Matlab数值计算

Author : 郭少华

ID Numer:2021050264

Class: 应数2101

Date : 2023/7/20

第二章 插值法

2.1 Lagrange Interpolation

2.1.1 数学基础

给出$(x_i,y_i)(i=0,1,...,n),n+1$个节点，可以构造出次数不超过$n$的多项式$P_n(x)$

如两个节点$(x_0,y_0),(x_1,y_1)$构造线性插值多项式

$P_1(x)=y_0\frac{x-x_1}{x_0-x_1}+y_1\frac{x-x_0}{x_1-x_0}$

为一次多项式.

2.1.2 算法详解

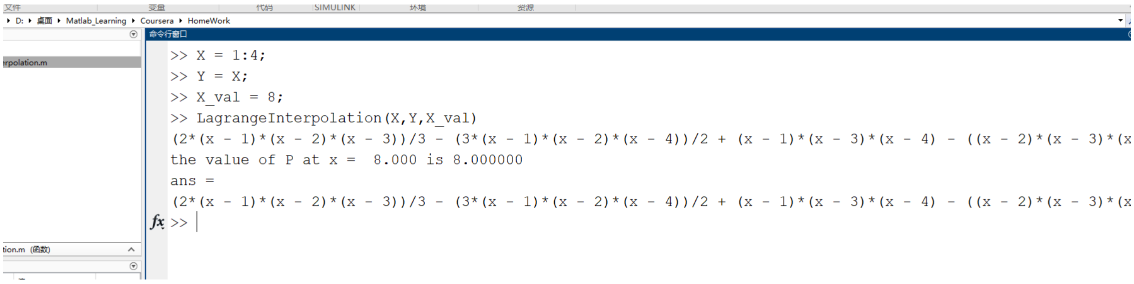
这个函数接收3个参数，第一个参数是给定节点的x坐标向量，第二个参数是给定节点的y坐标向量，第三个参数是希望根据插值多项式求出其他离散点的坐标点

函数返回两个结果，一个结果是插值多项式的表达式，注意，我们在显示表达式时，保留了拉格朗日插值多项式的原始形态，并未进行进一步的整理。另一个结果是需要求的其他离散的函数值。

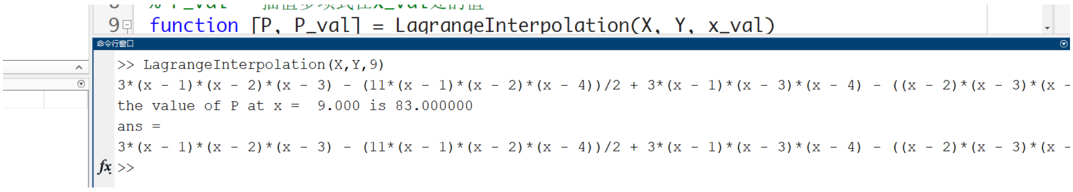
```
% LagrangeInterpolation 函数用于计算拉格朗日插值多项式及其在给定点的值。
% 输入参数：
% X - 插值点的x坐标，一个向量；特别注意，X的元素不可以相同！
% Y - 插值点的y坐标，一个向量
% x_val - 需要计算插值结果的x坐标点
% 输出参数：
% P - 拉格朗日插值多项式，一个符号表达式
% P_val - 插值多项式在x_val处的值
function [P, P_val] = LagrangeInterpolation(X, Y, x_val)

% 获取插值点的数量
n = length(X);

% 定义一个符号变量x
syms x;

% 初始化拉格朗日插值多项式为0
P = 0;

% 创建一个随机数向量作为系数的初始值
coff = rand(1,n);

% 对于每一个插值点
for i = 1:n
    % 计算拉格朗日插值多项式的系数
    coff(i) = Y(i) * 1/prod(X(i)-X([1:i-1,i+1:end]));

    % 计算每个拉格朗日基函数，即x减去其他所有插值点x坐标的连乘积
    product(i) = prod( x - X([1:i-1,i+1:end]));

    % 计算每个拉格朗日基函数与其对应系数的乘积
    term(i) = coff(i)*product(i);

    % 将所有项加起来得到拉格朗日插值多项式
    P = P + term(i);
end

% 显示拉格朗日插值多项式
disp(P);

% 计算并返回插值多项式在给定点x_val处的值
P_val = subs(P,x,x_val);

% 打印出插值结果
fprintf('the value of P at x =  %.3f is %f\n',x_val,P_val);

end
```

