbol{x}_{iter}$，每次迭代误差范数 $\|\varepsilon\|_p$；

begin

1: 设置 $k=0$；$\boldsymbol{x}_{iter}=[\,]$；$\|\varepsilon\|_p=[\,]$；计算 $\boldsymbol{B}_0=\boldsymbol{f}'(\boldsymbol{x}^{(0)})$;

2: **while** $k<K$ **do**

3: 根据式 (2.15) 计算 $\begin{cases}\boldsymbol{x}_{k+1}=\boldsymbol{x}_k-\boldsymbol{B}_k\boldsymbol{f}\left(\boldsymbol{x}_k\right), \quad k=0,1,2,\cdots,\\ \boldsymbol{B}_{k+1}=\boldsymbol{B}_k-\dfrac{\boldsymbol{B}_k\boldsymbol{f}\left(\boldsymbol{x}_{k+1}\right)\left(\boldsymbol{x}_{k+1}-\boldsymbol{x}_k\right)^{\mathsf{T}}\boldsymbol{B}_k}{\left(\boldsymbol{x}_{k+1}-\boldsymbol{x}_k\right)^{\mathsf{T}}\boldsymbol{B}_k\left[\boldsymbol{f}\left(\boldsymbol{x}_{k+1}\right)-\boldsymbol{f}\left(\boldsymbol{x}_k\right)\right]}\end{cases}$

4: $k=k+1$，计算误差 $\|\varepsilon^{(k)}\|_p=\|\boldsymbol{f}(\boldsymbol{x}_k)\|_p$；$\|\varepsilon\|_p=[\|\varepsilon\|_p,\|\varepsilon^{(k)}\|_p]$；$\|\varepsilon_{\boldsymbol{x}}^{(k)}\|_p=\|\boldsymbol{x}_{k+1}-\boldsymbol{x}_k\|_p$;

5: **if** $(\|\varepsilon^{(k)}\|_p<\|\varepsilon^*\|)\ \|\ (\|\varepsilon_{\boldsymbol{x}}^{(k)}\|_p<\|\varepsilon_{\boldsymbol{x}}^*\|)$ **then**

6: break;

7: **return** $\boldsymbol{x}^*=\boldsymbol{x}_k$，$\boldsymbol{x}_{iter}$，$\|\varepsilon\|_p$；

