Numerical Computation Methods

Homework 2

学院: 数据科学与计算机学院

班级: 软工 3 班

姓名: 李天译

学号: 16340122

Contents

1 插值问题

1.1 问题描述

已知 $\sin 0.32 = 0.314567, \sin 0.34 = 0.333487, \sin 0.36 = 0.352274, \sin 0.38 = 0.370920$ 请采用线性插值、二次插值、三次插值分别计算 $\sin 0.35$ 的值

1.2 线性插值

1.2.1 算法设计

使用点斜式的方法，根据已知两点求出插值点

$$L_1(x) = y_k + \frac{y_{k+1} - y_k}{x_{k+1} - x_k}(x - x_k)$$

1.3 二次插值

1.3.1 算法设计

构造二次插值基函数 l_{k-1}, l_k, l_{k+1}

$$l_{k-1} = \frac{(x - x_k)(x - x_{k+1})}{(x_{k-1} - x_k)(x_{k-1} - x_{k+1})}$$

$$l_k = \frac{(x - x_{k-1})(x - x_{k+1})}{(x_k - x_{k-1})(x_k - x_{k+1})}$$

$$l_{k+1} = \frac{(x - x_{k-1})(x - x_k)}{(x_{k+1} - x_{k-1})(x_{k+1} - x_k)}$$

根据上面式子算出：

$$L_2(x) = y_{k-1}l_{k-1} + y_k l_k + y_{k+1}l_{k+1}$$

1.4 三次插值

1.4.1 算法设计

与二次插值同理，构造三次插值基函数

$$l_k(x) = \frac{(x-x_0)\cdots(x-x_{k-1})(x-x_{k+1})\cdots(x-x_n)}{(x_k-x_0)\cdots(x_k-x_{k-1})(x_k-x_{k+1})\cdots(x_k-x_n)}$$

其中，$k = 0, 1, \cdots, n.$

并根据上式计算拉格朗日插值多项式的值

$$L_n(x_j) = \sum_{k=0}^{n} y_k l_k(x_j) = y_j$$

其中，$j = 0, 1, \cdots, n.$

1.5 数值实验及结果分析

分别使用上述三种插值方法求插值结果，并与真实值对比

```
>> interpolation

ans =

[ 0.3616675, 0.3428971, 0.3428976]


ans =

0.3428978
```