试验结果

1. $X=[1,2,3,4], Y=X, X_{val}=8$



2. $X=[1,2,3,4], Y=X^2+2, X_{val}=9$



可以看到，试验结果良好。

当然，我们可以对函数稍作修改，一次性计算多个离散点的函数值

```matlab
% LagrangeInterpolation 函数用于计算拉格朗日插值多项式及其在给定点的值。
% 输入参数：
% X - 插值点的x坐标，一个向量
% Y - 插值点的y坐标，一个向量
% x_val - 需要计算插值结果的x坐标点，一个向量
% 输出参数：
% P - 拉格朗日插值多项式，一个符号表达式
% P_val - 插值多项式在x_val处的值，一个向量
function [P, P_val] = LagrangeInterpolation(X, Y, x_val)

% 获取插值点的数量
n = length(X);

% 定义一个符号变量x
syms x;

% 初始化拉格朗日插值多项式为0
P = 0;

% 创建一个随机数向量作为系数的初始值
coff = rand(1,n);

% 对于每一个插值点
for i = 1:n
    % 计算拉格朗日插值多项式的系数
    coff(i) = Y(i) * 1/prod(X(i)-X([1:i-1,i+1:end]));

    % 计算每个拉格朗日基函数，即x减去其他所有插值点x坐标的连乘积
    product(i) = prod( x - X([1:i-1,i+1:end]));

    % 计算每个拉格朗日基函数与其对应系数的乘积
    term(i) = coff(i)*product(i);

    % 将所有项加起来得到拉格朗日插值多项式
```

```
    P = P + term(i);
end

% 显示拉格朗日插值多项式
disp(P);

% 计算并返回插值多项式在给定点x_val处的值，x_val可以是一个向量
P_val = subs(P,x,x_val);

% 打印出插值结果，需要处理x_val为向量的情况
for i = 1:length(x_val)
    fprintf('the value of P at x =  %.3f is %f\n',x_val(i),P_val(i));
end

end
```

用$X=[1,2,3,4],Y=X^2+3X,X_{val}=[5,6,7,8]$试验

```
>> LagrangeInterpolation(X,Y,[5,6,7,8])
(14*(x - 1)*(x - 2)*(x - 3))/3 - 9*(x - 1)*(x - 2)*(x - 4) + 5*(x - 1)*(x - 3)*(x - 4) - (2*(x - 2)*(x - 3)*(
the value of P at x =  5.000 is 40.000000
the value of P at x =  6.000 is 54.000000
the value of P at x =  7.000 is 70.000000
the value of P at x =  8.000 is 88.000000
ans =
```

可以看到，效果很理想。

2.2 Newton Interpolation

2.2.1 数学基础

牛顿插值多项式的公式为

$P_n(x)=a_0+a_1(x-x_0)+a_2+(x-x_0)(x-x_1)+...+a_n(x-x_0)(x-x_1)...(x-x_n),a_i$ 为$f(x)$关于前$i+1$个节点$(x_0,y_0),(x_1,y_1),...,(x_i,y_i)$的差商

2.2.2 算法分析

```
function [P, P_val] = NewtonInterpolation(X, Y, x_val)
    % 存储节点个数
    n = length(X);
    % 定义符号变量
    syms x;

    % 计算差商表

    %初始化差商表为n阶矩阵
    diff_table = zeros(n, n);
    % 差商表第一列为节点的Y值，存储为列向量
    diff_table(:,1) = Y';

    % 从第2列开始，差商表每一列的数值由第前一列的数值作差比上相应的x数值做作差
    % 从第2列开始计算，到第n列结束
    for j = 2:n
        % 在第j列，分别计算第j行到第n行的差
        for i = j:n
```

```
            diff_table(i,j) = (diff_table(i,j-1) - diff_table(i-1,j-1)) / (x(i) - x((i-1) - (j-2)); % 特别注意，这里的x( (i-1) + (j+2) )是这么找到的：在第j-1列，向左边移动 j-2列到达第1列，每移动1列，对应的x的索引就要减少1，于是x的索引就减少了 j-2，对应的x为x((i-1) - (j-2)) = x(i-j+1)
        end
    end

    % 构造牛顿插值多项式
    % a0 等于差商表的第一个元素，用它初始化插值多项式
    P = diff_table(1,1);
    for i = 2:n
    %取差商表的对角元为系数
        term = diff_table(i,i);
        for j = 1:i-1
            term = term * (x - x(j));
        end
        P = P + term;
    end

    % 计算插值多项式在给定的x_val处的值
    P_val = subs(P, x, x_val);
end
```

