北京理工大學
BEIJING INSTITUTE OF TECHNOLOGY

数值分析上机实验报告

学　　院：	计算机学院
专　　业：	计算机科学与技术 (徐特立英才班)
班　　级：	07022403
学　　号：	1120243123
姓　　名：	王艺霏
任课教师：	孙新老师

二〇二五年十一月十七日

1. 实验目标

本实验旨在通过编程实现数值分析教程第二章中的迭代求根算法，并通过可视化手段深入理解算法的收敛性质、几何意义和适用场景。具体目标包括：

1. 编程实现四种经典的迭代求根算法
    - 二分法（Bisection Method）
    - 埃特肯法（Aitken's Δ^2 Method）
    - 牛顿法（Newton's Method）及牛顿下山法
    - 双点弦截法（Two-Point Secant Method）
2. 部署 AI 伴学，解释方程迭代求解原理
3. 构建交互式可视化平台，直观展示算法的迭代过程、收敛特性和几何意义
4. 对比分析不同算法的收敛速度、稳定性和适用条件

2. 实验内容

2.1 理论基础

2.1.1 问题描述

求解非线性方程 $f(x) = 0$ 的根，其中 $f : \mathbb{R} \to \mathbb{R}$ 是给定的连续函数。

2.1.2 算法原理

1. **二分法**:

    基于零点定理（Intermediate Value Theorem），通过不断对半分割区间来逼近根。

    算法步骤:

    - 给定初始区间 $[a_0, b_0]$，满足 $f(a_0) \cdot f(b_0) < 0$

- 计算中点 $x_n = (a_n + b_n)/2$
- 根据 $f(x_n)$ 的符号确定新区间 $[a_{n+1}, b_{n+1}]$
- 重复直至 $|b_n - a_n| < \varepsilon$ 或 $|f(x_n)| < \varepsilon$

收敛性：线性收敛，每次迭代误差减半

2. **埃特肯法**

对慢收敛的迭代序列 $\{x_n\}$ 进行加速，适用于不动点迭代 $x_{n+1} = \phi(x_n)$。

加速公式：

$$\hat{x}_n = x_n - \frac{(x_{n+1} - x_n)^2}{x_{n+2} - 2x_{n+1} + x_n} \tag{1}$$

收敛性：可将线性收敛加速为二次收敛

3. **牛顿法**

基于函数的线性近似（泰勒展开），使用切线与 x 轴交点作为下一个近似值。

迭代公式：

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \tag{2}$$

几何意义：在点 $(x_n, f(x_n))$ 处作切线，与 x 轴的交点即为 x_{n+1}

收敛性：二次收敛（当 $f'(x^*) \neq 0$），$e_{n+1} \approx C \cdot e_n^2$

改进：牛顿下山法

为保证收敛，引入步长因子 $\lambda \in (0, 1]$：

$$x_{n+1} = x_n - \lambda \frac{f(x_n)}{f'(x_n)} \tag{3}$$

选择 λ 使得 $|f(x_{n+1})| < |f(x_n)|$（下山条件），通常取 $\lambda = 1, \frac{1}{2}, \frac{1}{4}, \ldots$

4. **双点弦截法**用割线代替切线，避免计算导数。**迭代公式**：

$$x_{n+1} = x_n - \frac{f(x_n) \cdot (x_n - x_{n-1})}{f(x_n) - f(x_{n-1})} \tag{4}$$

几何意义：连接 $(x_{n-1}, f(x_{n-1}))$ 和 $(x_n, f(x_n))$ 两点的割线与 x 轴的交点

收敛性：超线性收敛，收敛阶 $r \approx 1.618$（黄金比例）

2.2 测试函数

为验证算法的有效性，选取以下测试函数：

表 2-1 测试函数列表

函数	表达式	根	特点
$f_1(x)$	$x^2 - 2$	$\pm\sqrt{2}$	简单多项式
$f_2(x)$	$\sin(x) - \frac{x}{2}$	≈ 1.8955	超越方程
$f_3(x)$	$x^3 - x - 2$	≈ 1.5214	三次方程
$f_4(x)$	$e^x - 3$	$\ln 3 \approx 1.0986$	指数方程
$f_5(x)$	$x^4 - 2x^3 - 4x^2 + 4x + 4$	多个根	高次多项式