end

3.2 非线性方程组迭代算法代码的编写

通过对 Newton 法、修正 Newton 法、割线法和拟 Newton 法的整合，将代码保存集中在同一函数中，以对非线性方程组迭代求解，并命名为 “inleqs.m”。

```
function [Xstar, X_iter, errorFun] = inleqs(n, nlef, nlefd, method, options)
% ILEQS - Iterative optimization solution of Nonlinear equations by Newton
% methods or some variant method
%
% Input:
%   n        - Dimension of NLEQS
%   nlef     - n equations of nonlinear funcions
%   nlefd    - diff of nonlinear funcions
%   method   - Method of iteration: 'Newton', 'QNewton', 'MNewton', 'Secant'
%   options  - Options of algorithm(Struct data), include:
%            - options.p: use p-norm
%            - options.X0: Intial value of algorithm
%            - options.TolFun: tolerance of equation
%            - options.TolX: tolerance of solution
%            - options.Maxiter: Maximum number of iterations
%            - options.Display: If display the final iterations number
%            - options.PlotFcns: Draw the error value of each iteration
%                                during the execution of the algorithm
%            - options.Smethod: method for Secant
%
% Output:
%   Xstar    - Iterative solutions of equations
%   X_iter   - Iterative solutions in every iteration
%   errorFun - Error of linear equations in each iteration
%
% Usage:
%   [] = INLEQS(nlef, nlefd) uses the default settings soluting nonlinear equations
%   [] = INLEQS(nlef, nlefd, method) uses the input method to iterate NLEs
%   [] = INLEQS(nlef, nlefd, [], options) uses the options' settings soluting NLEs
%   Xstar = INLEQS( ... ) returns the iterative solution of NLEs
%   [~, X_iter] = ILEQS( ... ) returns the solutions in each iterations
%   [~, ~, error] = ILEQS( ... ) returns the error of nonlinear equations
%        in each iterations

% Author  : ZH.Yuan
% Update  : 2021/12/26 (First Version: 2021/12/26)
% Email   : zihaoyuan@whut.edu.cn (If any suggestions or questions)

% Set default value of value
if ~exist('method', 'var') || isempty(method)
    method = 'Newton';
end

% Set default value of options
```

```
if ~exist('options', 'var') || isempty(options)
    options.p = 2;
    options.X0 = zeros(n, 1);
    options.TolFun = 1e-04;
    options.TolX = 1e-04;
    options.Maxiter = max([1000 n]);
    options.Display = 0;
    options.omega = [];
    options.PlotFcns = 'off';
    options.leqsM = 0;
    if strcmp(method, 'Secant')
        options.Smethod = 'N';
    elseif strcmp(method, 'MNewton')
        options.s = 3;
    end
end

% Set default value of options.p
if ~isfield(options, 'p')
    options.p = 2;
elseif isempty(options.p)
    options.p = 2;
end

% Set default value of options.X0
if ~isfield(options, 'X0')
    options.X0 = zeros(n, 1);
elseif isempty(options.X0)
    options.X0 = zeros(n, 1);
end

% Set default value of options.TolFun
if ~isfield(options, 'TolFun')
    options.TolFun = 1e-04;
elseif isempty(options.TolFun)
    options.TolFun = 1e-04;
end

% Set default value of options.TolX
if ~isfield(options, 'TolX')
    options.TolX = 1e-04;
elseif isempty(options.TolX)
    options.TolX = 1e-04;
end

% Set default value of options.Maxiter
if ~isfield(options, 'Maxiter')
    options.Maxiter = max([1000 n]);
```

```
elseif isempty(options.Maxiter)
    options.Maxiter = max([1000 n]);
end

% Set default value of options.Display
if ~isfield(options, 'Display')
    options.Display = 0;
elseif isempty(options.Display)
    options.Display = 0;
end

% Set default value of options.PlotFcns
if ~isfield(options, 'PlotFcns')
    options.PlotFcns = 'off';
elseif isempty(options.PlotFcns)
    options.PlotFcns = 'off';
end

% Set default value of options.leqsM
if ~isfield(options, 'leqsM')
    options.leqsM = 0;
elseif isempty(options.leqsM)
    options.leqsM = 0;
end

% Set default value of options.Smethod for 'Secant' method
if strcmp(method, 'Secant')
    if ~isfield(options, 'Smethod')
        options.Smethod = 'N';
    elseif isempty(options.Smethod)
        options.Smethod = 'N';
    end
end

% Set default value of options.s for 'MNewton' method
if strcmp(method, 'MNewton')
    if ~isfield(options, 's')
        options.s = 3;
    elseif isempty(options.s)
        options.s = 3;
    end
end

iter = 0;
iter_m = 0;
X_old = reshape(options.X0, n, 1);
nlefd_m = nlefd(X_old);