这个函数的构造采取了和拉格朗日插值函数同样的思路，接收三个参数：节点的X向量和Y向量，需要求值的离散点的横坐标，使用方法和拉个朗日函数相同

试验结果：

X = [1,2,3,4];Y = X; x_val =9;

```
>> NewtonInterpolation([1,2,3,4],[1,2,3,4],9)
the NewtonInterpolation polynomial is:f(x) = x

the value of P at x =  9.000 is 9.000000
```

第四章 数值微积分

4.1 数值积分

以下讨论的近似公式，均为$\int_a^b f(x)dx$的近似，约定步长$h = \frac{1}{n}(b-a)$

4.1.1 梯形公式

数学表达式：$\int_a^b f(x) \approx \frac{1}{2}(f(a)+f(b))$

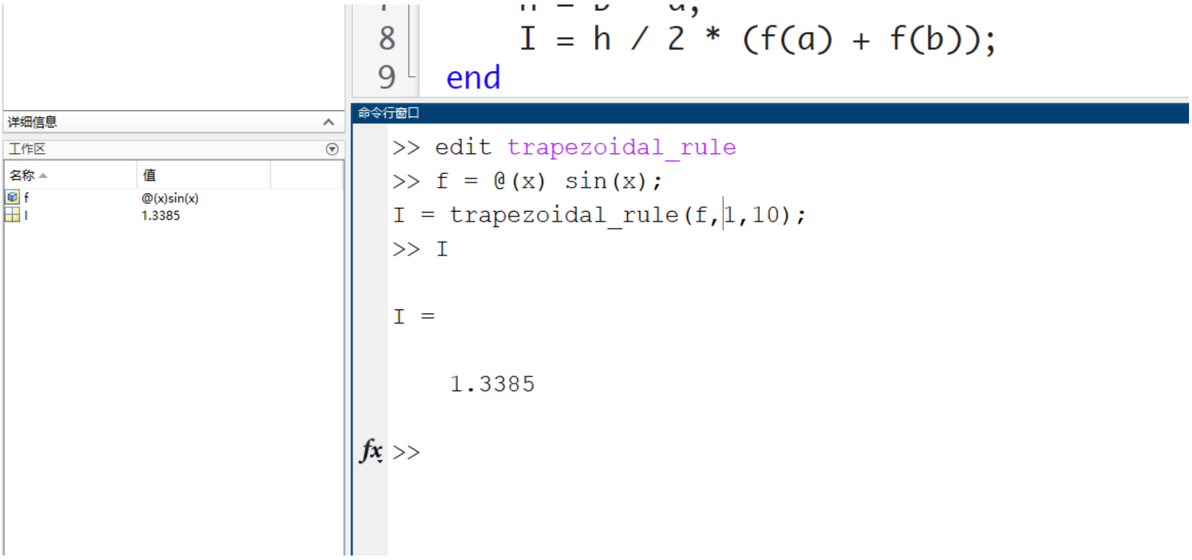
我们用$f(x) = sin(x)$ 试验；

```
% 定义梯形公式的函数
function I = trapezoidal_rule(f, a, b)
    % f: 被积函数
    % a: 积分下限
    % b: 积分上限

    h = b - a;
    I = h / 2 * (f(a) + f(b));
end
```

试验：

```
f = @(x) sin(x);
I = trapezoidal_rule(f,1,10);
```

结果：



我们再用牛顿莱布尼兹公式计算一遍：

```
True_value = cos(1) - cos (10)
```

计算结果为1.3794，误差为0.0408

```
    1.3385

>> True_value = cos(1) - cos (10)

True_value =

    1.3794

>> e = True_value - I

e =

    0.0408
```

