代数方程式

数値計算 西谷　滋人

1　概要

代数方程式の解 (f(x)=0) を数値的に求めることを考える．標準的な**二分法 (bi-section method)** と**ニュートン法 (Newton method or Newton-Raphson method)** の考え方と例を説明し，**収束性 (convergency)** と**安定性 (stability)** について議論する．

二分法のアイデアは単純である．中間値の定理より連続な関数では，関数の符号が変わる二つの変数の間には根が必ず存在する．したがって，この方法は収束性は決して高くはないが，確実である．一方，ニュートン法は関数の微分を用いて収束性を速めた方法である．しかし，不幸にして収束しない場合があり，初期値や使用対象には注意を要する．これらの手法は非線形な関数に対しても用いることが出来る．

連立方程式 (f(x,y)=0,g(x,y)=0) を解くための一般的な優れた方法はない．しかし，根の近くの点が分かればニュートン法が威力を発揮する．

また，多項式の根を全て求めるには，因数分解を用いて多項式の次数を減らしていく減次という操作が必要となる．その原理となる組み立て除法を見た後，代表的な Bairstow-Hitchcock 法の導出を Maple で試みる．

2　二分法とニュートン法の原理

Maple では代数方程式の解は，fsolve で求まる．

$$f(x) = x^2 - 4x + 1 \tag{1}$$

の解を考える．(Maple script の fsolve を参照)　未知の問題では時として異常な振る舞いをする関数を相手にすることがあるので，先ずは関数の概形を見ることを常に心がけるべきである．また，解析解が容易に求まるなら，その結果を使うほうがよい場合がある．ニュートン法を視覚化して解説するコマンドを紹介しておく．

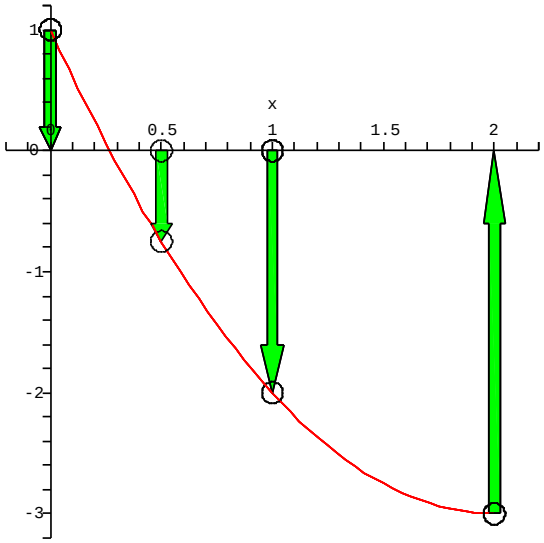

図 1: 二分法の原理の模式図.

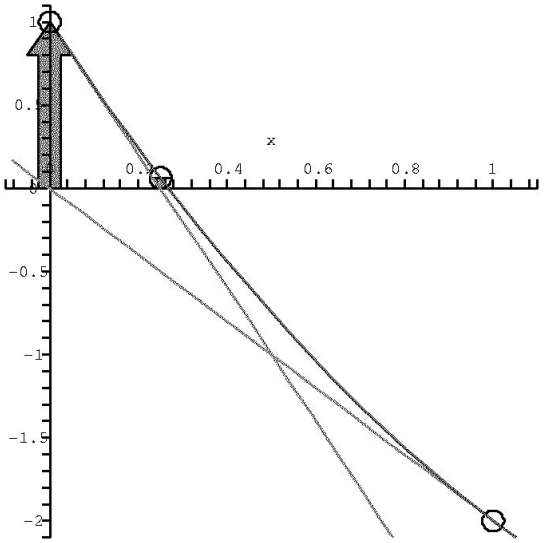

図 2: ニュートン法の原理の模式図.