```

```
while iter < options.Maxiter
    iter = iter + 1;

    switch method
        case 'Newton'
            if options.leqsM == 1
                DX = reshape(ileqs([nlefd(X_old), -nlef(X_old)]), n, 1);
            else
                DX = - (nlefd(X_old))^(-1) * nlef(X_old);
            end

        case 'MNewton'
            iter_m_new = floor(iter / options.s);
            if iter_m_new > iter_m
                nlefd_m = nlefd(X_old);
                iter_m = iter_m_new;
            end
            if options.leqsM == 1
                DX = reshape(ileqs([nlefd_m, -nlef(X_old)]), n, 1);
            else
                DX = - nlefd_m^(-1) * nlef(X_old);
            end

        case 'Secant'
            switch options.Smethod
                case 'Two'
                    if iter == 1
                        DX = 0.3 * randn(1, n);
                    end
                    DXNM = diag(DX);
                    XN = DXNM + X_old;
                    Ak = zeros(n, n);
                    for iAk = 1 : n
                        Ak(:, iAk) = (nlef(XN(:, iAk)) - nlef(X_old))/ DX(iAk);
                    end
                    if options.leqsM == 1
                        DX = reshape(ileqs([Ak, -nlef(X_old)]), n, 1);
                    else
                        DX = - Ak^(-1) * nlef(X_old);
                    end
                case 'N'
                    if iter == 1
                        P = eye(n) - diag(ones(1, n-1), 1);
                        XN = 0.2 * randn(n, n) + X_old;
                        Gammak = zeros(n, n);
                        for iGamma = 1 : n
                            Gammak(:, iGamma) = nlef(XN(:, iGamma)) - nlef(X_old);
                        end
```

```matlab
                        Hk = XN - X_old;
                        Hkbar = Hk * P;
                        Gammakbar = Gammak * P;
                    end
                    if options.leqsM == 1
                        DX = Hkbar * reshape(ileqs([Gammakbar, -nlef(X_old)]), n, 1);
                    else
                        DX = Hkbar * Gammakbar^(-1) * -nlef(X_old);
                    end
                    Hkbar = [-DX, Hkbar(:, 1 : end - 1)];
                    Gammakbar = [nlef(X_old) - nlef(X_old + DX), ...
                                Gammakbar(:, 1 : end - 1)];
            end

        case 'QNewton'
            if iter == 1
                Bk = nlefd(X_old)^(-1);
            end
            DX = - Bk * nlef(X_old);
            Bk = Bk - (Bk * nlef(X_old + DX) * DX' * Bk) / ...
                (DX' * Bk * (nlef(X_old + DX) - nlef(X_old)));

    end

    X_new = X_old + DX;
    errorX = (norm(DX, options.p))^(1/options.p);
    X_iter(:, iter) = X_new;
    errorFun(iter) = (norm(nlef(X_old), options.p))^(1/options.p);

    if errorFun(iter) <= options.TolFun || errorX <= options.TolX
        break
    end
    X_old = X_new;
end

if options.Display
    fprintf(['Algorithm-' mfilename ' stop at the %d-th iteration by ' ...
        method ' method.\n'], iter);
end

Xstar = X_new;

if strcmp(options.PlotFcns, 'on')
    plot(errorFun, 'LineWidth', 2)
    title('Error of equations in each iteration')
    xlabel('Iteration number')
    ylabel('Error of equations')
end
```

