牛顿法求解非线性方程

C++ 和 MATLAB 版
第一版

天天向尚磊
lei_tech@qq.com

2013.11.20

目录

1 牛顿法

以牛顿法解下面的非线性方程

$$f(x) = 0. \tag{1}$$

即以牛顿迭代式经过多次迭代得到近似解。其中，牛顿迭代式为

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}. \tag{2}$$

牛顿法的简略算法如下。在具体实现中只依据其思想，而不是具体的算法流程。

Newton's Method[1]

---

input $x_0, M, \delta, \varepsilon$
$v \leftarrow f(x_0)$
output $0, x_0, v$
if $|v| < \varepsilon$ **then stop**
for $k = 1$ **to** M **do**
    $x_1 \leftarrow x_0 - v/f'(x_0)$
    $v \leftarrow f(x_1)$
    output k, x_1, v
    if $|x_1 - x_0| < \delta$ **or** $|v| \leftarrow \varepsilon$ **then stop**
    $x_0 \leftarrow x_1$
end do

---

2 程序使用说明

牛顿法的实现加入异常处理，比如除数为零，主要体现算法本身。停止准则选取了三个：最大迭代次数限制、迭代值之间距离精度限制、函数值精度限制。其中，关于精度的限制可以改进为相对精度。

MATLAB 至少需要输入三个参数。最后两项参数分别为最大迭代次数和精度限制。三种有效调用形式：

- newton(f, df, x0)
- newton(f, df, x0, maxiter)
- newton(f, df, x0, maxiter, tol)

求解方程 $x^2 - 2 = 0$，调用方法如下。

[1] 数值分析 (原书第三版), David Kincaid and Ward Cheney, 王国荣等译, 机械工业出版社, page 64.

Listing 1: MATLAB EXAMPLE

```
% 2013/11/20 15:14:38

f = @(x)x^2 - 2;
df = @(x)2*x;
x0 = 3;
root = newton(f, df, x0);
```

C++ 以 C++ 实现的方法并未编写成为一般可调用的方法，而作为一个独立的文件（包含一个实例），修改部分即可求解对应的方程。具体参照 cpp 文件内注释。

A 附录

A.1 MATLAB

Listing 2: MATLAB CODE

```
function root = newton(f, df, x0, maxiter, tol)
%NEWTON Newton's method for nonlinear equations.
%
% NEWTON's method: x(k+1) = x(k) - f(x(k))/f'(x(k)).
%
% Inputs
% f - nonlinear equation.
% df - derivative of f(x).
% x0 - initial value.
% maxiter - maximum iterated times.
% tol - precision.
%
% Outputs
% root - root of f(x) = 0.

% 天天向尚磊
% Last Reversion: 2013/11/20 10:56:48
% email: lei_tech@qq.com

if nargin < 5, tol = eps; end
if nargin < 4, maxiter = 50; end

x_old    = x0;
x_iter   = x0 + 1;
times    = 0;

IfTrue
if true_sig
```

```
    fprintf(' k       iter\n');
    fprintf('--------+----------------\n');
    fprintf('%3.0f     | %.7f\n', 0, x_old);
end

tic
while true_sig
    x_iter = x_old - f(x_old)/df(x_old);
    times = times + 1;
    IfTrue
    x_old = x_iter;
    fprintf('%3.0f     | %.10f\n', times, x_old);
end
time = toc;

fprintf('\nIterated times is %g.\n', times);
fprintf('Elapsed time is %g seconds.\n', time);

root = x_iter;

% subfunction
    function IfTrue
        if times == 0
            true_sig = (times < maxiter) ...
                && (abs(x_old - x_iter) > tol) ...
                && (abs(f(x_old)) > tol);
        else
            true_sig = (times < maxiter) ...
                && (abs(x_old - x_iter) > tol) ...
                && (abs(f(x_iter)) > tol);
        end
    end

end
```

A.2 C++

Listing 3: C++ CODE

```
// 牛顿法求解非线性方程
// 作者：天天向尚磊
// 时间：2013-11-14 - 2013-11-18
// 版本：0.2

// 功能描述：求解非线性方程根，并输出最终解
// 迭代式：x(k+1) = x(k) - f(x(k))/df(x(k)).
// 使用：修改标出的“修改”部分即可自定义参数

// 输入：函数 fun，函数导数 dfun，初值 x0，
```

```cpp
// 最大迭代次数 maxiter，停止精度 tol
// 输出：迭代数值解 x_iter2


#include <iostream>
#include <cmath>
using namespace std;

// ----------- 修改 1 ------------
const double x0 = 2;                    // 迭代初值
const int maxiter = 20;                 // 最大迭代次数
const double tol = 1e-6;                // 精度限制
// ----- 参数设置修改这里 -------

// --------------- 修改 2 ----------------
double fun(double x) {return x*x -2;}   // 所求根的函数 x2^ - 2 = 0
double dfun(double x) {return 2*x;}     // 函数导数
// ----- 换用其它函数，修改此部分 -------

double newton(double x0, int maxiter, double tol);
                                        // 牛顿法函数原型

int main(int argc, char const *argv[])
{
    double root;

    cout << "\n k            iteration" << "\n";
    cout << "----------------------------------------------" << "\n";

    root = newton(x0, maxiter, tol);    // 调用牛顿法

    cout << "\nthe last iteration is " << root << "\n";

    return 0;
}


double newton(double x0, int maxiter, double tol)
{
    double x_iter1 = x0, x_iter2;
    bool true_sig;
    int times = 0;
    true_sig = (fun(x_iter1) > tol);    // 如果初值已使得函数值达到精度

    cout.precision(11);
    cout << " " << times << "       "<< x_iter1 << "\n";

    while(true_sig)
```

```
    {
        x_iter2 = x_iter1 - fun(x_iter1)/dfun(x_iter1);
                                                // 牛顿迭代式
        times ++;                               // 记录迭代次数
        true_sig = (times < maxiter) && (abs(fun(x_iter2)) > tol);
        true_sig = true_sig && (abs(x_iter2 - x_iter1) > tol);
                                                // 判断是否停止迭代
        x_iter1 = x_iter2;                      // 更新迭代值
        cout << " " << times << "       "<< x_iter1 << "\n";
    }

    return x_iter2;                             // 返回最终的迭代值
}
```