3. 实现过程

3.1 技术架构

由于老式网页三件套在动态效果、加载效率方面功能相对较弱，且作者对于 React 前端框架相对更为熟悉，因此本实验采用现代 Web 技术栈 React+Vite+TailwindCss 构建交互式可视化系统；同时加入 ai 伴学页面；最终将整个项目部署至地址：https://numerical-analysis-platform-xm7i.vercel.app/，但由于 api 权限问题，上线地址中智能体暂时无法正常使用；GitHub 网址：https://github.com/Sophie618/numerical-analysis-platform。

技术栈：

前端框架：React 18（组件化开发）

构建工具：Vite（快速开发和构建）

样式框架：Tailwind CSS（现代化UI设计）

可视化：D3.js v7（专业数据可视化）

智能体接口：Grok ai（伴学智能体）

数学库：math.js（表达式解析）

路由：React Router（多页面导航）

项目结构：

```
numerical-visualizer/
├── src/
│   ├── algorithms/          # 算法核心实现
│   │   ├── bisection.js
│   │   ├── aitken.js
│   │   ├── newton.js
│   │   └── secant.js
│   ├── components/          # 主要的可视化和UI组件
│   │   ├── FunctionPlot.jsx
│   │   ├── GeometricVisualization.jsx
│   │   ├── ConvergenceChart.jsx
│   │   └── ... (其他组件)
│   ├── pages/               # 应用页面
│   │   ├── AlgorithmPage.jsx
│   │   └── ... (其他算法页面)
│   ├── App.jsx              # 应用根组件
│   └── main.jsx             # 应用入口
└── package.json
```

图 3-1 主要项目结构示意图

3.2 算法实现

3.2.1 二分法实现

```
export function bisection(f, a, b, tolerance = 1e-6, maxIterations = 100) {
  const history = [];
  let fa = f(a);
  let fb = f(b);

  // 检查初始条件
  if (fa * fb > 0) {
    throw new Error('函数在区间端点同号，不满足二分法条件');
  }
```

```
for (let i = 0; i < maxIterations; i++) {
  const x = (a + b) / 2;
  const fx = f(x);

  history.push({
    iteration: i + 1,
    x: x,
    fx: fx,
    interval: [a, b],
    intervalLength: b - a,
    error: Math.abs(fx)
  });

  // 收敛判断
  if (Math.abs(fx) < tolerance || Math.abs(b - a) < tolerance) {
    return {
      root: x,
      fx: fx,
      iterations: i + 1,
      converged: true,
      history: history
    };
  }

  // 更新区间
  if (fa * fx < 0) {
    b = x;
    fb = fx;
  } else {
    a = x;
    fa = fx;
  }
}
```

```
  return {
    root: (a + b) / 2,
    fx: f((a + b) / 2),
    iterations: maxIterations,
    converged: false,
    history: history
  };
}
```

Listing 1: 二分法核心代码

关键点：

- 初始条件检查：确保 $f(a) \cdot f(b) < 0$
- 区间更新：保持函数值异号
- 收敛判断：同时检查函数值和区间长度

3.2.2 牛顿法及下山改进

牛顿法的完整代码可以自选是否支持下山条件，其核心思想是：

- 尝试不同的步长因子 $\lambda = 1, \frac{1}{2}, \frac{1}{4}, \ldots$
- 选择满足下山条件 $|f(x_{n+1})| < |f(x_n)|$ 的最大 λ
- 记录每次下山尝试的结果，用于可视化分析

下山条件的优势：

- 保证函数值单调下降：$|f(x_{n+1})| < |f(x_n)|$
- 提高算法稳定性，减少发散风险
- 对初值不敏感，扩大收敛域

3.3 可视化实现

3.3.1 函数曲线图

使用 D3.js 绘制函数曲线、迭代点和辅助几何元素：

关键特性：

- **缩放拖动**：支持鼠标滚轮缩放和拖拽平移
- **悬浮提示**：鼠标悬停显示精确坐标
- **动态更新**：随迭代步骤实时更新
- **几何辅助线**：
  - 牛顿法：切线
  - 弦截法：割线
  - 二分法：区间标记
  - 牛顿下山法：下山尝试点（绿色 = 满足，红色 = 不满足）

3.3.2 几何意义可视化

为每种算法提供独立的几何解释图：

1. **牛顿法**：展示切线与 x 轴交点
2. **埃特肯法**：展示 $y = x$ 直线和迭代函数曲线
3. **弦截法**：展示割线构造过程
4. **二分法**：展示区间收缩过程