我们在用matlab提供的积分函数计算一遍

```
quad(f,1,10)
```

结果也是1.3794

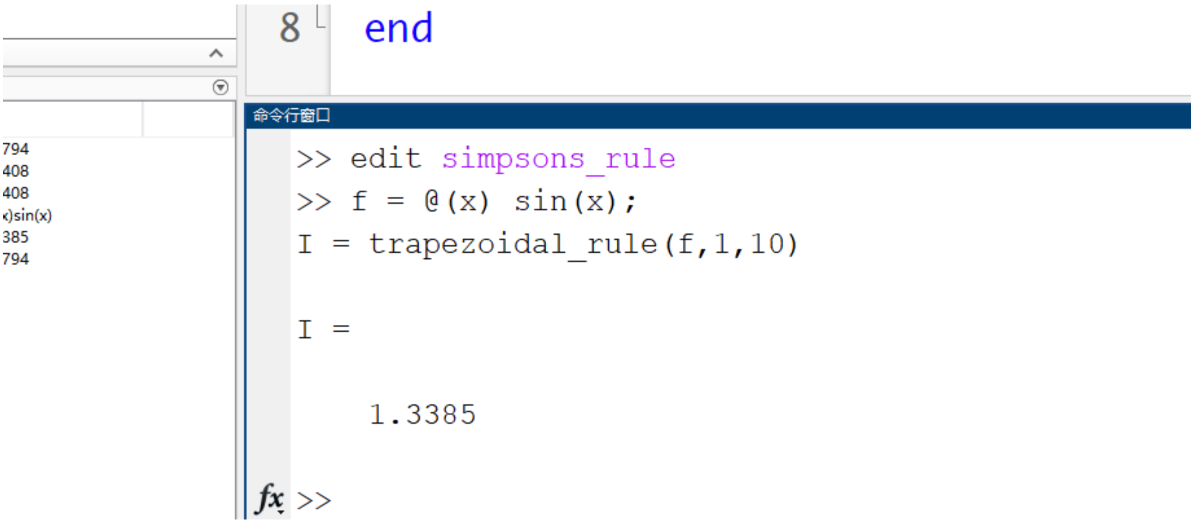
4.1.2 辛普森公式

```
function I = simpsons_rule(f, a, b)
    % f: 被积函数
    % a: 积分下限
    % b: 积分上限

    h = (b - a) / 2;
    I = h / 3 * (f(a) + 4*f(a + h) + f(b));
end
```