为了对迭代算法进行更为深入的研究和探析，本文建立一个线性和非线性函数随机组合生成的 n 元非线性方程组。该生成函数保存在”genNLE.m” 中。

```
function [nlef, nlefd, Xtrue] = genNLE(n)
% GENNLE - Generation of Nonlinear Functional Equations
% Input :
%    n          -  Dimensions * Functions (n * n)
%
% Output:
%    nlef     -  n equations of nonlinear funcions
%    nlefd    -  diff of nonlinear funcions
%    Xtrue    -  true root of groups
%
% Usage:
%    [nlef, nlefd, Xtrue] = GENNLE(n, nset, nconset)
%
% See also INLEQS, ILEQS

% Author  : ZH.Yuan
% Update  : 2021/12/26 (First Version: 2021/12/26)
% Email   : zihaoyuan@whut.edu.cn (If any suggestions or questions)

A = n * eye(n) + rand(n);
A = A' + A;

Xtrue = ones(n, 1) + rand(n, 1);
b = A * Xtrue;

k = randperm(14, 2);
[rNLEQ1, rNLEQD1] = TestFun(n, k(1));
[rNLEQ2, rNLEQD2] = TestFun(n, k(2));
nlef = @(x) [rNLEQ1(x) - rNLEQ1(Xtrue); rNLEQ2(x) - rNLEQ2(Xtrue); ...
    A(3 : end, :) * reshape(x, n, 1) - b(3 : end, :)];
nlefd = @(x) [rNLEQD1(x)'; rNLEQD2(x)'; A(3 : end, :)];
end

%% Generate Test Function
function [NLEQ, NLEQD] = TestFun(n, k)
% Select K n-ary nonlinear functions Randomly

if ~exist('k', 'var') || isempty(k)
    k = ceil(14 * rand);
end

A = [2; 2; zeros(n - 2, 1)] + rand(n, 1);

switch k
    case 1
```

```
            NLEQ = @(x) (sum(A .* reshape(x, n, 1)))^2;
            NLEQD = @(x) 2 * sum(A .* reshape(x, n, 1)) * A;
        case 2
            NLEQ = @(x) sum(A .* reshape(x, n, 1).^2);
            NLEQD = @(x) 2 * A .* reshape(x, n, 1);
        case 3
            NLEQ = @(x) log(sum(A .* reshape(x, n, 1)));
            NLEQD = @(x) A ./ sum(A .* reshape(x, n, 1));
        case 4
            NLEQ = @(x) (sum(A .* reshape(x, n, 1)))^(3/2);
            NLEQD = @(x) (3/2) * sum(A .* reshape(x, n, 1))^(1/2) * A;
        case 5
            NLEQ = @(x) sum(A .* reshape(x, n, 1).^(3/2));
            NLEQD = @(x) (3/2) * A .* reshape(x, n, 1).^(3/2);
        case 6
            NLEQ = @(x) (sum(A .* reshape(x, n, 1)))^3;
            NLEQD = @(x) 3 * sum(A .* reshape(x, n, 1))^2 * A;
        case 7
            NLEQ = @(x) (sum(A .* reshape(x, n, 1)))^(5/2);
            NLEQD = @(x) (5/2) * sum(A .* reshape(x, n, 1))^(3/2) * A;
        case 8
            NLEQ = @(x) (sum(A .* reshape(x, n, 1)))^4;
            NLEQD = @(x) 4 * sum(A .* reshape(x, n, 1))^3 * A;
        case 9
            NLEQ = @(x) sum(A .* reshape(x, n, 1).^3);
            NLEQD = @(x) 3 * A .* reshape(x, n, 1).^2;
        case 10
            NLEQ = @(x) sum(A .* reshape(x, n, 1).^(5/2));
            NLEQD = @(x) (5/2) * A .* reshape(x, n, 1).^(3/2);
        case 11
            NLEQ = @(x) sum(A .* reshape(x, n, 1).^3);
            NLEQD = @(x) 3 * A .* reshape(x, n, 1).^2;
        case 12
            NLEQ = @(x) sum(A .* reshape(x, n, 1).^4);
            NLEQD = @(x) 4 * A .* reshape(x, n, 1).^3;
        case 13
            NLEQ = @(x) sum(A .* reshape(x, n, 1).^(7/2));
            NLEQD = @(x) (7/2) * A .* reshape(x, n, 1).^(5/2);
        case 14
            NLEQ = @(x) (sum(A .* reshape(x, n, 1)))^(7/2);
            NLEQD = @(x) (7/2) * sum(A .* reshape(x, n, 1))^(5/2) * A;
end

end
```

第 4 章　实验结果

4.1　简例

设有非线性方程组 $\boldsymbol{f}(\boldsymbol{x})=\mathbf{0}$，其中：

$$\begin{cases} f_1(\boldsymbol{x})=(a_{11}*x_1+a_{12}*x_2+\ldots+a_{1n}*x_n)^2 \\ f_2(\boldsymbol{x})=a_{21}*x_1^2+a_{22}*x_2^2+\ldots+a_{2n}*x_n^2 \\ f_i(\boldsymbol{x})=\boldsymbol{A}_{i,\cdot}*\boldsymbol{x}-\boldsymbol{b}_i \quad i=3,4,\ldots,n \end{cases}$$

$$\boldsymbol{A}=\begin{pmatrix} 10 & 1 & \cdots & 1 & 1 \\ 1 & 10 & \cdots & 1 & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & 1 & \cdots & 10 & 1 \\ 1 & 1 & \cdots & 1 & 10 \end{pmatrix}_{n\times n} \qquad \boldsymbol{b}=\begin{pmatrix} (n+9)^2 \\ n+99 \\ \vdots \\ n+9 \\ n+9 \end{pmatrix}_n$$