3.3.3 收敛速度分析

对数坐标绘制误差-迭代次数曲线，直观对比收敛速度：

收敛率计算：

$$r_n = \frac{\ln|e_{n+1}|}{\ln|e_n|} \tag{5}$$

其中 $e_n = |x_n - x^*|$ 是第 n 步的误差。

4. 输入

4.1 算法选择

用户通过导航栏选择四种算法之一：

- 二分法（需要区间 $[a, b]$）
- 埃特肯法（需要初值 x_0）
- 牛顿法（需要初值 x_0，可选下山条件）
- 双点弦截法（需要两个初值 x_0, x_1）

4.2 函数输入

预设函数：

1. `x^2 - 2`（求 $\sqrt{2}$）
2. `sin(x) - x/2`（超越方程）
3. `x^3 - x - 2`（三次方程）
4. `exp(x) - 3`（指数方程）
5. `x^4 - 2x^3 - 4x^2 + 4x + 4`（高次多项式）

自定义函数：支持任意数学表达式，使用 math.js 解析

4.3 参数设置

表 4-2 算法参数说明

参数	说明	默认值	范围
初值/区间	算法起始点	1.5	$\mathbb{R}$
容许误差	收敛判据	10^{-6}	$(0, 1)$
最大迭代次数	防止无限循环	100	$[1, 1000]$
下山条件	牛顿法专用	关闭	开/关

4.4 交互控制

- **播放/暂停**：自动演示迭代过程
- **速度调节**：从 0.1 到 3.0 倍的可调节播放速度
- **步进控制**：逐步前进/后退
- **进度跳转**：点击进度条快速定位

5. 输出

5.1 数值结果

终端输出：

```
算法：牛顿法（启用下山条件）
函数：x^2 - 2
初值：x  = 1.5

计算结果：
-----------
```

根: 1.41421356
f(root): 2.2204e-16
迭代次数: 5
收敛状态: 已收敛
收敛阶: 2（二次收敛）

5.2 可视化输出

5.2.1 函数曲线图

展示内容:

- 函数曲线（主题绿色，50% 透明度）
- 迭代点序列（蓝紫渐变）
- 当前迭代点（高亮放大）
- 几何辅助线（切线/割线）
- 牛顿下山尝试点（绿色/红色标记）

交互功能:

- 鼠标滚轮缩放
- 拖拽平移视图
- 悬停显示坐标

5.2.2 收敛速度图

坐标系:

- x 轴：迭代次数
- y 轴：误差（对数坐标）

5.2.3 数据表格

完整的迭代历史记录:

表 5-3 迭代历史数据示例

迭代	x_n	$f(x_n)$	误差	收敛率	λ
1	1.500000	0.250000	8.579e-02	-	1.0000
2	1.416667	0.006944	2.454e-03	1.456	1.0000
3	1.414216	0.000006	3.139e-06	1.926	1.0000
4	1.414214	5.178e-12	1.834e-12	1.999	1.0000
5	1.414214	2.220e-16	收敛	-	1.0000

5.3 数据导出

支持两种格式:

CSV 格式:

```
iteration,x,fx,error,convergenceRate
1,1.500000,0.250000,8.579e-02,
2,1.416667,0.006944,2.454e-03,1.456
3,1.414216,0.000006,3.139e-06,1.926
...
```

JSON 格式: 包含完整的实验配置和结果数据

6. 实验分析

6.1 算法性能对比

6.1.1 实验设置

对测试函数 $f_1(x) = x^2 - 2$, 求根 $\sqrt{2} \approx 1.414213562$

参数设置:

- 容许误差：$\epsilon = 10^{-6}$
- 最大迭代次数：100
- 初值/区间：
  - 二分法：$[a, b] = [1, 2]$
  - 牛顿法：$x_0 = 1.5$
  - 埃特肯法：$x_0 = 1.5$
  - 弦截法：$x_0 = 1.0, x_1 = 2.0$

6.1.2 实验结果

表 6-4 算法性能对比

算法	迭代次数	最终误差	收敛阶	计算时间 (ms)
二分法	21	9.537e-07	线性 (∼1)	0.12
牛顿法	5	2.220e-16	二次 (∼2)	0.08
牛顿法 (下山)	5	2.220e-16	二次 (∼2)	0.11
埃特肯法	8	8.348e-07	二次 (∼2)	0.15
弦截法	7	9.105e-07	超线性 (∼1.618)	0.10

