# 实验一 简单迭代法
# 与 *Steffensen* 加速迭代法
# 实验报告

| | |
|---|---|
| 课　　程 | 计算方法与 MATLAB |
| 院（系） | 计算机科学与技术学院 |
| 专　　业 | 人工智能 |
| 级　　别 | 2021 级 |
| 学　　号 | 2115102001 |
| 姓　　名 | 安炳旭 |
| 指导老师 | 陈叶旺 |

2023 年 11 月 24 日

# 目 录

# 实验一　简单迭代法与 *Steffensen* 加速迭代法

## 1　算法思想

### 1.1　简单迭代法简介

设方程$f(x)=0$, 为求方程实根，将方程写作如下形式：

$$x=\varphi(x) \tag{1}$$

则$x^*=\varphi(x^*)$即为方程的根。现如今可取方程根的初始近似值$x_0$, 作迭代过程如下式：

$$x_{n+1}=\varphi(x_n) \quad (n=0,1,2,...) \tag{2}$$

若(2)式产生的序列$\{x_n\}$收敛，则说明$\varphi(x)$选取恰当，将这种迭代方式称作简单迭代法。可见，序列的收敛性取决于$\varphi(x)$的选取，根据学习的相关知识可知，当$\varphi(x)$满足压缩映像原理时，产生的序列$\{x_n\}$一定会收敛到$x^*$. 压缩映像原理如下所示：

**定理 1 压缩映像原理**　设$\varphi(x)$在闭区间$[a,b]$上满足：

1. $\forall x\in[a,b], \varphi(x)\in[a,b]$
2. $\forall x_1,x_2\in[a,b],\ \exists L\in(0,1), |\varphi(x_1)-\varphi(x_2)|\le L|x_1-x_2|$

简单迭代法的几何解释如图 1 所示，方程$x=\varphi(x)$的解$x^*$是直线$y=x$与曲线$y=\varphi(x)$交点的横坐标，迭代过程是收敛的

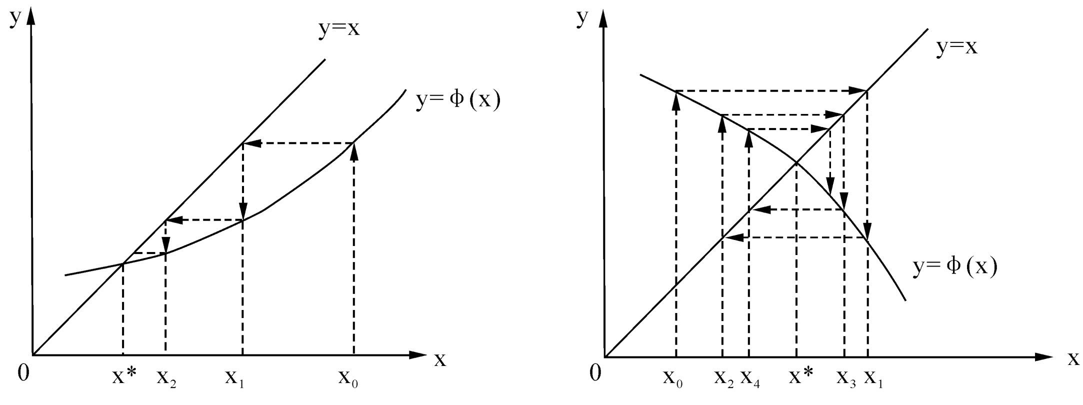


图 1 简单迭代法的几何解释

### 1.2 *Steffensen* 迭代法简介

设方程$f(x)=0$, 对于 *Steffensen* 迭代法，格式为

$$\begin{cases} y_n=\varphi(x_n) \\ z_n=\varphi(y_n) \\ x_{n+1}=x_n-\dfrac{(y_n-x_n)^2}{z_n-2y_n+x_n}\ (n=0,1,2,...) \end{cases} \tag{3}$$

对于 *Steffensen* 迭代法而言，可以证明其确定的序列至少以平方收敛到$x^*$.