很容易解析得到 $\boldsymbol{x}^*=(1,1,\cdots,1,1)$。若是用第三节的函数运行可以得到：

```matlab
clearvars; close all; clc;
n = 10;
A = ones(n) + (n - 1) * diag(ones(n, 1));
x0 = ones(n, 1);
f1 = @(x) sum(A(1, :) * reshape(x, numel(x), 1))^2;
f2 = @(x) A(2, :) * reshape(x, numel(x), 1).^2 ;
fx = @(x) [f1(x) - f1(x0) ; f2(x) - f2(x0) ;...
     A(3 : end, :) * (reshape(x, numel(x), 1) - x0)];
fxd = @(x) [2 * sum(A(1, :) * reshape(x, numel(x), 1)) * A(1, :); ...
     2 * A(2, :) * reshape(x, numel(x), 1) * A(2, :); A(3 : end, :)];

options.X0 = rand(n, 1);
options.PlotFcns = 'on';
options.Smethod = 'Two';
options.leqsM = 0;
options.s = 2;

subplot(2,2,1)
[Xstar1, X_iter1, errorFun1] = inleqs(n, fx, fxd, 'Newton', options);
subtitle('by Newton')
subplot(2,2,2)
[Xstar2, X_iter2, errorFun2] = inleqs(n, fx, fxd, 'MNewton', options);
subtitle('by MNewton')
subplot(2,2,3)
[Xstar3, X_iter3, errorFun3] = inleqs(n, fx, fxd, 'Secant', options);
subtitle('by Secant')
subplot(2,2,4)
[Xstar4, X_iter4, errorFun4] = inleqs(n, fx, fxd, 'QNewton', options);
subtitle('by QNewton')
```

结果如4.1所示

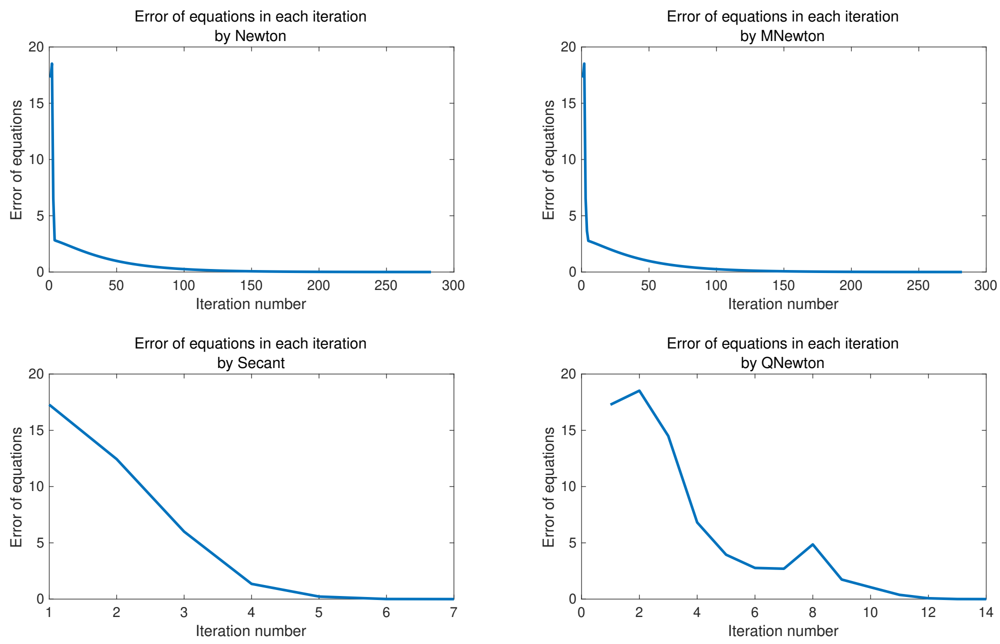

图 4.1: 小例运行结果图示

其中，Xstar_i 表示方程解的输出解；X_iter_i 表示每次迭代的迭代解，每一列为一次解；errorFun 表示线性方程组的残差向量的 p-范数值。这里将割线法迭代过程中的迭代解与迭代误差表示在表 4.1 中如下：