上图向量值从左到右依次为一、二、三次插值的结果，数据值为函数在该

点真实值。

可以看出，三种插值得到的结果与真实值的误差随着插值次数增加而减少，三次插值时已经能达到很好的效果。

2 非线性方程的求解

2.1 问题描述

请采用下述方法计算 115 的平方根, 精确到小数点后六位。

(1) 二分法。选取求根区间为 $[10, 11]$

(2) 牛顿法

(3) 简化牛顿法

(4) 弦截法

绘出横坐标分别为计算时间、迭代步数时的收敛精度曲线。

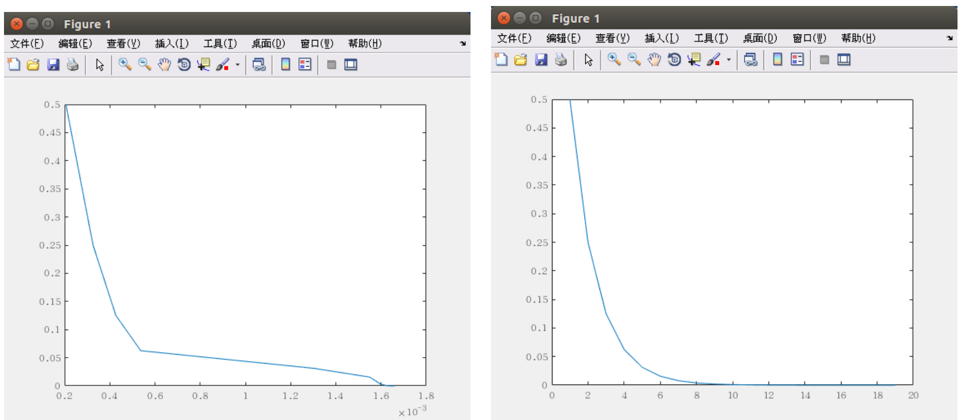
2.2 二分法

2.2.1 算法设计

- 准备 计算 $f(x)$ 在有根区间 $[a, b]$ 端点处的值 $f(a), f(b)$
- 二分 计算 $f(x)$ 在区间中点 $\frac{a+b}{2}$ 处的值 $f(\frac{a+b}{2})$
- 判断 若 $f(\frac{a+b}{2}) = 0$,则 $\frac{a+b}{2}$ 是根,计算结束,否则检验:若 $f(\frac{a+b}{2})f(a) < 0$，则以 $\frac{a+b}{2}$ 代替 b，否则以 $\frac{a+b}{2}$ 代替 a.

重复二分和判断的步骤，直到区间长度小于允许误差 ϵ，此时中点 $\frac{a+b}{2}$ 为近似根

2.2.2 数值实验



(a) time-converge (b) step-converge

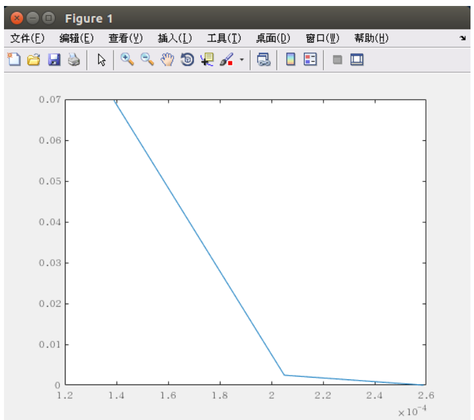
2.3 牛顿法

2.3.1 算法设计

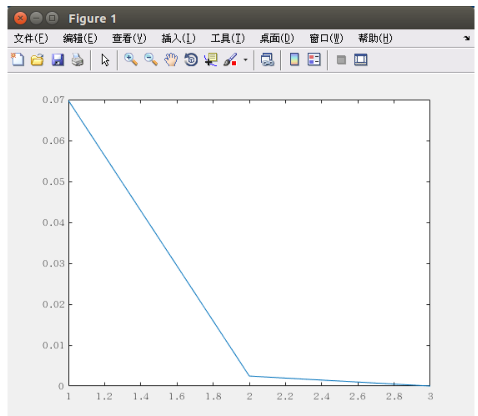
- 准备　选定初始近似值 x_0, 计算 $f_0 = f(x_0), f_0' = f'(x_0)$
- 迭代　按公式 $x_1 = x_0 - f_0/f_0'$ 迭代一次，得到新的近似值 x_1，计算 $f_1 = f(x_1), f_1' = f'(x_1)$
- 控制　如果 x_1 满足 $\frac{|x_1 - x_0|}{|x_1|}$ 时终止迭代，以 x_1 为所求根，否则转下一步

- 修改　如果迭代次数达到预先设定次数 N，或者 $f_1'=0$，则方法失败; 否则以 (x_1, f_1, f_1') 代替 (x_0, f_0, f_0') 转迭代步继续迭代