*Steffensen* 迭代法的几何解释如图 2 所示

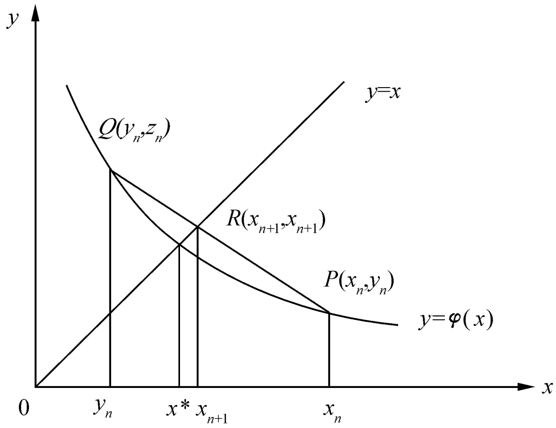


图 2 Steffensen 迭代法的几何解释

## 2 算法步骤

本次实验需要完成两个问题：

1. 使用简单迭代法和 *Steffensen* 加速迭代法求解方程$x^5-3x-1=0$的根，要求精度($\varepsilon=\dfrac{1}{2}\times10^{-5}$)
2. 使用简单迭代法和 *Steffensen* 加速迭代法求解方程$x=1+\dfrac{1}{2}sinx$的根，要求精度($\varepsilon=\dfrac{1}{2}\times10^{-5}$)

首先对这两个问题进行初步分析，对于问题 1 而言，需要先将方程整理成$x=\varphi(x)$的形式，即$x=(3x+1)^{\frac{1}{5}}$, 此时$\varphi(x)$满足压缩映像条件，说明产生的序列必定是收敛的。而对于问题 2 来说，方程$x=1+\dfrac{1}{2}sinx$已经整理成了$x=\varphi(x)$的形式。

初步分析，问题 1 的解在 1.4 到 1.5 之间，问题 2 的解在 1.4 到 1.5 之间，接下来进行 MATLAB 代码的编写。

# 3 代码实现

本次实验代码及其功能如下表所示，具体功能请参考注释：

**Simple_Interation_Method.m**

功能：使用简单迭代法进行迭代

```
function result = Simple_Interation_Method(phi, initial_guess, tolerance,
max_iterations)
    % 初始化 x_0
    x = initial_guess;
    % 迭代次数，初始为 1
    iteration = 1;

    while iteration <= max_iterations
        % 使用迭代函数得到新的值
        x_new = phi(x);
        % 计算当前迭代的误差
        error = abs(x_new - x);
        % 输出当前迭代的结果
        fprintf('第 %d 次迭代: x = %f\n', iteration, x_new);

        % 如果误差小于阈值，达到所需精度
        if error < tolerance
            fprintf('在 %d 次迭代后达到所需的精度 (%e)。\n', iteration,
tolerance);
            result = x_new;
            return;
        end

        % 更新当前值
        x = x_new;
        % 增加迭代次数
        iteration = iteration + 1;
    end

    % 输出在最大迭代次数内未收敛的消息
    disp('在最大迭代次数内未收敛。');
    result = [];
end
```

| **steffensen_iteration.m** |
|---|
| 功能：使用 steffensen 加速迭代法进行迭代 |

```
function result = steffensen_iteration(phi, initial_guess, tolerance,
max_iterations)
    % 初始化 x_0
    x = initial_guess;
    % 迭代次数，初始为 1
    iteration = 1;

    while iteration < max_iterations
        % 使用迭代函数计算 y 和 z
        y = phi(x);
        z = phi(y);

        % 防止分母为零
        if z - 2*y + x == 0
            disp('分母为零。无法继续迭代。');
            result = [];
            return;
        end

        % 使用 Steffensen 加速迭代公式得到新的值
        x_new = x - ((y - x)^2) / (z - 2*y + x);

        % 计算当前迭代的误差
        error = abs(x_new - x);
        % 输出当前迭代的结果
        fprintf('第 %d 次迭代: x = %f\n', iteration, x_new);

        % 如果误差小于阈值，达到所需精度
        if error < tolerance
            fprintf('在 %d 次迭代后达到所需的精度 (%e)。\n', iteration,
tolerance);
            result = x_new;
            return;
        end

        % 更新当前值
        x = x_new;
        % 增加迭代次数
        iteration = iteration + 1;
    end
```

```
    % 输出在最大迭代次数内未收敛的消息
    disp('在最大迭代次数内未收敛。');
    result = [];
end
```

**phi1_function.m**

功能：问题 1 的函数

```
function y = phi1_function(x)
    y = (3*x + 1)^(1/5);
end
```

**phi2_function.m**

功能：问题 2 的函数

```
function y = phi2_function(x)
    y = 1 + (1/2) * sin(x);
end
```