表 4.1: 割线法求解简例的迭代过程的解

迭代次数	迭代解 $\boldsymbol{x}$										残差范数
1	1.3745	1.2598	0.9627	0.9627	0.9627	0.9627	0.9627	0.9627	0.9627	0.9627	17.2774
2	1.0931	1.0678	0.9905	0.9905	0.9905	0.9905	0.9905	0.9905	0.9905	0.9905	12.4373
3	0.9952	0.9947	1.0006	1.0006	1.0006	1.0006	1.0006	1.0006	1.0006	1.0006	5.9962
4	1.0001	1.0003	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.3526
5	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.2256
6	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.0082
7	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	5.2762e-05

4.2 各方法收敛速度的可视化展示

函数“inleqs.m”的运行样例函数“inleqsDEMOsimple.m”运行结果展示如4.2所示，其中。图中命令行窗口展示了各方法（Newton、修正 Newton、割线法和拟 Newton 法）的迭代终止次数，工作区中为运行后的各方法结果：其中，Xstar1、Xstar2、Xstar3 和

Xstar4 分别表示各迭代算法的迭代解；errorall1、errorall2、errorall3 和 errorall4 分别表示各迭代算法每次的迭代误差；X_iter1、X_iter2、X_iter3 和 X_iter4 分别表示各迭代算法的所有迭代步骤过程中的向量值构成的矩阵。