3　収束性

解の収束の速さは，2 分法では一回の作業で解の区間が半分になる．このように繰り返しごとに誤差幅が前回の誤差幅の定数 (< 1) 倍になる方法は **1 次収束 (linear convergence)** するという．ニュートン法では (3.11) 式のごとく 2 次収束を示す．(Maple script の derivation of eq.(3.11) を参照)

注意しなければならないのは，この収束性には一回の計算時間の差は入っていないことである．ニュートン法で解析的に微分が求まらない場合，数値的に求めるという手法がとられるが，これにかかる計算時間はばかにできない．二分法を改良した割線法 (secant method) がより速い場合がある (NumRecipe9 章参照)．

4　C 言語による二分法とニュートン法のコード

二分法とニュートン法による代数方程式の解を求めるプログラムを示す．二分法でも実際のコードでは機械精度よりやや大きな eps を設定して，繰り返しを打ち切る．

リスト 1: 3.1 二分法

```
#include <stdio.h>

double func(double x);

int main(void){
  int i,imax=40;
  double x1=0.0,x2=2.0,f1,f2,f,x;
  f1=func(x1);
  f2=func(x2);
  printf("iter          x1             x2");
  printf("          (x1+x2)/2      f((x1+x2)/2)\n");
  for (i=1;i<=imax;i++){
    x=0.5*(x1+x2);
    f=func(x);
    printf("%4d %15.10f %15.10f %15.10f %15.10f\n",i,x1,x2,x,f);
    if (f*f1>=0.0){
      x1=x;f1=f;
    } else {
      x2=x;f2=f;
    }
  }
  return 0;
}

double func(double x){
```

```
  double func;
  func=x*x-4.0*x+1;
  return func;
}
```

リスト 2: 3.2 ニュートン法

```
#include <stdio.h>
#include <math.h>

double func(double x);
double dfunc(double x);

int main(void){
  double x=1.0,eps=1.0e-10,f0;
  int i,imax=40;
  f0=func(x);
  if (fabs(f0)<=eps){
    printf("%15.10f %15.10f\n",x,f0);
    printf("Accidentally finished.\n");
    return 0;
  }
  for (i=0;i<imax;i++){
    x=x-f0/dfunc(x);
    f0=func(x);
    printf("%4d %15.10f %15.10e\n",i,x,f0);
    if (fabs(f0)<=eps) {
      printf("Successfully converged.\n");
      return 0;
    }
  }
  printf("No solution after %4d iterations\n",i);
  return 1;
}

double func(double x){
  double func;
  func=x*x-4.0*x+1;
  return func;
}
double dfunc(double x){
  double dfunc;
  dfunc=2.0*x-4.0;
  return dfunc;
}
```

5　組み立て除法

多項式

$$P(x) = a_0x^n + a_1x^{n-1} + a_2x^{n-2} + \cdots + a_n \tag{2}$$

を $x - \alpha$ で割ったときの商 $Q(x)$ と余り R

$$P(x) = (x - \alpha)Q(x) + R \tag{3}$$

を考える．上式の両辺に α をいれれば，$P(\alpha) = R$，つまり余り R は $P(\alpha)$ で求まることは明らか．$Q(x)$ は，以下に示す組み立て除法で機械的に求めることが出来る．

$$\begin{array}{rcccccc}
 & & b_0x^{n-1} & & +b_1x^{n-2} & +b_2x^{n-3} & +\cdots \\
\cline{3-6}
x-\alpha &)a_0x^n & +a_1x^{n-1} & & +a_2x^{n-2} & +a_3x^{n-3} & +\cdots \\
 & b_0x^n & -\alpha b_0x^{n-1} & & & & \\
\cline{2-5}
 & [0] & (a_1+\alpha b_0)\,x^{n-1} & & +a_2x^{n-2} & & \\
 & & b_1x^{n-1} & & -\alpha b_1x^{n-2} & & \\
\cline{3-6}
 & & [0] & & (a_2+\alpha b_1)\,x^{n-2} & +a_3x^{n-3} & \\
 & & & & & \vdots &
\end{array} \tag{4}$$