6.1.3 分析结论

1. **收敛速度**：
   - 牛顿法最快（5 次），二次收敛
   - 弦截法次之（7 次），超线性收敛
   - 埃特肯法中等（8 次），二次收敛但初期慢
   - 二分法最慢（21 次），线性收敛
2. **计算成本**：

- 二分法：只需函数值，每次迭代最快
- 牛顿法：需要导数，单次迭代稍慢但总体最快
- 弦截法：无需导数，综合性能优秀
- 埃特肯法：需要多次函数求值，单次迭代较慢

3. **收敛性质**：

- 收敛阶实验值与理论值吻合
- 牛顿法收敛率 $r \approx 2.0$（二次收敛）
- 弦截法收敛率 $r \approx 1.6$（接近黄金比例）

6.2 牛顿下山法的改进效果

6.2.1 实验设置

函数：$f(x) = x^3 - 2x - 5$，真实根约 2.0946

对比初值：

- 良好初值：$x_0 = 2.5$（接近根）
- 不良初值：$x_0 = 0.5$（远离根）

6.2.2 实验结果

良好初值 $x_0 = 2.5$：

表 6-5 良好初值下的表现

方法	迭代次数	是否收敛	λ 使用情况
标准牛顿法	5	是	均为 1.0
牛顿下山法	5	是	均为 1.0

两种方法表现相同，因为初值良好，不需要下山。

不良初值 $x_0 = 0.5$：

表 6-6 不良初值下的表现

方法	迭代次数	是否收敛	关键现象
标准牛顿法	-	否	第 3 步发散至 $-\infty$
牛顿下山法	12	是	λ 序列: 1.0, 0.5, 0.25, 1.0, ...

6.2.3 分析结论

1. **下山条件的作用**：
   - 保证 $|f(x_{n+1})| < |f(x_n)|$，函数值单调下降
   - 避免迭代点跳到更远处
   - 显著扩大收敛域
2. **代价分析**：
   - 每次下山需要额外的函数求值（平均 1-3 次）
   - 迭代次数可能增加，但保证收敛
   - 对不良初值，总体计算量远小于重新选择初值
3. **适用场景**：
   - 初值选择困难时
   - 函数性质复杂，如多峰、震荡
   - 需要算法鲁棒性时

6.3 初值敏感性分析

6.3.1 实验设置

函数：$f(x) = \sin(x) - \frac{x}{2}$，根约 $x^* \approx 1.8955$
测试不同初值对牛顿法（标准 vs 下山）的影响。

6.3.2 实验结果

表 6-7 不同初值下的收敛性

初值 x_0	标准牛顿法	牛顿下山法
0.5	发散	收敛（15 次）
1.0	发散	收敛（10 次）
1.5	收敛（6 次）	收敛（6 次）
2.0	收敛（4 次）	收敛（4 次）
2.5	收敛（5 次）	收敛（5 次）
3.0	震荡	收敛（12 次）
4.0	发散	发散

6.3.3 分析结论

1. **标准牛顿法**：
   - 收敛域：$x_0 \in [1.5, 3.0]$ 附近
   - 对初值敏感，离根较远易发散
   - 收敛时速度最快
2. **牛顿下山法**：
   - 收敛域显著扩大（约扩大 50%）
   - $x_0 \in [0.5, 3.0]$ 均可收敛
   - 远离根时迭代次数增加，但保证收敛
3. **实用建议**：
   - 初值选择困难时，优先使用下山法
   - 初值接近根时，标准法更高效
   - 可先用下山法靠近根，再切换标准法

6.4 算法适用性分析

6.4.1 不同函数类型的性能

表 6-8 算法在不同函数类型上的表现

函数类型	二分法	牛顿法	埃特肯法	弦截法	推荐
简单多项式	稳定	极快	快	快	牛顿法
超越方程	稳定	快（初值重要）	中等	快	弦截法
高次多项式	慢	快但易发散	中等	适中	牛顿下山
震荡函数	稳定	易失败	易失败	适中	二分法
多重根	慢	退化为线性	失效	退化	二分法