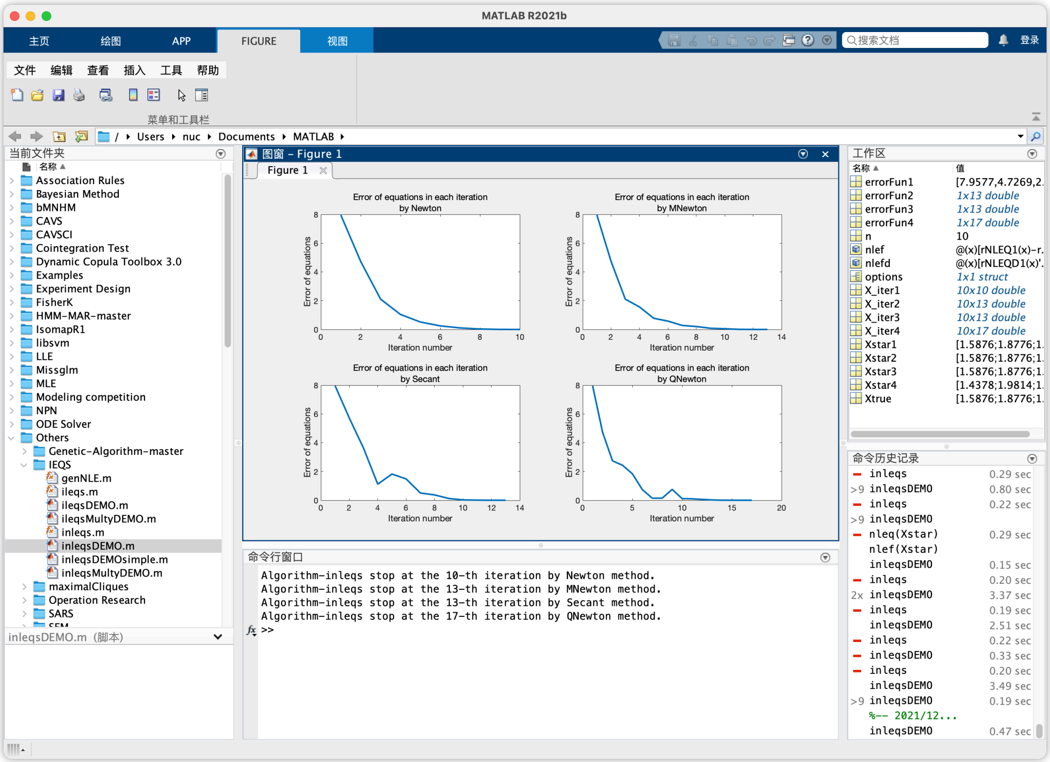

图 4.2: 运行结果图示

```matlab
%% inleqsDEMO
clearvars; close all; clc

n = 10;
options.X0 = ones(n, 1);
options.Display = 1;
options.PlotFcns = 'on';
options.Smethod = 'Two';
options.leqsM = 0;
options.s = 2;

[nlef, nlefd, Xtrue] = genNLE(n);

subplot(2,2,1)
[Xstar1, X_iter1, errorFun1] = inleqs(n, nlef, nlefd, 'Newton', options);
subtitle('by Newton')
subplot(2,2,2)
[Xstar2, X_iter2, errorFun2] = inleqs(n, nlef, nlefd, 'MNewton', options);
subtitle('by MNewton')
subplot(2,2,3)
```

```
[Xstar3, X_iter3, errorFun3] = inleqs(n, nlef, nlefd, 'Secant', options);
subtitle('by Secant')
subplot(2,2,4)
[Xstar4, X_iter4, errorFun4] = inleqs(n, nlef, nlefd, 'QNewton', options);
subtitle('by QNewton')
```

在阶数 N 分别取 10, 50, 100, 500 时，将 erroralll1、erroralll2、erroralll3 和 erroralll4 分别对应的 Newton、修正 Newton、割线法和拟 Newton 法的迭代算法每次的迭代误差随迭代次数的增加的趋势进行画图，所得结果如4.3所示。可以看出：该次模拟中，Newton、修正 Newton 的收敛性不如两点割线法和拟牛顿法稳定。最后各阶数下，各方法的误差都趋于 0，即各方法都收敛。

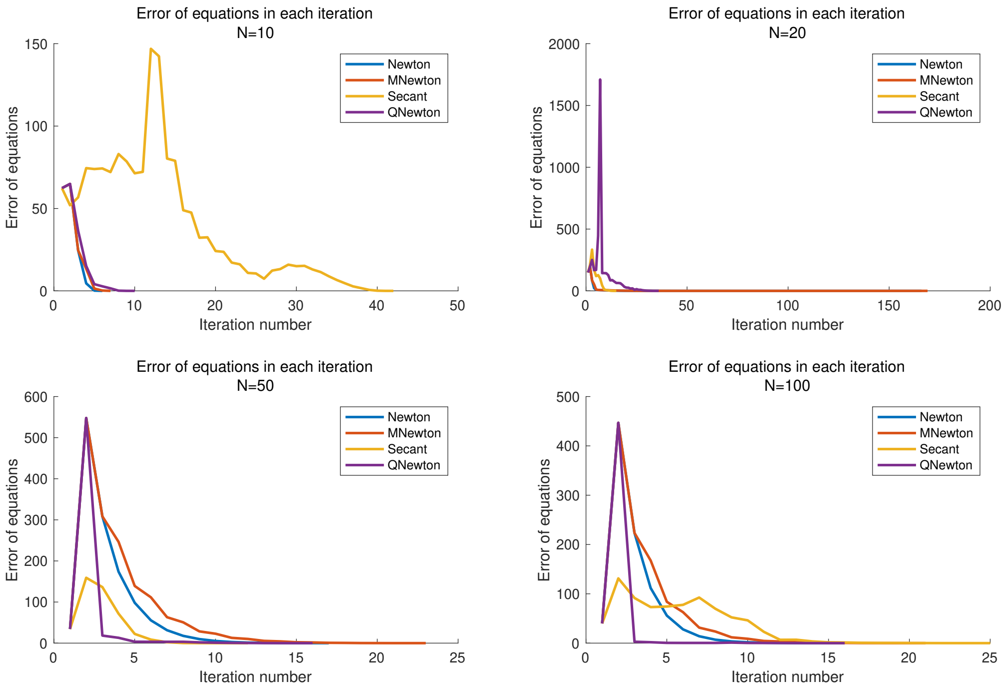

图 4.3: 200 次模拟实验的迭代次数的箱线图

为了更避免模拟实验的随机性，本文又在阶数 N 分别取 10，20，50，100 时，对 Newton、修正 Newton、割线法和拟 Newton 法的迭代算法进行了 200 次的模拟测试，测试结果取每个方法每次最终的迭代终止时的迭代次数。最终，将 200 次的迭代次数以箱线图4.4的形式进行展示。