试验

```
f = @(x) sin(x);
I = trapezoidal_rule(f,1,10);
```

结果：



误差：

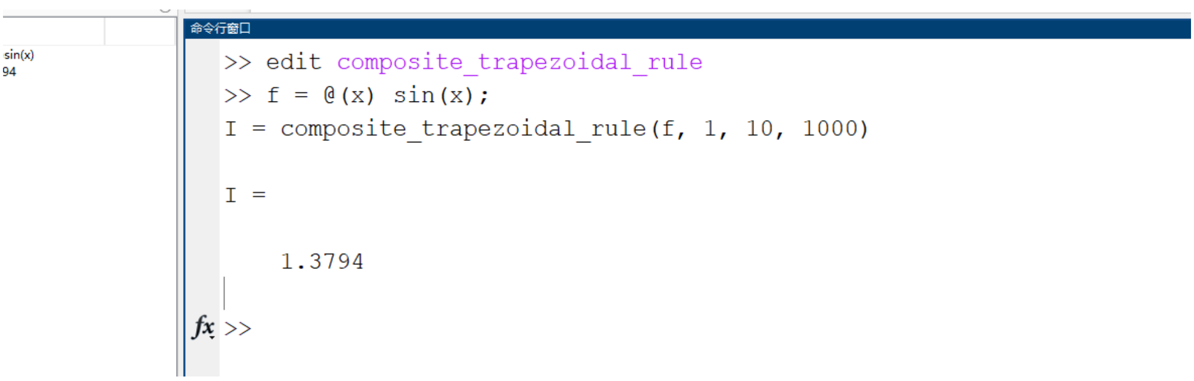
e = 1.3395 - 1.3794 = - 0.0399

4.1.3 复合梯形公式

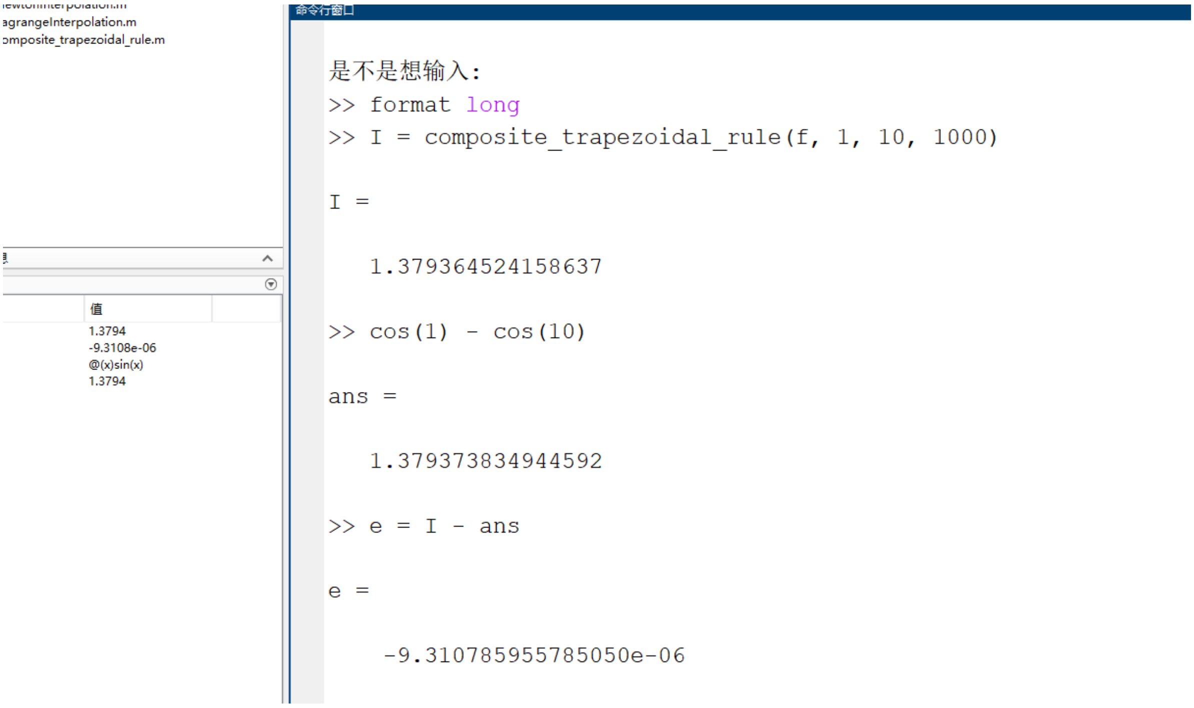
```
function I = composite_trapezoidal_rule(f, a, b, n)
    % f: 被积函数
    % a: 积分下限
    % b: 积分上限
    % n: 子区间个数

    h = (b - a) / n;
    x = a:h:b;
    fx = f(x);
    I = h / 2 * (fx(1) + 2*sum(fx(2:end-1)) + fx(end));
end
```