6.4.2 优缺点总结

* 注：其中 [✓] 表示优点，[✗] 表示缺点。

二分法：

✓ 绝对稳定，保证收敛

✓ 不需要导数

✓ 对多重根、震荡函数有效

✗ 收敛慢（线性）

✗ 需要提供区间

✗ 无法处理复根

牛顿法：

✓ 收敛最快（二次）

✓ 几何意义清晰

✗ 需要计算导数

✗ 初值敏感

✗ 多重根退化

► 下山改进：显著扩大收敛域

埃特肯法：

✓ 加速线性收敛序列

✓ 不需要导数

✓ 适合改造现有迭代法

✗ 需要三个点（初期慢）

✗ 对某些函数不稳定

弦截法：

✓ 无需导数

✓ 超线性收敛（$r \approx 1.618$）

✓ 综合性能优秀

✓ 实用性强

✗ 需要两个初值

✗ 收敛速度次于牛顿法

6.5 精度分析

6.5.1 机器精度的影响

对于函数 $f(x) = x^2 - 2$，理论根 $\sqrt{2}$ 的双精度表示为：

1.4142135623730951（精确到小数点后16位）

实验结果：

表 6-9 不同容差下的精度表现

设定容差	牛顿法迭代次数	最终误差	实际精度
10^{-3}	3	6.008e-04	达到要求
10^{-6}	5	2.220e-16	超过要求
10^{-9}	5	2.220e-16	受限机器精度
10^{-12}	5	2.220e-16	受限机器精度
10^{-15}	5	2.220e-16	达到极限

结论：

- 牛顿法在 5 次迭代后达到机器精度（$\approx 2.2 \times 10^{-16}$）
- 继续迭代无法提高精度
- 容差设置为 10^{-6} 至 10^{-10} 较为合理

6.5.2 误差来源分析

1. **截断误差**：算法本身的误差
    - 牛顿法：理论上二次收敛
    - 二分法：每步减半
2. **舍入误差**：浮点运算的误差
    - 累积影响：多次运算后放大
    - 减法相消：接近根时 $x_{n+1} - x_n$ 可能不准确
3. **函数求值误差**：
    - 超越函数本身有舍入误差
    - 复杂表达式误差累积

7. 实验结论

7.1 主要结论

1. **算法收敛性验证**：
   - 四种算法均成功实现，实验收敛阶与理论值一致
   - 二分法：线性收敛，$r \approx 1.0$
   - 牛顿法：二次收敛，$r \approx 2.0$
   - 埃特肯法：二次收敛，$r \approx 2.0$
   - 弦截法：超线性收敛，$r \approx 1.618$
2. **牛顿下山法的价值**：
   - 显著扩大收敛域（约 50%）
   - 对不良初值保证收敛
   - 仅增加少量计算成本（平均 2-3 次函数求值/步）
   - 推荐作为牛顿法的标准改进
3. **算法性能对比**：
   - **速度**：牛顿法 > 弦截法 > 埃特肯法 > 二分法
   - **稳定性**：二分法 > 弦截法 > 牛顿下山 > 牛顿法 > 埃特肯法
   - **易用性**：二分法 > 弦截法 > 牛顿法 > 埃特肯法
4. **可视化的重要性**：
   - 几何意义图直观展示算法原理
   - 收敛速度图清晰对比性能差异
   - 交互式探索加深理解
   - 下山过程可视化揭示算法稳定性机制

7.2 算法选择建议

根据问题特点选择算法：

1. **需要绝对稳定**：二分法
    - 适用场景：多重根、震荡函数、求区间内所有根
    - 代价：收敛慢
2. **追求极致速度**：牛顿法（标准）
    - 适用场景：初值接近根、导数易求、单根
    - 代价：初值敏感
3. **平衡速度与稳定**：牛顿下山法或弦截法
    - 牛顿下山：初值选择困难时
    - 弦截法：无法求导数时
    - 代价：略慢于标准牛顿法
4. **加速现有迭代**：埃特肯法
    - 适用场景：已有线性收敛迭代序列
    - 代价：初期慢，对某些函数不稳定

7.3 实践经验

1. **参数设置**：
    - 容差：10^{-6} 至 10^{-8} 适合大多数问题
    - 最大迭代次数：50-100，防止无限循环
    - 初值：尽量选择接近根的点，可通过图形估计
2. **收敛判据**：