それぞれの係数 a_n, b_n は，以下の関係を満たす．

$$\begin{aligned}
a_0 &= b_0 \\
a_1 + \alpha b_0 &= b_1 \\
a_2 + \alpha b_1 &= b_2 \\
&\vdots \\
a_{n-1} + \alpha b_{n-2} &= b_{n-1} \\
a_n + \alpha b_{n-1} &= P(\alpha)
\end{aligned} \tag{5}$$

プログラムでは a_n, α は既知であり，順番に b_n を求めていけばよい．逆にこうして余りを求める手法をホーナー法といい，乗算の回数が少なく効率がいいことが知られている．

6　Bairstow-Hitchcock 法の Maple による導出

Bairstow-Hitchcock 法の導出を試みる．多項式 $P(x)$ に根 a, b があるとすると，

$$(x - a)(x - b) = x^2 - (a + b)x + ab = x^2 + Bx + C$$

なる 2 次式の因数がある．これは複素共役な根 $a \pm bI$ がある場合にも成り立ち，

$$(x - a - bI)(x - a + bI) = x^2 - 2ax + (a^2 + b^2) = x^2 + Bx + C$$

となる．多項式 $Q(x)$ 等を導入して，$P(x)$ が

$$P(x) = (x^2 + Bx + C)Q(x) + Rx + S$$

と書き直せるとする．多項式 $P(x)$ に 2 次の因数 $x^2 + Bx + C$ が存在する場合は，$R = S = 0$ である．あるいは，$R = S = 0$ となる係数 B, C を求めることが，Bairstow-Hitchcock 法の基本である．

具体的な例として”Simplest explanation”に示したような多項式を考える．Bairstow-Hitchcock 法の基本的な考えかたは，”Analytical solution”に示したように二つの式 (P1 と P2) を等値して係数を決めていく．その反復アルゴリズムは”Numerical solution”に示したように，配列変数を巧妙に取ることによって，非常にシンプルにコード化することができる．

7 Fortran の文法

7.1 副プログラム

プログラムの一部を分けて作成し，ある部分を他の部分の下請けとして使う．この下請け部分にはその使い方から，サブルーチンと関数がある．関数は C 言語での関数と同じで，サブルーチンは C 言語では戻り値がない (void 型) 関数を意味する．Fortran では以下に示すように呼びだし方が違う．

リスト 3: 関数の書き方

```
関数の型名 function 関数名 (引数, 引数,... )
   関数の記述
関数名=計算式
end

real*8 function func(x)
real*8 x
func=x**2-4.0*x+1.0
end
```

```
f1=func(x1)
f2=func(x2)
```

図 3: 関数の呼びだし

リスト 4: サブルーチンの書き方

```
subroutine サブルーチン名 (引数, 引数,... )
   手続きの記述
end

subroutine force(fx0,fy0,...)
real*8 fx0,fy0
  ...
end
```

```
call force(fx0,fy0, ...)
```

図 4: サブルーチン呼びだし

- サブルーチン名は英数字 6 文字までで，先頭は英字にする．
- 文番号や変数名はプログラム単位ごとに独立．他の部分と同じ文番号や変数名を使ってよい．
- 型や配列の宣言はたとえ既に宣言したのと同じ内容であっても，再度宣言する必要がある．
- 引数によって呼びだし側と呼びだされる側のデータの受け渡しをする．計算結果をもとのプログラムに返すにも引数を用いる．入力用 (呼びだし側からサブルーチン) の引数と出力用 (サブルーチンから呼びだし側) の引数も文法上は区別しておらず，入出力兼用も許される．
- 対応する引数の型や配列の次元は一致させておく必要がある．
- 主プログラムは，一個に限る．主プログラムはプログラムの先頭に置き，その最後の行には end を置き，その後に副プログラムを置く．