2.3.2 数值实验



(c) time-converge



(d) step-converge

2.4 简化牛顿法

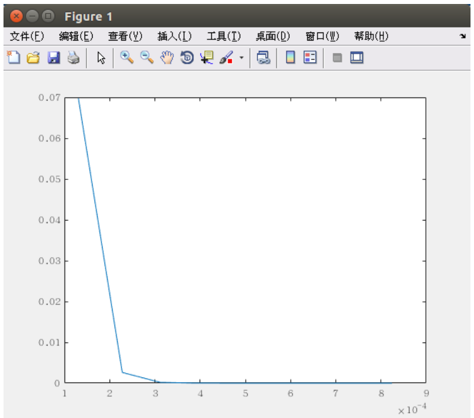
2.4.1 算法设计

用下面的式子取代牛顿法中迭代步的公式，即可得到简化牛顿法

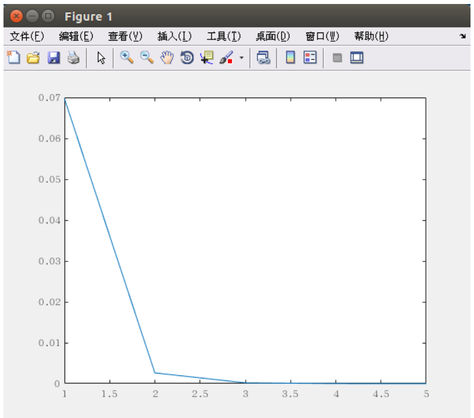
$$x_{k+1} = x_k - Cf(x_k)$$

其中，$C = \frac{1}{f'(x_0)}$

2.4.2 数值实验



(e) time-converge



(f) step-converge

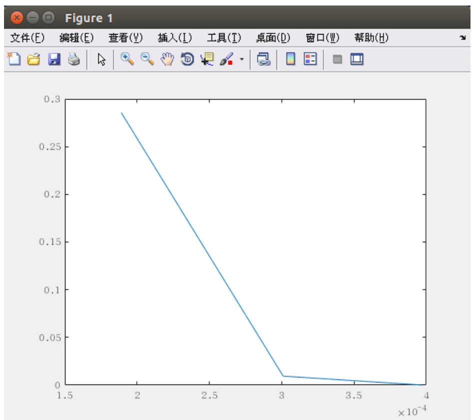
2.5 弦截法

2.5.1 算法设计

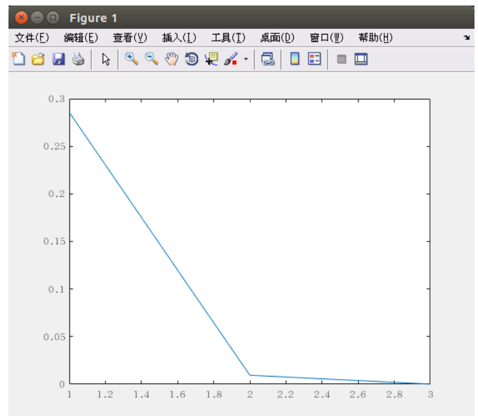
用下面的式子取代牛顿法中迭代步的公式，即可得到弦截法

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

2.5.2 数值实验



(g) time-converge



(h) step-converge

2.6 结果分析

```
>> square_root

ans =

[ 10.72380543, 10.72380529, 10.72380524, 10.72380529]


ans =

10.72380529
```

根据上述数值实验可以发现:

- 四种算法的收敛均先快后慢
- 四种算法中，相对而言，牛顿法和弦截法的计算精度更高，并且二者在计算代价上相差不大

- 二分法的收敛速度，计算时间显著劣于其他三种算法
- 根据理论，简化牛顿法在初始近似值的不合适选取，进而导致不收敛的问题上做了优化，算法的鲁棒性提升的同时，牺牲了少量步数和迭代时间

3 递推最小二乘 (RLS)

3.1 问题描述

请采用递推最小二乘法求解超定线性方程组 $Ax = b$, 其中 A 为 $m \times n$ 维的已知矩阵,b 为 m 维的已知向量,x 为 n 维的未知向量, 其中 $n = 10, m = 10000$。A 与 b 中的元素服从独立同分布的正态分布。绘出横坐标为迭代步数时的收敛精度曲线。