试验

```
f = @(x) sin(x);
I = composite_trapezoidal_rule(f, 1, 10, 1000);
```



结果：

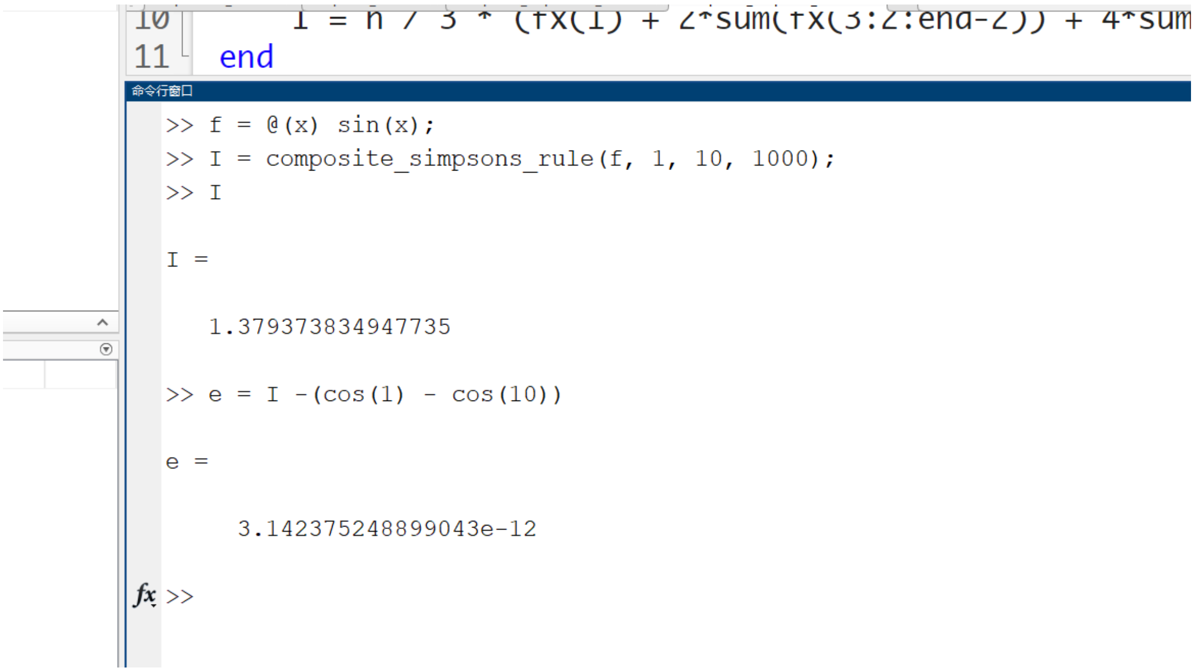


4.1.4 复合辛普森公式

```
function I = composite_simpsons_rule(f, a, b, n)
   % f: 被积函数
    % a: 积分下限
    % b: 积分上限
    % n: 子区间个数

    h = (b - a) / (2*n);
    x = a:h:b;
    fx = f(x);
    I = h / 3 * (fx(1) + 2*sum(fx(3:2:end-2)) + 4*sum(fx(2:2:end)) + fx(end));
end
```

试验

```
f = @(x) sin(x);
I = composite_simpsons_rule(f, 1, 10, 1000);
```



第七章 非线性方程组的数值解法

7.1 牛顿法

7.1.1 数学基础

我们可以用迭代公式找到非线性方程$f(x)=0$的近似解

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

牛顿法对迭代初值的选取要求严格，较好的迭代初值可以使得迭代格式迅速收敛到近似解，如果迭代初值选取不当，牛顿法可能发散

7.1.2 算法分析

为了使得函数具有普遍性，我们仍然采取和上面一样的思路，构造函数接收三个参数，一个是非线性多项式$f(x)$，一个是迭代初值x_0，一个是精度ε

其中，$f(x)$为符号函数

```
function root = newtonMethod(f, x0, epsilon)
    syms x;
    % 计算导数
    df = diff(f);

    % 将x0转换为符号变量
    x0 = sym(x0);

    % 初始化差值
    delta = Inf;

    while abs(delta) > epsilon
        % 计算f(x0)和df(x0)
        f_val = double(subs(f, x, x0));
        df_val = double(subs(df, x, x0));

        % 计算新的x值
        x1 = x0 - f_val / df_val;
```

```
        % 计算差值
        delta = abs(x1 - x0);

        % 更新x0
        x0 = x1;
    end

    root = x0;
end
```

调用函数，并且用$f(x) = sin(x), x_0 = 2$试验

```
syms x;
f = sin(x);
x0 = 2;
epsilon = 0.001;
root = newtonMethod(f, x0, epsilon);

```

我们查看结果，并且检查在迭代结果处的函数值

```
double(sin(root))
```

结果:

```
命令行窗口
>> syms x;
f = sin(x);
x0 = 2;
epsilon = 0.001;
root = newtonMethod(f, x0, epsilon);

disp(root);
28976077833147918747/9223372036854775808

>> sin(root)

ans =

sin(28976077833147918747/9223372036854775808)

>> double(sin(root))

ans =

  -9.1011e-11
```