当然，更多的参数可以进行设置与调整，详细的参数设置可以参照函数文件“inleqs.m”的 Input 注释进行阅读后调整，这里不再给出过多的说明。图 4.3 及图 4.4 的详细模拟过程见 GitHub[2]中的“inleqsDEMO0.m”、“inleqsDEMO.m”和“inleqsMulty-DEMO.m”。

[2] https://github.com/yuanzihao945/IEQS

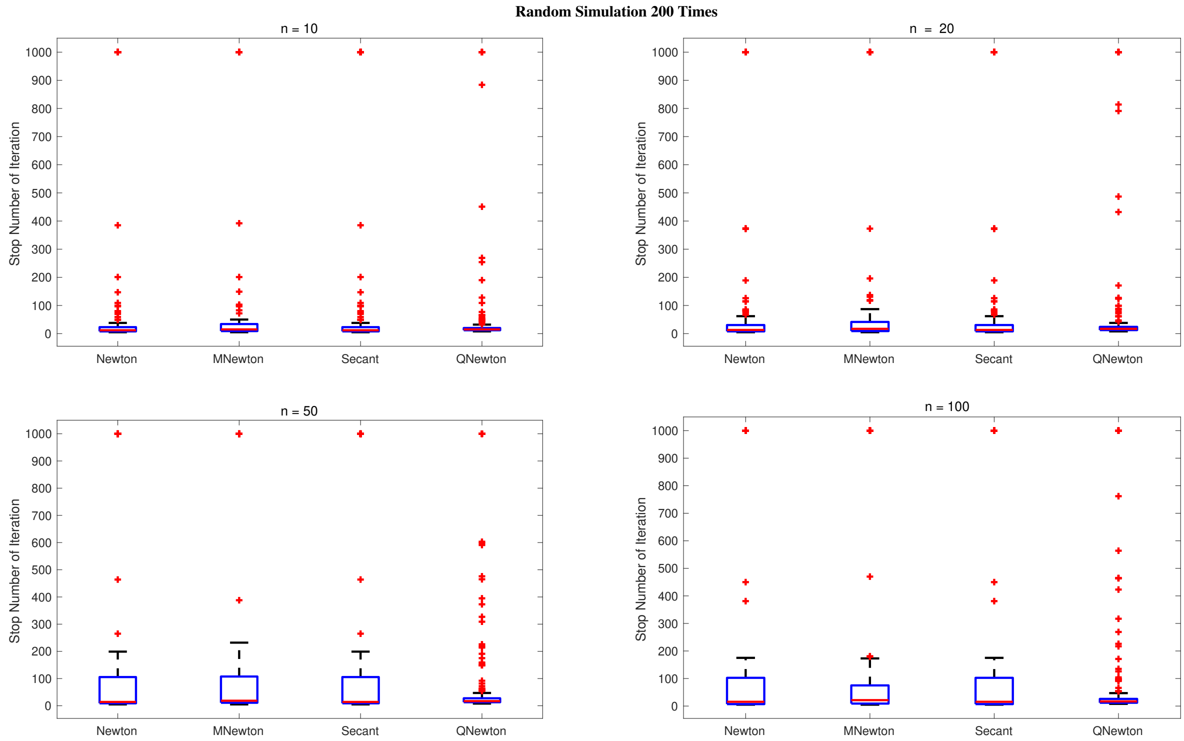

图 4.4: 200 次模拟实验的迭代次数的箱线图

4.3 总结展望

本文在 MATLAB_R2021b 环境下，编写了 Newton、修正 Newton、割线法和拟 Newton 法的迭代算法的迭代矩阵的表达函数，利用迭代矩阵和表达式对非线性方程组进行求解，已验证该迭代思想的正确性与局部收敛性。通过数值模拟分析，验证了其迭代收敛结果的正确性，并通过对误差的分析，确保了其收敛的局部稳定性。

致谢

感谢朱国甫老师对本实验报告的指导与鼓励！朱老师为我指点迷津，帮助我开拓研究思路，精心点拨、热忱鼓励。朱老师严谨细致、一丝不苟的作风，认真教学的态度，踏踏实实的精神让我深受感动。虽历时仅半年，却给我以终生受益无穷之道。对朱老师的感激之情是无法用言语表达的。