3.2 算法设计

$$tmp = (P0 \times A^T(k+1))/(1 + A(k+1) \times P0 \times A^T(k+1))$$

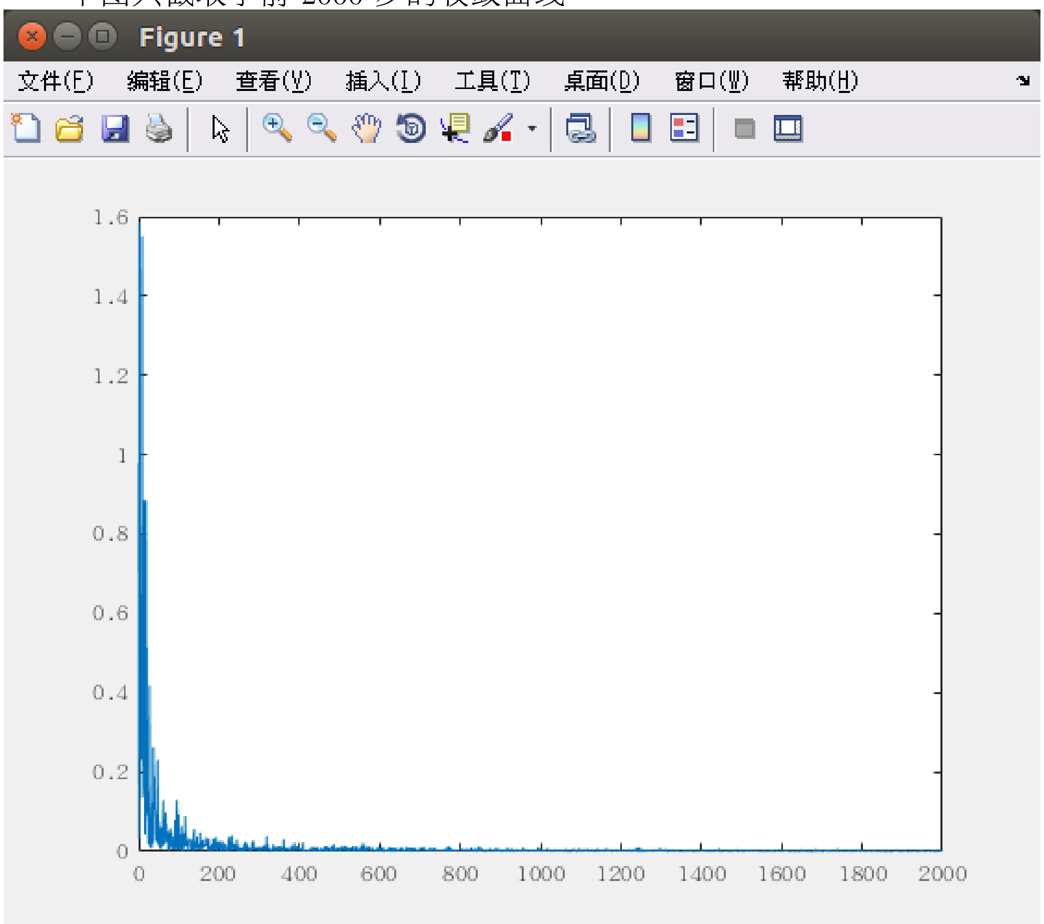
$$P1 = P0 - tmp \times A(k+1) \times P0$$

$$x1 = x0 + tmp * (b(k+1) - A(k+1) \times x0)$$

根据上述递推公式迭代求解，初始化 $P0$ 为一个较大值，$x0$ 为较小随机值。迭代 m 次后可以得到 m 个方程计算递推最小二乘法的结果

3.3 数值实验及结果分析

下图只截取了前 2000 步的收敛曲线。



在测试程序中，使用随机输入对函数进行测试，并与同样输入下的普通最小二乘法结果进行比较，可以发现二者在较高精度下完全一致，能够验证所实现 RLS 算法的正确性。

从实验结果可以看出，RLS 算法在收敛时波动幅度剧烈，这可以通过 RLS 递推过程中的逐渐加入新方程，更正原有的推测值得到一定的解释。

4 快速傅立叶变换 (FFT)