可以看到，迭代精度非常高，说明牛顿法是一种高效的寻找近似解的方式。

7.2 不动点迭代

7.1.1 数学基础

$x_k = f(x_{k-1}) k = 1, 2, 3, \ldots$

收敛性：

若$f(x)$满足

1.$f(x) \in C(a, b)$

2.$\forall x \in [a, b], a \leq f(x) \leq b$

3. $\exists L < 1, \forall x, y \in [a, b], |f(x) - f(y)| \leq L|a - b|$

则对$\forall x_0 \in [a, b]$，由x_0得到的不动点迭代序列收敛

7.1.2 算法分析

这里我们采用和上面一样的思路构建函数，就不写那么多注释啦！

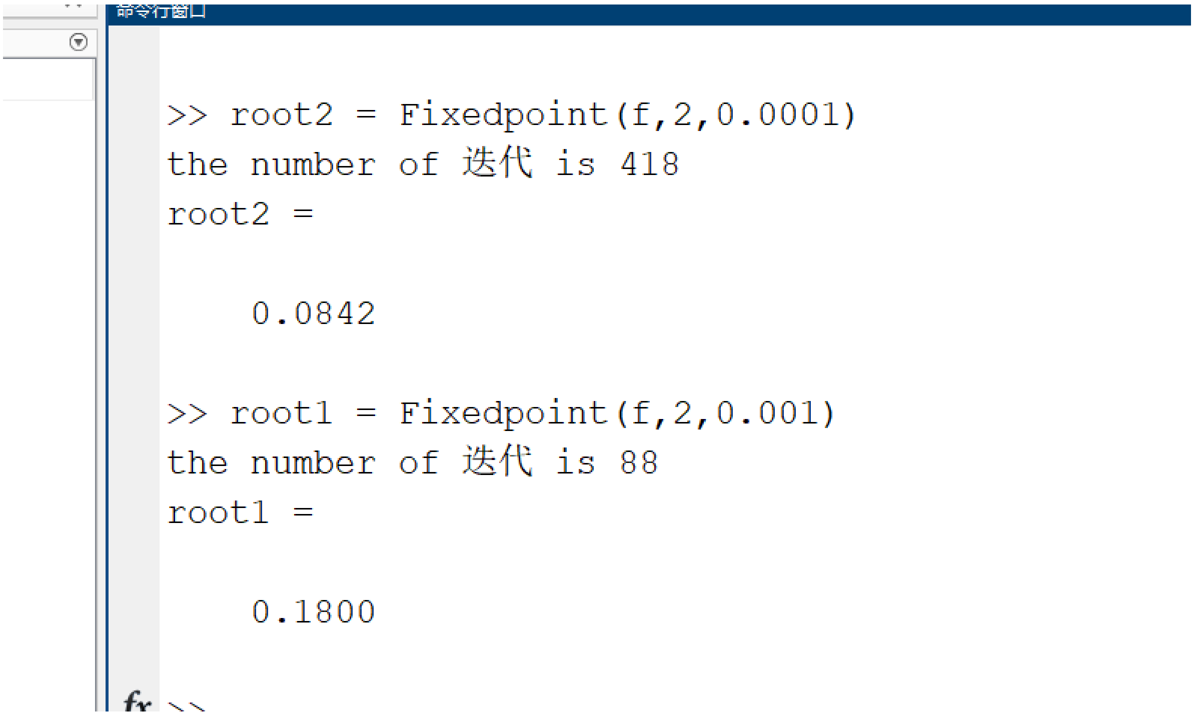
```
function root = newtonMethod(f, x0, epsilon)
    syms x;
    % 将x0转换为符号变量
    x0 = sym(x0);
    % 初始化差值
    delta = Inf;
    i = 0;%统计迭代次数
    while abs(delta) > epsilon
         f_val = double(subs(f, x, x0));
         %新的x值
         x1 = f_val;
         %计算差值
         delta = abs(x1 - x0);
         %更新x值
         x0 = x1;
    end
     fprintf("the number of 迭代 is %d",i);
    root = x0;
end
```

这次我们利用$f(x) = sin(x), x \in [-1, 1]$来测试

```
root = Fixedpoint(f,-1/2,0.001);
root2 = Fixedpoint(f,-1/2,0.0001);
```

试验结果：

!



(C:\Users\LENOVO\AppData\Roaming\Typora\typora-user-images\image-20230721092541117.png)

可以看到，对精度要求越高，结果越精确