**main.m**

功能：main 函数，用于调用上述函数并输出结果

```
clc;clear;
initial_guess =10;
tolerance = 0.5 * 1e-5;
max_iterations = 1000;

disp(['使用简单迭代法对函数 x=(3x+1)^(1/5) 进行迭代']);
result1 = Simple_Interation_Method(@phi1_function, initial_guess,
tolerance, max_iterations);
disp(['近似解为: ', num2str(result1)]);

disp(['使用 Steffensen 加速迭代法对函数 x=(3x+1)^(1/5) 进行迭代']);
result2 = steffensen_iteration(@phi1_function, initial_guess, tolerance,
max_iterations);
disp(['近似解为: ', num2str(result2)]);

disp(['使用简单迭代法对函数 x=(1/2)sin(x) 进行迭代']);
result3 = Simple_Interation_Method(@phi2_function, initial_guess,
tolerance, max_iterations);
disp(['近似解为: ', num2str(result3)]);
```

```
disp(['使用 Steffensen 加速迭代法对函数 x=(1/2)sin(x) 进行迭代']);
result4 = steffensen_iteration(@phi2_function, initial_guess, tolerance,
max_iterations);
disp(['近似解为: ', num2str(result4)]);
```

# 4 实验结果

对于问题 1，选取$x_0=1.5$，分别使用简单迭代法和 *Steffensen* 迭代法所得到的迭代过程与结果如表 1 所示：

表 1 问题 1 在$x_0=1.5$的情况下的迭代结果

| |
|---|
| 使用简单迭代法对函数 x=(3x+1)^(1/5) 进行迭代 |
| 第 1 次迭代：x = 1.406282 |
| 第 2 次迭代：x = 1.391602 |
| 第 3 次迭代：x = 1.389245 |
| 第 4 次迭代：x = 1.388865 |
| 第 5 次迭代：x = 1.388804 |
| 第 6 次迭代：x = 1.388794 |
| 第 7 次迭代：x = 1.388792 |
| 在 7 次迭代后达到所需的精度 (5.000000e-06)。 |
| 近似解为: 1.3888 |
| 使用 Steffensen 加速迭代法对函数 x=(3x+1)^(1/5) 进行迭代 |
| 第 1 次迭代：x = 1.388875 |
| 第 2 次迭代：x = 1.388792 |
| 第 3 次迭代：x = 1.388792 |
| 在 3 次迭代后达到所需的精度 (5.000000e-06)。 |
| 近似解为: 1.3888 |

同样地，将$x_0=1.5$应用于问题二，使用简单迭代法和 *Steffensen* 迭代法所得到的迭代过程与结果如表 2 所示：

表 2 问题 2 在$x_0=1.5$的情况下的迭代结果

| |
|---|
| 使用简单迭代法对函数 x=(1/2)sin(x) 进行迭代 |

```
第 1 次迭代: x = 1.498747
第 2 次迭代: x = 1.498703
第 3 次迭代: x = 1.498701
在 3 次迭代后达到所需的精度 (5.000000e-06)。
近似解为: 1.4987
使用 Steffensen 加速迭代法对函数 x=(1/2)sin(x) 进行迭代
第 1 次迭代: x = 1.498701
第 2 次迭代: x = 1.498701
在 2 次迭代后达到所需的精度 (5.000000e-06)。
近似解为: 1.4987
```

# 5 实验结论

为了进一步验证简单迭代法和 Steffensen 加速迭代法的收敛速度，本实验继续对$x_0$的取值进行修改，分别选取$x_0=100,1000$进行实验。得到的结果如表 3-表 6 所示：

表 3 问题 1 在$x_0=100$的情况下的迭代结果

```
使用简单迭代法对函数 x=(3x+1)^(1/5) 进行迭代
第 1 次迭代: x = 3.131218
第 2 次迭代: x = 1.597179
第 3 次迭代: x = 1.420885
第 4 次迭代: x = 1.393930
第 5 次迭代: x = 1.389620
第 6 次迭代: x = 1.388925
第 7 次迭代: x = 1.388814
第 8 次迭代: x = 1.388795
第 9 次迭代: x = 1.388793
在 9 次迭代后达到所需的精度 (5.000000e-06)。
近似解为: 1.3888
```

| |
|---|
| 使用 Steffensen 加速迭代法对函数 x=(3x+1)^(1/5) 进行迭代<br>第 1 次迭代：x = 1.572495<br>第 2 次迭代：x = 1.389007<br>第 3 次迭代：x = 1.388792<br>第 4 次迭代：x = 1.388792<br>在 4 次迭代后达到所需的精度 (5.000000e-06)。<br>近似解为: 1.3888 |