- 同时检查相对误差和绝对误差：

$$|x_{n+1} - x_n| < \epsilon \quad \text{且} \quad |f(x_n)| < \epsilon$$

- 对于病态问题，适当放宽条件

3. **异常处理**：
   - 导数接近零：切换到弦截法
   - 迭代发散：启用下山条件或更换初值
   - 震荡不收敛：检查函数性质，考虑二分法
4. **工程实现**：
   - 模块化设计：算法、可视化、UI 分离
   - 错误处理：捕获除零、溢出等异常
   - 性能优化：避免重复计算函数值
   - 用户体验：提供预设函数、参数校验、实时反馈

7.4 进一步研究方向

1. **可视化增强**：
   - 三维函数的等高线图
   - 收敛域的动态展示
   - 算法动画的录制和回放
2. **性能优化**：
   - 自适应步长策略
   - 混合算法
   - 并行化处理多个初值

3. **应用拓展**：

    - 优化问题
    - 微分方程求解
    - 复根的可视化

8. 实验心得

8.1 技术收获

1. **算法理解深化**：

    - 通过编程实现，深刻理解了迭代求根算法的原理
    - 可视化展示使抽象的数学概念具象化
    - 实验验证了理论分析的正确性

2. **能力提升**：

    - 进一步熟悉 React 组件化开发方法
    - 学习 D3.js 专业可视化技术
    - 积累科学计算软件设计经验

3. **问题解决思路**：

    - 面对算法失败（如牛顿法发散），学会分析原因并寻找改进方案
    - 通过可视化快速定位问题
    - 培养了系统性能优化的意识

8.2 实验亮点

1. **牛顿下山法的完整实现**：

    - 不仅实现算法，还可视化下山过程

- 通过对比实验证明改进效果
- 为理解算法稳定性提供直观工具

2. **丰富的交互功能**：
   - 缩放拖动 + 悬浮提示
   - 设计独特—使用北理校徽特色双色设计
   - 动画播放结合步进控制
   - 数据导出、自定义函数
3. **几何意义可视化**：
   - 为每种算法设计专门的几何解释图
   - 帮助理解算法的几何本质

8.3 不足与改进

1. **当前不足**：
   - 未实现复根的可视化
   - 对病态问题（如多重根）处理不够完善
   - 大规模计算时性能有待优化
2. **改进方向**：
   - 添加区间分析功能，自动寻找合适初值
   - 根据函数特性推荐实现算法自动选择

9. 运行环境配置

Listing 2: 环境配置步骤

```
# 1. 安装Node.js (https://nodejs.org/)
# 版本要求: >= 16.0.0

# 2. 克隆项目
git clone https://github.com/Sophie618/numerical-analysis-platform.git
cd numerical-analysis-platform/numerical-visualizer

# 3. 安装依赖
npm install

# 4. 启动开发服务器
npm run dev

# 5. 浏览器访问
# http://localhost:5173
```

10. 测试用例

10.1 测试用例 1：求 $\sqrt{2}$

函数: `f(x) = x^2 - 2`

真实根: `1.414213562373095`

初值: `x0 = 1.5`

容差: `1e-6`

期望结果: 收敛于 `1.414214` 附近

10.2 测试用例 2：超越方程

函数: `f(x) = sin(x) - x/2`

真实根：1.895494267033981

初值：x0 = 2.0

容差：1e-6

期望结果：收敛于 1.895494 附近

10.3 测试用例 3：多重根

函数：f(x) = (x-1)^2

真实根：1.0（二重根）

初值：x0 = 2.0

容差：1e-6

期望结果：牛顿法退化为线性收敛，二分法稳定

11. 源代码仓库

完整源代码已开源至 GitHub：

仓库地址：https://github.com/Sophie618/numerical-analysis-platform

报告完成日期：2025 年 11 月

12. 总结陈述

本实验成功实现了数值分析中四种经典迭代求根算法，并构建了功能完善的可视化系统。通过对比实验，验证了算法的收敛性质，深入分析了各算法的优缺点和适用场景。特别是对牛顿下山法的研究，展示了算法改进的价值和方法。

实验不仅巩固了数值分析的理论知识，更提供了可视化系统，使抽象的数学算法变得直观易懂，具有良好的使用和体验感。

未来或将继续完善系统功能，探索更多数值方法的可视化实现，为数值分析的学习和研究提供更好的工具。