数値計算の理論と実際

第 2 回 非線形方程式の解法 1

1 変数の代数方程式は 5 次以上では解の公式が存在しないことが知られている (アーベルルフィニの定理)。一般的に非線形方程式 $f(x) = 0$ は特殊な場合を除いて、解析的に解を求めることは困難である。したがって数値的に近似値を求めることが多い。代表的な方法として **2 分法**と**ニュートン法**が存在する。

1　2 分法

ある連続な方程式 $f(x) = 0$ について、$f(a)$ と $f(b)$ が異符号である 2 点を考える。関数は連続であるため、a と b の間に $f(x) = 0$ を満たす点が少なくとも 1 つは存在する (**中間値の定理**)。2 分法はこの性質を利用する。

いま c として

$$c = \frac{a+b}{2} \tag{1}$$

を考える (a、b の中点)。解は少なくとも a と c の間か、c と b の間に存在する。そこで前者の場合は c を b とし、後者の場合には c を a と置き換える。こうして解が存在する範囲が半分に縮小される。これを繰り返すことで近似解を求めるのが **2 分法**である。

例えば

$$f(x) = x^3 - 3x^2 + 9x - 8 \tag{2}$$

を 2 分法で解いてみる。$f(1) = -1$、$f(2) = 6$ なので、中間値の定理により区間 [1,2] に解が存在することがわかる。

区間 [1,2] の中間値は 1.5 であり $f(1.5) = 2.125 > 0$ であることから、解は [1,1.5] にあることがわかる。次にこの区間の中間値 1.25 を考えるわけであるが、このプロセスをコンピュータ上で反復する。

```
f(1.5) = 2.12
f(1.25) = 0.51
f(1.125) = -0.2
f(1.1875) = 0.13
f(1.15625) = -0.05
.
.
f(1.165955) = 0.0003
```

15 回程繰り返すと解は [1.165894, 1.165955] に存在することがわかる。小数点以下 3 桁の精度でよければここで計算を打ちきる。数値計算であり厳密解を得ることは難しいので、通常は適当な収束条件を設定する。2 分法の場合は例えば区間幅が ϵ 以下になったら計算を打ちきる。

2 分法の一種として、c を中点ではなく、2 点 $(a,\ f(a))$,$(b,\ f(b))$ を結ぶ直線と x 軸の交点にすることもある。これを **Regula-Falsi 法**という。この時 c は

$$c = \frac{af(b) - bf(a)}{f(b) - f(a)} \tag{3}$$

となる (各自導出してみよ)。関数によっては収束がかなり速くなるが、遅くなることもある。

1.1 練習問題

式 (2) について、$f(x) = 0$ となる x を 2 分法で求めるプログラムを実装せよ。以下 C++ でのサンプルである。C で実装しても構わない。コンパイルは

```
g++ -Wall XXX.cpp
```

でできる (XXX.cpp はソースコードのファイル名)。また Regula-Falsi 法 で実装しさまざまな ϵ で収束性を比較せよ。

```
#include<iostream>
#include<cstdio>
#include<cmath>

using namespace std;

double func(const double x){

  関数の値を返す

}

int main(){

  double epsilon = 0.0;
  cerr << "Enter epsilon" << endl;
  cin >> epsilon;

  double a = 0.0;
  double b = 0.0;
  while(1){
    f(a) * f(b) <0 になるまで a, b の入力を繰り返す
  }

  int step = 0;
  while( b-a > epsilon){
    2分法の本体の実装
  }

}
```

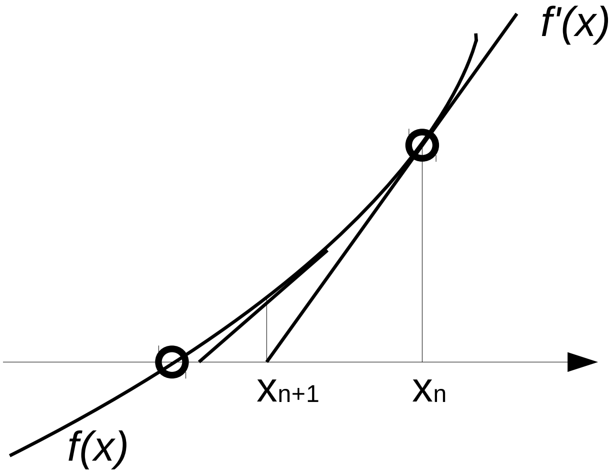

図 1: ニュートン法

2　ニュートン法

関数 $f(x)$ の微分 $f'(x)$ を利用するのがニュートン法である。x_n を第 n 番目の近似値としたとき、$(x_n, f(x_n))$ での接線と x 軸との交点を次の近似値 x_{n+1} とするものである (図 1)。接線の傾きは

$$f'(x_n) = \frac{0 - f(x_n)}{x_{n+1} - x_n} \tag{4}$$

である。したがって

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \tag{5}$$

である。初期値 x_0 を適当に決め、漸化式 (5) にしたがって x_n を修正する。$x_0 = 2$ からはじめると

$x_1 = 1.3333333...$
$x_2 = 1.1695906...$
$x_3 = 1.1659067...$
$x_4 = 1.165905...$

となる。

見てわかるようにニュートン法は収束が速いが、2 分法とは異なり、解が収束しないことがある。例えば

$$3\tan^{-1}(x-1) + \frac{x}{4} = 0 \tag{6}$$

は、$x_0 = 2$ の場合収束するが、$x_0 = 4$ の場合は収束しない。

ニュートン法ではこのように収束せず振動し、解が求まらないことがある。この時プログラムは無限ループに入ってしまう。それを防ぐために、通常反復の上限回数を設定する。

ニュートン法では微分を計算する必要があるが、微分を得ることが難しい場合もある。差分を利用することで直線の傾きを近似することができる。ある点 $(x_n, f(x_n))$ の接線の傾きは 1 つ前の反復値 $(x_{n-1}, f(x_{n-1}))$ を使って、

$$\frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}} \tag{7}$$

と近似できる。したがって式 (5) は、

$$x_{n+1} = x_n - \frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})} f(x_n) \tag{8}$$

となる。これを**セカント法**という。ニュートン法に比べ収束が遅く、初期値が 2 つ必要である。また 2 つの初期値は近いところにとるのがよい。

2.1 練習問題

式 (2) について、$f(x) = 0$ となる x をニュートン法、セカント法で求めるプログラムを実装せよ。以下 c++ でのサンプルである。

```
#include<iostream>
#include<cstdio>
#include<cmath>

using namespace std;

double func(const double x){

  関数の値を返す

}

double func_d(const double x){

  微分値の値を返す

}

int main(){

  double epsilon = 0.0;
  double x = 0.0;
  cerr << "Enter x0, epsilon" << endl;
  cin >> x >> epsilon;

  ニュートン法の本体の実装

}
```

3　演習問題

次の方程式を 2 分法、ニュートン法およびセカント法で解いて収束性を比較せよ。また初期値を変えて比較せよ。

$$
\begin{aligned}
&(1) \qquad \cos(x) - x^2 = 0 \\
&(2) \qquad e^x = \frac{1}{x} \\
&(3) \qquad 3\tan^{-1}(x-1) + \frac{x}{4} = 0
\end{aligned}
$$