表 4 问题 2 在$x_0=100$的情况下的迭代结果

| |
|---|
| 使用简单迭代法对函数 x=(1/2)sin(x) 进行迭代<br>第 1 次迭代：x = 0.746817<br>第 2 次迭代：x = 1.339653<br>第 3 次迭代：x = 1.486703<br>第 4 次迭代：x = 1.498233<br>第 5 次迭代：x = 1.498684<br>第 6 次迭代：x = 1.498701<br>第 7 次迭代：x = 1.498701<br>在 7 次迭代后达到所需的精度 (5.000000e-06)。<br>近似解为: 1.4987<br>使用 Steffensen 加速迭代法对函数 x=(1/2)sin(x) 进行迭代<br>第 1 次迭代：x = 1.336133<br>第 2 次迭代：x = 1.499244<br>第 3 次迭代：x = 1.498701<br>第 4 次迭代：x = 1.498701<br>在 4 次迭代后达到所需的精度 (5.000000e-06)。<br>近似解为: 1.4987 |

表 5 问题 1 在$x_0=1000$的情况下的迭代结果

| |
|---|
| 使用简单迭代法对函数 x=(3x+1)^(1/5) 进行迭代<br>第 1 次迭代：x = 4.959675<br>第 2 次迭代：x = 1.738460 |

第 3 次迭代：x = 1.441098

第 4 次迭代：x = 1.397128

第 5 次迭代：x = 1.390134

第 6 次迭代：x = 1.389008

第 7 次迭代：x = 1.388827

第 8 次迭代：x = 1.388798

第 9 次迭代：x = 1.388793

在 9 次迭代后达到所需的精度 (5.000000e-06)。

近似解为: 1.3888

使用 Steffensen 加速迭代法对函数 x=(3x+1)^(1/5) 进行迭代

第 1 次迭代：x = 1.727998

第 2 次迭代：x = 1.389460

第 3 次迭代：x = 1.388792

第 4 次迭代：x = 1.388792

在 4 次迭代后达到所需的精度 (5.000000e-06)。

近似解为: 1.3888

表 6 问题 2 在$x_0=1000$的情况下的迭代结果

使用简单迭代法对函数 x=(1/2)sin(x) 进行迭代

第 1 次迭代：x = 1.413440

第 2 次迭代：x = 1.493822

第 3 次迭代：x = 1.498519

第 4 次迭代：x = 1.498695

第 5 次迭代：x = 1.498701

第 6 次迭代：x = 1.498701

在 6 次迭代后达到所需的精度 (5.000000e-06)。

近似解为: 1.4987

使用 Steffensen 加速迭代法对函数 x=(1/2)sin(x) 进行迭代

第 1 次迭代：x = 1.493816

第 2 次迭代：x = 1.498701

```
第 3 次迭代: x = 1.498701
在 3 次迭代后达到所需的精度 (5.000000e-06)。
近似解为: 1.4987
```

经过实验结果所示，问题 1 在$x_0=100$和$x_0=1000$时使用简单迭代法和 *Steffensen* 迭代法都需要 9 步和 4 步; 问题 2 在$x_0=100$和$x_0=1000$时使用简单迭代法和 *Steffensen* 迭代法需要 7 步和 4 步.

因此可以得出结论：**简单迭代法的迭代速度往往比 *Steffensen* 迭代法速度更慢，同时迭代速度的快慢取决于初始点$x_0$的取值以及$\varphi(x)$，除此之外，也与精度$\varepsilon$有关.**

最后，本实验总结了简单迭代法和 *Steffensen* 迭代法的优缺点如下所示：

- 简单迭代法

优点：

- 简单迭代法是一种直观、易于理解和实现的迭代方法，通常只需要基本的代数运算即可完成。
- 对于一些简单的迭代问题，特别是在具有较好收敛性的情况下，简单迭代法可以取得较好的效果。

缺点：

- **简单迭代法的主要缺点是收敛速度相对较慢，特别是在迭代问题的收敛性较差的情况下，可能需要较多的迭代步骤才能达到精度要求。**

- *Steffensen* 迭代法

优点：

- *Steffensen* 加速迭代法是一种用于提高收敛速度的方法，通过引入二阶收敛的思想，加速了原始迭代法的收敛。

缺点：

- 不保证总是收敛，需要事先判断是否满足压缩映像原理，因此需要仔细选择初始估计值和迭代准则，以保证算法的可靠性和有效性。