4.1 问题描述

请编写 1024 点快速傅里叶变换的算法。自行生成一段混杂若干不同频率正弦的信号, 测试所编写的快速傅里叶变换算法。

4.2 算法设计

使用分治法的思想，先将奇偶下标的数据分开，对每一部分计算傅立叶变换值，存储到对应位置完成合并。

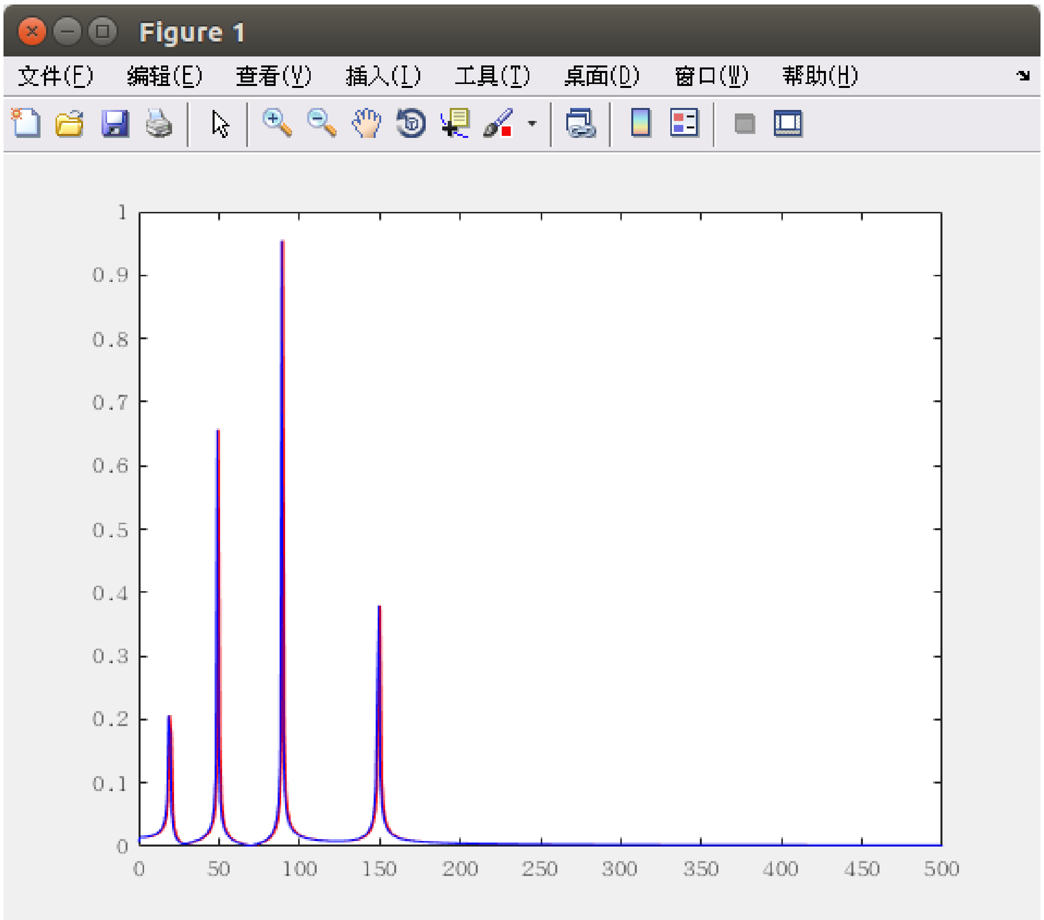
每部分的迭代公式如下:

$$c_{2j} = sum_{k=0}^{N/2-1}(x_k + x_{N/2+k})\omega_{N/2}^{jk}$$

$$c_{2j+1} = sum_{k=0}^{N/2-1}(x_k - x_{N/2+k})\omega_N^k\omega_{N/2}^{jk}$$

其中，$\omega_N = e^{i\frac{2\pi}{N}}$

4.3 数值实验及结果分析



使用 matlab 自带的 fft 函数对相同输入进行处理，可以看出二者结果差别不大，验证了所实现算法的正确性（其中红色线为 fft 函数，蓝色为 myFFT）

5 数值积分

5.1 问题描述

请采用复合梯形公式与复合辛普森公式, 计算 $\sin x/x$ 在 $[0,1]$ 范围内的积分。采样点数目为 5 9 17 33

5.2 复合梯形公式

5.2.1 算法设计

$$T_n = \frac{h}{2} sum_{k=0}^{n-1}[f(x_k) + f(x_{k+1}] = \frac{h}{2}[f(a) + 2sum_{k=1}^{n-1} f(x_k) + f(b)]$$

5.3 复合辛普森公式

5.3.1 算法设计

$$S_n = \frac{h}{6}[f(a) + 4sum_{k=0}^{n-1} f(x_{k+1/2}) + 2sum_{k=1}^{n-1} f(x_k) + f(b)]$$

5.4 数值实验及结果分析

```
ans =

[ 0.9445135217, 0.9456908636, 0.9459850299, 0.946058561]


ans =

[ 0.9460833109, 0.9460830854, 0.9460830713, 0.9460830704]


ans =

0.9460830704
```

可以看出，随着采样点数的增加，两种算法都逐渐接近真实值; 且可以看出，复合辛普森公式比复合梯形公式在相同条件下的效果更好

6 常微分方程初值问题

6.1 问题描述

请采用下述方法, 求解常微分方程初值问题 $y' = y - 2x/y, y(0) = 1$, 计算区间为 $[0, 1]$, 步长为 0.1

(1) 前向欧拉法。

(2) 后向欧拉法。

(3) 梯形方法。

(4) 改进欧拉方法

6.2 前向欧拉法

6.2.1 算法设计

$$y_{n+1} = y_n + hf(x_n, y_n)$$

6.2.2 数值实验

对应所求值如下：

```
cur =

  1 至 9 列

    1.0000    1.1000    1.1918    1.2774    1.3582    1.4351    1.5090    1.5803    1.6498

  10 至 11 列

    1.7178    1.7848
```

6.3 后向欧拉法

6.3.1 算法设计

使用欧拉公式给出迭代初值：

$$y_{n+1}^{(0)} = y_n + hf(x_n, y_n)$$

迭代代入下面的式子直到一定精度为止

$$y_{n+1}^{(k+1)} = y_n + hf(x_{n+1}, y_{n+1}^{k}), k = 0, 1, \cdots .$$

6.3.2 数值实验

对应所求值如下：

```
cur =

  1 至 9 列

    1.0000    1.0907    1.1741    1.2512    1.3231    1.3902    1.4529    1.5114    1.5658

  10 至 11 列

    1.6160    1.6618
```

6.4 梯形方法

6.4.1 算法设计

使用欧拉公式给出迭代初值：

$$y_{n+1}^{(0)} = y_n + hf(x_n, y_n)$$

迭代代入下面的梯形公式直到一定精度为止

$$y_{n+1}^{(k+1)} = y_n + \frac{h}{2}[f(x_n, y_n) + f(x_{n+1}, y_{n+1}^k)], k = 0, 1, \cdots.$$

6.4.2 数值实验

对应所求值如下：

```
cur =

  1 至 9 列

    1.0000    1.0957    1.1836    1.2654    1.3423    1.4151    1.4843    1.5504    1.6139

  10 至 11 列

    1.6751    1.7342
```

6.5 改进欧拉方法

6.5.1 算法设计

欧拉公式一步预测加上梯形公式一步校正即为改进的欧拉公式:

预测:

$$y_{n+1} = y_n + hf(x_n, y_n)$$

校正:

$$y_{n+1} = y_n + \frac{h}{2}[f(x_n, y_n) + f(x_{n+1}, y_{n+1})]$$

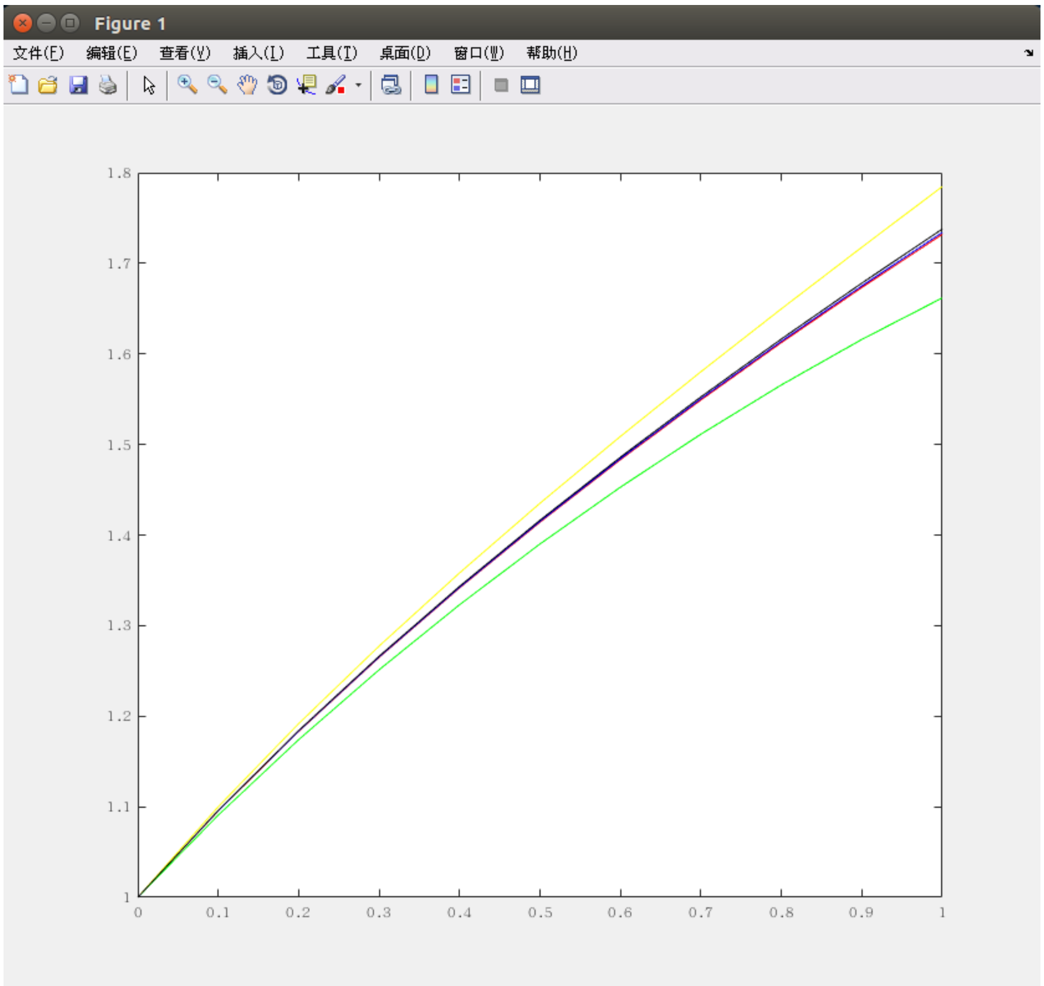
6.5.2 数值实验

对应所求值如下:

```
cur =

  1 至 9 列

    1.0000    1.0959    1.1841    1.2662    1.3434    1.4164    1.4860    1.5525    1.6165

  10 至 11 列

    1.6782    1.7379
```

6.6 结果分析

黄色为前向欧拉法，绿色为后向欧拉法，蓝色为梯形方法，黑色为改进欧拉法的图像，红色为真实值图像



对比可以看出，前向和后向欧拉方法均有偏差，但是两个不同方向的偏差，效果相差不大; 梯形方法和改进欧拉方法的效果很好，在改进欧拉方法中，实际上只是将梯形方法的内层迭代改为一次，但效果变化不大，且能提高计算效率，说明改进欧拉方法是一种解决此类问题很好的算法。